Книга, выпускаемая по лицензии издательства Addison-Wesley, является каноническим описанием языка программирования Java. Структура книги и стиль изложения напоминают «библию» программистов на C — работу «Язык программирования C» Б. Кернигана и Д. Ричи. Эта аналогия тем более оправдана, что один из авторов «Языка программирования Java» (Д. Гослинг) известен как основной разработчик этого языка. Книга в равной степени может служить учебником и справочником по Java, а многочисленные упражнения позволят читателю попрактиковаться в использовании популярного языка программирования.

ВВЕДЕНИЕ
ГЛАВА 1 ПЕРВОЕ ЗНАКОМСТВО С JAVA
ГЛАВА 2 КЛАССЫ И ОБЪЕКТЫ
ГЛАВА 3 РАСШИРЕНИЕ КЛАССОВ
ГЛАВА 4 ИНТЕРФЕЙСЫ
ГЛАВА 5 ЛЕКСЕМЫ, ОПЕРАТОРЫ И ВЫРАЖЕНИЯ
ГЛАВА 6 ПОРЯДОК ВЫПОЛНЕНИЯ
ГЛАВА 7 ИСКЛЮЧЕНИЯ
ГЛАВА 8 СТРОКИ
ГЛАВА 9 ПОТОКИ
ГЛАВА 10 ПАКЕТЫ
ГЛАВА 11 ПАКЕТ ВВОДА/ВЫВОДА
ГЛАВА 12 СТАНДАРТНЫЕ ВСПОМОГАТЕЛЬНЫЕ СРЕДСТВА
ГЛАВА 13 ПРИМЕНЕНИЕ ТИПОВ В ПРОГРАММИРОВАНИИ
ГЛАВА 14 СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ
ПРИЛОЖЕНИЕ А Родные методы
ПРИЛОЖЕНИЕ Б Runtime-исключения в Java
ПРИЛОЖЕНИЕ В Полезные таблицы

Далее

ИЗДАТЕЛЬСКИЙ ДОМ ПИТЕР® WWW.PITER.COM
© 1997-2002 Издательский дом "Питер". Авторские права охраняются.
Предназначено только для частного использования!
Воспроизведение материалов или частей данной книги в любом виде без письменного разрешения Издательского дома "Питер" запрещено!

Книга посвящается команде Java,
чей упорный труд и умение предвидеть будущее
вырастили могучее дерево.
Посвящается Сьюзен - К. А.
Посвящается Джуди и Кейт - Дж. Г.

Об авторах этой книги

Кен Арнольд, ведущий инженер Sun Microsystems Laboratories, является одним из экспертов в области объектно-ориентированных технологий. Он много пишет о программировании на C и C++ для UNIX Review. Кен Арнольд - автор книги «A C User's Guide to ANSI C».

Джеймс Гослинг, член совета директоров и вице-президент Sun Microsystems, является создателем языка Java и одним из самых известных программистов в современном компьютерном мире. В 1996 году ему была присуждена премия «Programming Excellence Award». Ранее он участвовал в разработке NeWS (Network-extensible Window System, сетевой расширяемой оконной системы компании Sun), а также в проекте Andrew в университете Карнеги-Мэллон, где получил докторскую степень.

ПРЕДИСЛОВИЕ

Красивые здания - нечто большее, чем создание науки. Это настоящие живые организмы, постигаемые на духовном уровне; это произведения искусства, в которых современная технология подчиняется вдохновению более, нежели вопросам вкуса и соображениям житейской пользы.
Фрэнк Ллойд Райт

Язык программирования Java (в дальнейшем именуемый просто Java) был тепло встречен мировым сообществом программистов и провайдеров Internet-услуг. Его преимущество для пользователей Internet и World Wide Web заключается в появлении безопасных, платформонезависимых приложений, которые могут использоваться в любом узле Internet. Программисты, создающие приложения на Java, пишут свой код всего один раз - им не приходится «переносить» свои приложения на все возможные программные и аппаратные платформы.

Многие рассматривают Java в первую очередь как средство создания *аплетов* для World Wide Web. Термином «аплет» в Java обозначается мини-приложение, работающее внутри Web-страницы. После того как аплет загружен на компьютер пользователя, он может выполнять определенные задачи и взаимодействовать с пользователем через броузер, не требуя ресурсов Web-сервера. Разумеется, некоторые аплеты могут взаимодействовать с сервером для решения своих внутренних задач, но это их личное дело.

Java является действительно ценным инструментом разработки в распределенных сетевых средах, к которым относится World Wide Web. Тем не менее Java не ограничивается этими рамками и представляет собой мощный универсальный язык программирования, подходящий для создания самых разнообразных приложений, которые либо не зависят от сетевых средств, либо используют их нестандартным образом. Способность Java безопасно выполнять код на удаленных хост-машинах оказалась очень кстати во многих крупных организациях.

Кроме того, Java используется как язык программирования общего назначения для реализации тех проектов, в которых платформенная независимость не так важна. Простота программирования и средства безопасности позволяют быстро создавать отлаженный код. Некоторые распространенные в других языках ошибки в программах на Java вообще не встречаются благодаря таким средствам, как сборка мусора и ссылки, безопасные по отношению к типам. Современные приложения, работающие в сетевых условиях и применяющие графический пользовательский интерфейс, которым приходится одновременно обслуживать несколько задач, используют поддержку многопоточности в Java, а *механизм исключений* облегчает обработку ошибок. Несмотря на всю мощь своих встроенных средств, Java - это *простой* язык, который быстро осваивается программистами.

Язык Java проектировался с расчетом на максимальную переносимость и на минимальную зависимость от конкретной системы. Например, во всех реализациях Java тип int представляет собой 32-разрядное число со знаком, дополняемое по модулю 2, независимо от архитектуры процессора, на котором выполняется Java-программа. Определение всех стандартов, касающихся языка и его runtime-среды, позволяет выполнять скомпилированный код в каком угодно месте и переносить его на любую машину, где присутствует среда Java.

Java обладает многими языковыми средствами, присущими большинству современных языков программирования. Тем не менее, в отличие от C и C++, Java автоматизирует хранение переменных и обработку исключений, дополняя их поддержкой многопоточности.

Об этой книге

Книга обучает программированию на Java и рассчитана на читателей, знакомых с основными концепциями программирования. Язык Java рассматривается в ней без претензий на формальность или полноту описания. Книга не является пособием по объектно-ориентированному программированию, хотя некоторые его аспекты рассматриваются для выработки общей терминологии.

Программистам на C и C++ язык Java должен показаться знакомым, поскольку в нем используются многие конструкции этих языков. Другие книги данной серии, а также большая часть онлайновой документации посвящены программированию аплетов (см. ссылки, приведенные в разделе «Библиография»).

Глава 1 - «Первое знакомство с Java» - содержит краткий обзор Java. Программистам, не владеющим концепциями объектно-ориентированного программирования, следует внимательно прочитать эту главу, а для остальных она станет полезным введением в объектно-ориентированные возможности Java.

В главах 2, 3 и 4 рассматриваются фундаментальные аспекты объектно-ориентированного программирования в Java, а именно объявления классов, их реализация и создание объектов на основе определений классов. Глава 2 - «Классы и объекты» - описывает основы языка Java. Глава 3 - «Расширение классов» - посвящена расширению, или *субклассированию*, существующих классов, в результате которого появляются новые классы со своими данными и другим поведением. Глава 4 - «Интерфейсы» - рассказывает о том, как объявляются интерфейсы, представляющие собой абстрактные описания поведения объектов, обеспечивающие максимальную гибкость для проектировщиков и программистов.

В главах 5 и 6 рассматриваются стандартные языковые конструкции, общие для большинства языков программирования. Глава 5 - «Лексемы, операторы и выражения» - посвящена лексемам языка, его операторам, их использованию для построения выражений и процессу вычислений. Глава 6 - «Порядок выполнения» - показывает, каким образом управляющие операторы изменяют последовательность выполнения операторов в программе.

Глава 7 - «Исключения» - посвящена очень мощному средству Java - обработке исключений. Глава 8 - «Строки» - описывает встроенные языковые и runtime-средства для работы с объектами String.

В главе 9 - «Потоки» - объясняется, как в Java реализована многопоточность. Многие приложения (в частности, те, что основаны на графическом интерфейсе пользователя) должны одновременно обслуживать несколько задач. Чтобы обеспечить их правильное функционирование, необходимо организовать надлежащее взаимодействие между ними. Потоки Java отвечают таким требованиям.

Глава 10 - «Пакеты» - описывает механизм группировки классов Java в отдельные пакеты.

В главах с 11 по 14 рассматриваются основные пакеты, входящие в библиотеку классов Java. Глава 11 - «Пакет ввода/вывода» - описывает систему ввода/вывода. Глава 12 - «Стандартные вспомогательные средства» - посвящена *вспомогательным классам (utility classes)*, к которым относятся векторы и хеш-таблицы. В главе 13 - «Применение типов в программировании» - рассматриваются классы Java, относящиеся к таким типам, которые представляют собой как отдельные объекты, описывающие класс или интерфейс, так и классы, служащие «оболочками» для примитивных типов данных (в частности, целых и вещественных значений). Глава 14 - «Системное программирование» - объясняет, как получить доступ к системным средствам.

Приложение А показывает, каким образом в Java поддерживаются *родные методы (native methods)* - средства для работы с кодом, написанным на «родном» языке программирования базовой платформы.

В приложении Б перечисляются все runtime-исключения и ошибки, возбуждаемые самой системой Java.

Приложение В содержит ряд полезных таблиц, которые пригодятся для справочных целей.

Наконец, в разделе «Библиография» приведены ссылки, которые могут представлять интерес при дальнейшем знакомстве с объектно-ориентированным программированием, многопоточностью и другими темами.

Примеры и документация

Все примеры были откомпилированы и запущены с помощью последней версии языка (FCS-версия Java 1.0.2), существовавшей на момент написания книги. Вообще говоря, нами рассматривается язык Java 1.0.2. Мы также уделили внимание некоторым аспектам, выходящим за пределы написания безошибочно компилируемых программ, - просто пользоваться языком недостаточно, нужно делать это *правильным образом*. Мы попытались объяснить, что такое хороший стиль программирования.

В ряде мест встречаются ссылки на онлайновую документацию. Среда разработки Java позволяет автоматически генерировать файл справки (обычно в формате HTML) по откомпилированному классу с помощью документирующих комментариев. Для просмотра таких файлов обычно используется Web-броузер.

Ни одна техническая книга не пишется «на необитаемом острове», а в нашем случае уместней было бы говорить о целом континенте. Множество людей помогало авторам полезными советами, точными рецензиями, ценной информацией и рекомендациями.

Редактор Генри Мак-Гилтон (Henry McGilton) из Trilithon Software содействовал разрешению всех проблем и внес немалый вклад в создание книги. Редактору серии Лайзе Френдли (Lisa Friendly) мы обязаны беззаветной настойчивостью и поддержкой.

Многие рецензенты выделили свое драгоценное время на чтение, редактирование, пересмотр и удаление материала, и все это ради улучшения книги. Кевин Койл (Kevin Coyle) написал одну из самых подробных рецензий. Карен Беннет (Karen Bennet), Майк Бурати (Mike Burati), Патриция Гинке (Patricia Giencke), Стив Гильяр (Steve Gilliard), Билл Джой (Bill Joy), Розанна Ли (Rosanna Lee), Джон Мэдисон (Jon Madison), Брайан О'Нейл (Brian O'Neill), Сью Палмер (Sue Palmer), Стивен Перелгат (Stephen Perelgut), Р. Андерс Шнайдерман (R. Anders Schneidermann), Сьюзен Сим (Susan Sim), Боб Спраул (Bob Sproull), Гай Стил (Guy Steele), Артур Ван Хофф (Arthur Van Hoff), Джим Уолдо (Jim Waldo), Грег Уилсон (Greg Wilson) и Энн Уолрат (Ann Wollrath) представили нам свои содержательные рецензии. Джефф Арнольд (Geoff Arnold), Том Карджилл (Tom Kargill), Крис Дэрк (Chris Darke), Пэт Финнеган (Pat Finnegan), Майк Джордан (Mike Jordan), Дуг Ли (Doug Lea), Рэндалл Мюррей (Randall Murray), Роджер Риггс (Roger Riggs), Джимми Торрес (Jimmy Torres), Артур Ван Хофф (Arthur Van Hoff) и Фрэнк Йеллин (Frank Yellin) предоставили полезные комментарии и техническую информацию.

Алка Дешпанд (Alka Deshpande), Шерон Фланк (Sharon Flank), Нассим Фотухи (Nassim Fotouhi), доктор К. Калаянасундарам (Dr. K. Kalayanasundaram), Патрик Мартин (Patrick Martin), Пол Романья (Paul Romagna), Сьюзен Снайдер (Susan Snyder) и Николь Янкелович (Nicole Yankelovich) своими совместными усилиями сделали возможными пять слов на стр. , не относящихся к кодировке ISO-Latin-1. Джим Арнольд (Jim Arnold) предоставил информацию о правильном написании, использовании и этимологии слов smoog и moorge. Эд Муни (Ed Mooney) помог подготовить документацию. Херб (Herb) и Джой Кайзер (Joy Kaiser) были нашими консультантами по хорватскому языку. Куки Каллахан (Cookie Callahan), Роберт Пирс (Robert Pierce) и Рита Тавилла (Rita Tavilla) помогли сдвинуть проект с мертвой точки, когда он собирался остановиться всерьез и надолго.

Благодарим Ким Полез (Kim Polese) за краткую формулировку причин, по которым язык Java важен как для пользователей, так и для программистов.

В критические моменты нам помогали своими советами и поддержкой Сьюзен Уолдо (Susan Waldo), Боб Спраул (Bob Sproull), Джим Уолдо (Jim Waldo) и Энн Уолрат (Ann Wollrath). Спасибо нашим семьям, которые не только дарили нас своей любовью, но и не дали нам зачахнуть за работой, за что мы им глубоко благодарны.

Выражаем искреннюю признательность персоналу фирмы Peets Coffee and Tea, который снабжал нас лучшим кофе Java на всей планете.

Любые ошибки или недочеты, оставшиеся в этой книге (несмотря на все усилия перечисленных выше лиц), лежат исключительно на совести авторов.

Глава 1
ПЕРВОЕ ЗНАКОМСТВО С JAVA

Основные технологии Java, версии и спецификации
Среда разработки Java
Первая программа на Java

Если язык программирования Java является для вас первым знакомством, то Вы по-настоящему начали изучать язык программирования. Поэтому в первой главе мы будем знакомиться с основными понятиями, которые вам необходимо знать для дальнейшего изучения языка.

1.1. С самого начала

Язык программирования Java был разработан компанией Sun Microsystems. Его разработчиком является известный специалист программирования по имени Джеймс Гослинг. Язык Java был разработан для использования в сетевом программировании. Его отличительной особенностью является то, что он позволяет разрабатывать как обычные, так и сетевые приложения.

Язык Java является объектно-ориентированным языком. Поэтому каждая программа, которая будет разрабатываться на этом языке, будет иметь определенную объектную ориентацию. Но, прежде чем мы начнем знакомиться с объектной ориентацией, мы рассмотрим ряд вопросов, которые вам необходимо знать для дальнейшего изучения языка.

Основной единицей языка Java является слово. Словом называется последовательность символов, ограниченная пробелами. Пробелами называются такие символы, как пробел, перевод строки, табуляция и другие. Пробелы в программе игнорируются компилятором Java, если они не входят в состав какой-нибудь лексемы.

Лексемой называется минимальная единица языка, которая имеет для компилятора какой-то смысл. Лексемы могут быть разделены пробелами, а могут и не быть разделены пробелами. Например, лексема public может быть отделена от лексемы class пробелом, а может и не быть отделена пробелом, если после лексемы class идет, например, лексема myClass.

Есть несколько видов лексем:

- ключевые слова
- идентификаторы
- константы
- операции
- разделители

Ключевые слова (keywords) имеют для компилятора специальное значение. Например, ключевое слово class для компилятора означает, что сейчас будет описываться класс. Ключевые слова не могут использоваться в качестве имен переменных, классов, методов.

Идентификаторы используются для обозначения имени переменной, метода, класса, интерфейса, пакета. Например, в строке

```
int myVar = 0;
```

myVar является идентификатором.

Константы могут быть различных типов. Например, константа 0 имеет тип int, константа 0.0 имеет тип float, константа true относится к типу boolean, а константа "Java" относится к типу String.

Операции используются для выполнения определенных действий над переменными. Например, в строке

```
int myVar = 0;
```

знак равенства является операцией.

Разделители используются для разделения одной лексемы от другой. Например, в строке

```
int myVar = 0;
```

разделителями являются пробелы, которые отделяют лексему int от лексемы myVar, а также точка с запятой, которая отделяет лексему myVar от лексемы 0.

Все лексемы, которые могут быть использованы в программе на языке Java, определяются компилятором Java. Если компилятор встретит какую-то лексему, которая ему не знакома, то он выдаст ошибку. Например, если вы напишите

```
int myVar = 0;
```

то компилятор выдаст сообщение о том, что переменная myVar не определена.

Каждая лексема имеет определенный тип. Например, лексема int имеет тип primitive type, лексема class имеет тип class or interface, лексема myVar имеет тип identifier, лексема 0 имеет тип integer literal, лексема = имеет тип assignment operator, лексема ; имеет тип separator.

Каждый язык имеет определенный синтаксис. Например, английский язык имеет синтаксис, который позволяет составлять предложения из слов. Аналогично, язык Java имеет свой синтаксис, который позволяет из лексем составлять выражения. Например, выражение

```
int myVar = 0;
```

состоит из лексем int, myVar, =, 0, ;.

Каждый язык имеет определенный алфавит. Например, английский язык имеет алфавит, который состоит из 26 букв. Аналогично, язык Java имеет свой алфавит, который состоит из следующих символов:

- буквы латинского алфавита (A-Z, a-z)
- цифры (0-9)
- знаки подчеркивания (_)
- знаки доллара ($)

Из этих символов формируются лексемы. Например, из букв latin формируется слово latin. Из цифры 0 формируется лексема 0. Из знака подчеркивания формируется лексема _. Из знака доллара формируется лексема $.

Каждый язык имеет определенные "грамматические" конструкции для разбора слов, которые называются лексическими правилами. Например, в английском языке есть такое лексическое правило, что слово может состоять из нескольких слов, разделенных дефисом. Например, слово mother-in-law состоит из трех слов, разделенных дефисом. Аналогично, в языке Java есть такое лексическое правило, что лексема может состоять из нескольких лексем, разделенных пробелами.

Лексемы могут быть различных типов. Например, лексема int имеет тип primitive type, лексема class имеет тип class or interface, лексема myVar имеет тип identifier, лексема 0 имеет тип integer literal, лексема = имеет тип assignment operator, лексема ; имеет тип separator.

Каждый язык имеет определенный синтаксис. Например, английский язык имеет такой синтаксис, который позволяет из слов составлять предложения. Аналогично, язык Java имеет свой синтаксис, который позволяет из лексем составлять выражения. Например, выражение

```
int myVar = 0;
```

состоит из лексем int, myVar, =, 0, ;.

Каждый язык имеет определенный алфавит. Например, английский язык имеет такой

```
select Сотрудник=convert(char(30),Фамилия) ,
       Начальник=convert(char(30),
       (select Фамилия
        from Сотрудники
        where Код_сотрудника=
        (select Руководитель
         from Сотрудники
         where Фамилия='Иванов'))),
go
```

Если мы формируем сложное условие поиска, то обычно это приводит к ошибкам. Сложность состоит, как правило, в том, что мы не можем сразу сформулировать нужное условие, а пытаемся его подобрать, перебирая различные варианты. Для того, чтобы не приходилось каждый раз вносить изменения в запрос, можно использовать переменные. Переменные в Transact-SQL обозначаются так же, как и в Microsoft Access, с помощью символа @. Переменные могут использоваться в любом месте запроса, в том числе в условиях поиска. Для присвоения переменной значения используется оператор set. Переменные могут использоваться для передачи параметров в хранимые процедуры. Переменные могут использоваться для передачи параметров в хранимые процедуры. Переменные могут использоваться для передачи параметров в хранимые процедуры

Глава 8
СТРОКИ

Теперь пришло время рассмотреть, что же хранит компьютер?
Данные Мы, Вы, Пользователи.

Строка в Java является стандартным объектом и от встроенной изначальной подвержена. Пока рано встречать значения при проверкой строк, как объектов, но в дальнейшем, при изучении объектов, Вы сможете использовать свои знания. Для начала, рассмотрим, как же с строками работает компьютер. Для начала, рассмотрим, как же с строками работает компьютер. Строка в Java хранится в объекте String. Строка в StringBuffer, а также преобразование строк в другие типы данных и обратно. Также рассмотрим, как же с строками работает компьютер.

8.1. Основные операции со строками

Класс String позволяет работать со строками. Строка представляет собой последовательность символов. Строка хранится в специальном объекте. Строка в Java является стандартным объектом. Строка в Java хранится в объекте String. Строка в StringBuffer, а также преобразование строк в другие типы данных и обратно. String.

Для создания строки в виде литерала строки (String) можно использовать следующий синтаксис:

public String[]

Конструктор новой строки String, значение которого равно "".

public String(String value)

Конструктор новой строки String, инициализируемой копией аргумента.

Пример. Строка хранится в специальном объекте. Строка хранится в специальном объекте. Литерал строки можно использовать для создания объектов String. Литерал строки можно использовать для создания объектов String. Литерал строки можно использовать для создания объектов String.

```
char ch[] = {'a', 'b', 'c'};
String s = new String(ch);
```

Класс String позволяет работать со строками. Строка хранится в специальном объекте. Строка хранится в специальном объекте. Литерал строки можно использовать для создания объектов String. Литерал строки можно использовать для создания объектов String. Литерал строки можно использовать для создания объектов String.

Строка есть, является строка хранится или является строка хранится или является последовательность символов или строка хранится. Строка хранится в специальном объекте. Строка хранится в специальном объекте. Литерал строки можно использовать для создания объектов String.

```
static int countOccurrences(String str, char ch) {
    int length = str.length();
    int count = 0;
    for (int i = 0; i < length; i++)
        if (str.charAt(i) == ch)
            count++;
    return count;
}
```

Для этой цели хранит переход к переходу к переходу символа. Строка в Java. Если символ входит в строку начиная с указанного индекса, то метод возвращает индекс, с которого начинается вхождение. Если же такого символа в строке нет, то метод возвращает -1. Если такого символа в строке нет, то метод возвращает -1. Если такого символа в строке нет, то метод возвращает -1.

Для проверки принадлежности строки одному из нескольких вариантов. Строка в Java. Если символ входит в строку начиная с указанного индекса, то метод возвращает индекс, с которого начинается вхождение. Если же такого символа в строке нет, то метод возвращает -1. Если такого символа в строке нет, то метод возвращает -1. Если такого символа в строке нет, то метод возвращает -1.

indexOf(char ch)	первое вхождение ch
indexOf(char ch, int start)	первое вхождение ch, начиная с start
indexOf(String str)	первое вхождение str
indexOf(String str, int start)	первое вхождение str, начиная с start
lastIndexOf(char ch)	последнее вхождение ch
lastIndexOf(char ch, int start)	последнее вхождение ch, начиная с start
lastIndexOf(String str)	последнее вхождение str
lastIndexOf(String str, int start)	последнее вхождение str, начиная с start

Упражнение 8.1

Напишите метод, который подсчитывает количество вхождений данного символа в строку.

Упражнение 8.2

Напишите метод, который подсчитывает количество вхождений некоторой подстроки в данную строку.

8.2. Сравнение строк

В книге Гостер является использовать для сравнения строк и в сравнении частей. Не менее, между тем нет операции и сравнения подстрок. Для сравнения подстрок. Для сравнения подстрок. Для

Глава 12
ПРИМЕНЕНИЕ ТИПОВ В ПРОГРАММИРОВАНИИ

В данной главе:
Основные сведения о типах
Использование типов в Microsoft .Net
Разрыв типа либо его полное
Время Type Time

Типы в каждом языке задаются своим набором. Почти для каждого из применяемых типов (int, boolean и т.д.) существует подходящий класс, определяемый в библиотеке классов Microsoft .Net. Эти классы определяют, что можно делать с каждым из типов, а также как ведут себя эти типы при выполнении. Среди прочих преимуществ такого подхода можно выделить следующие:

- Поскольку стандартные классы для каждого типа хранятся в библиотеке, типы в каждом из них определяются, поскольку типы представляют собой объекты, а не просто фрагменты памяти, то в каждом из них можно вызывать методы, определенные в классе, соответствующем типу.
- Для каждого из типов в библиотеке классов .Net определяется, что можно сделать с каждым из типов, а также как ведут себя эти типы при выполнении. Поскольку типы определяются в библиотеке классов, можно создавать объекты обобщения, задействованные в любом из языков, которые поддерживают .Net, а также использовать эти объекты для обмена данными между приложениями.

Иерархия типов для этих классов выглядит следующим образом:

Класс для этой иерархии унаследует поведение от своего вышестоящего главного объекта по наследованию множества свойств. Все объекты, определяемые в библиотеке классов, являются производными от объекта Object. Каждый из классов в иерархии определяет свойства, которые являются уникальными для каждого из типов, определяемых в библиотеке классов .Net. Каждый из классов в иерархии определяет свойства, которые являются уникальными для каждого из типов, определяемых в библиотеке классов .Net.

В данной главе показано, как использовать объекты библиотеки классов .Net для определения и использования объектов типа, которые поддерживают каждый из языков, использующих библиотеку классов .Net. Вначале мы рассмотрим, как определяется каждый из типов, а затем рассмотрим, как использовать каждый из типов.

11.1. Класс Class

Для каждого класса в библиотеке классов имеется подкласс производный от класса Class. Этот объект может использоваться для получения информации о классе, определяемом в библиотеке классов. Класс Class определяется следующим образом:

Класс Class позволяет перебирать по каждому типу в библиотеке. Также можно формировать фабрику для каждого типа, определяемого в библиотеке классов. Класс Class также определяет ряд свойств, которые можно использовать для определения, что представляет собой каждый из типов, определяемых в библиотеке классов.

Следующие свойства используются чаще всего:

Следующие свойства используются для перечисления по каждому типу. Следующий метод класс выводит такую информацию для каждого типа:

```
public class TypeDemo
    public static void Main(string[] argv)
        Type[] types = Type.GetTypes();
        int i = 0;
        foreach (Type t in types)
            Console.Writeline("{0}: {1}", i, t);
            if (t.IsClass)
                Console.Writeline("  IsClass");
            if (t.IsEnum)
                Console.Writeline("  IsEnum");
            if (t.IsPublic)
                Console.Writeline("  IsPublic");
            if (t.IsSealed)
                Console.Writeline("  IsSealed");
            if (t.IsValueType)
                Console.Writeline("  IsValueType");
            if (t.IsArray)
                Console.Writeline("  IsArray");
            if (t.IsPrimitive)
                Console.Writeline("  IsPrimitive");
            i = i + 1;
        int[] myint = new int[5];
        int[,] myint2 = new int[3, 4];
        int[,,] myint3 = new int[2, 3, 4];
        bool[] mybool = new bool[10];
        bool[,] mybool2 = new bool[5, 5];
        Console.Writeline("myint: {0}", myint.GetType());
        Console.Writeline("myint2: {0}", myint2.GetType());
        Console.Writeline("myint3: {0}", myint3.GetType());
        Console.Writeline("mybool: {0}", mybool.GetType());
        Console.Writeline("mybool2: {0}", mybool2.GetType());
        int[] myint = GetInts();
        int[,] myint2 = GetInts();
        int[,,] myint3 = GetInts();
        bool[] mybool = GetBools();
        bool[,] mybool2 = GetBools();
        Console.Writeline("myint: {0}", myint.GetType());
        Console.Writeline("myint2: {0}", myint2.GetType());
        Console.Writeline("myint3: {0}", myint3.GetType());
        Console.Writeline("mybool: {0}", mybool.GetType());
        Console.Writeline("mybool2: {0}", mybool2.GetType());
    }
    int[] GetInts();
    int[,] GetInts();
    int[,,] GetInts();
    bool[] GetBools();
    bool[,] GetBools();
}
```

Данный пример программирует каждый тип, определяемый в библиотеке классов, и выводит для каждого из них на экран его индекс и полное имя. Также выводится информация о том, является ли каждый из типов классом, перечислением, простым, закрытым, общим, значимым или примитивным. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

```
class core.utils.dataobjects
    namespace Core.Lang.Objects
        namespace Core.Lang.Objects
            namespace Core.Lang.Objects
```

Данное в пример программирует следующие выходные потоки. Для примера:

```
myint: System.Int32[]
myint2: System.Int32[,]
myint3: System.Int32[,,]
mybool: System.Boolean[]
mybool2: System.Boolean[,]
```

Класс Type также определяет следующие свойства, которые позволяют определять, что представляет собой каждый из типов, определяемых в библиотеке классов. Следующие свойства используются чаще всего:

Следующие свойства используются для определения, что каждый из типов, определяемых в библиотеке классов, является простым, закрытым, общим, значимым или примитивным. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

Следующие свойства используются чаще всего:

Следующие свойства используются для определения, что каждый из типов, определяемых в библиотеке классов, является перечислением, является ли каждый из типов классом, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

```
class core.utils.dataobjects
    namespace Core.Lang.Objects
        namespace Core.Lang.Objects
            namespace Core.Lang.Objects
```

Данный пример программирует следующие выходные потоки. Для примера:

```
myint: System.Int32[]
myint2: System.Int32[,]
myint3: System.Int32[,,]
mybool: System.Boolean[]
mybool2: System.Boolean[,]
```

Данное в пример программирует следующие выходные потоки. Для примера:

Класс Type определяет ряд свойств, которые позволяют определять, что представляет собой каждый из типов, определяемых в библиотеке классов. Следующие свойства используются чаще всего:

Следующие свойства используются для определения, что каждый из типов, определяемых в библиотеке классов, является простым, закрытым, общим, значимым или примитивным. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

Следующие свойства используются чаще всего:

Следующие свойства используются для определения, что каждый из типов, определяемых в библиотеке классов, является перечислением, является ли каждый из типов классом, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

```
class core.utils.dataobjects
    namespace Core.Lang.Objects
        namespace Core.Lang.Objects
            namespace Core.Lang.Objects
```

Данный пример программирует следующие выходные потоки. Для примера:

```
myint: System.Int32[]
myint2: System.Int32[,]
myint3: System.Int32[,,]
mybool: System.Boolean[]
mybool2: System.Boolean[,]
```

Данное в пример программирует следующие выходные потоки. Для примера:

Класс Type определяет ряд свойств, которые позволяют определять, что представляет собой каждый из типов, определяемых в библиотеке классов. Следующие свойства используются чаще всего:

Следующие свойства используются для определения, что каждый из типов, определяемых в библиотеке классов, является простым, закрытым, общим, значимым или примитивным. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

Следующие свойства используются чаще всего:

Следующие свойства используются для определения, что каждый из типов, определяемых в библиотеке классов, является перечислением, является ли каждый из типов классом, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

11.2. Класс Object

Для каждого класса в библиотеке классов имеется подкласс производный от объекта Object. Этот объект может использоваться для получения информации о классе, определяемом в библиотеке классов. Класс Object определяется следующим образом:

Класс Object позволяет перебирать по каждому типу в библиотеке. Также можно формировать фабрику для каждого типа, определяемого в библиотеке классов. Класс Object также определяет ряд свойств, которые можно использовать для определения, что представляет собой каждый из типов, определяемых в библиотеке классов.

Следующие свойства используются чаще всего:

Следующие свойства используются для перечисления по каждому типу. Следующий метод класс выводит такую информацию для каждого типа:

```
public class TypeDemo
    public static void Main(string[] argv)
        Type[] types = Type.GetTypes();
        int i = 0;
        foreach (Type t in types)
            Console.Writeline("{0}: {1}", i, t);
            if (t.IsClass)
                Console.Writeline("  IsClass");
            if (t.IsEnum)
                Console.Writeline("  IsEnum");
            if (t.IsPublic)
                Console.Writeline("  IsPublic");
            if (t.IsSealed)
                Console.Writeline("  IsSealed");
            if (t.IsValueType)
                Console.Writeline("  IsValueType");
            if (t.IsArray)
                Console.Writeline("  IsArray");
            if (t.IsPrimitive)
                Console.Writeline("  IsPrimitive");
            i = i + 1;
        int[] myint = new int[5];
        int[,] myint2 = new int[3, 4];
        int[,,] myint3 = new int[2, 3, 4];
        bool[] mybool = new bool[10];
        bool[,] mybool2 = new bool[5, 5];
        Console.Writeline("myint: {0}", myint.GetType());
        Console.Writeline("myint2: {0}", myint2.GetType());
        Console.Writeline("myint3: {0}", myint3.GetType());
        Console.Writeline("mybool: {0}", mybool.GetType());
        Console.Writeline("mybool2: {0}", mybool2.GetType());
        int[] myint = GetInts();
        int[,] myint2 = GetInts();
        int[,,] myint3 = GetInts();
        bool[] mybool = GetBools();
        bool[,] mybool2 = GetBools();
        Console.Writeline("myint: {0}", myint.GetType());
        Console.Writeline("myint2: {0}", myint2.GetType());
        Console.Writeline("myint3: {0}", myint3.GetType());
        Console.Writeline("mybool: {0}", mybool.GetType());
        Console.Writeline("mybool2: {0}", mybool2.GetType());
    }
    int[] GetInts();
    int[,] GetInts();
    int[,,] GetInts();
    bool[] GetBools();
    bool[,] GetBools();
}
```

Данный пример программирует каждый тип, определяемый в библиотеке классов, и выводит для каждого из них на экран его индекс и полное имя. Также выводится информация о том, является ли каждый из типов классом, перечислением, простым, закрытым, общим, значимым или примитивным. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

```
class core.utils.dataobjects
    namespace Core.Lang.Objects
        namespace Core.Lang.Objects
            namespace Core.Lang.Objects
```

Данный пример программирует следующие выходные потоки. Для примера:

```
myint: System.Int32[]
myint2: System.Int32[,]
myint3: System.Int32[,,]
mybool: System.Boolean[]
mybool2: System.Boolean[,]
```

Данное в пример программирует следующие выходные потоки. Для примера:

Класс Type определяет ряд свойств, которые позволяют определять, что представляет собой каждый из типов, определяемых в библиотеке классов. Следующие свойства используются чаще всего:

Следующие свойства используются для определения, что каждый из типов, определяемых в библиотеке классов, является простым, закрытым, общим, значимым или примитивным. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

Следующие свойства используются чаще всего:

Следующие свойства используются для определения, что каждый из типов, определяемых в библиотеке классов, является перечислением, является ли каждый из типов классом, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

11.3. Класс Boolean

Класс Boolean создает для каждого типа данных, определяемого в библиотеке классов, свой подкласс производный от объекта Object. Этот подкласс позволяет определять, что представляет собой каждый из типов, определяемых в библиотеке классов. Класс Boolean определяется следующим образом:

Класс Boolean позволяет перебирать по каждому типу в библиотеке. Также можно формировать фабрику для каждого типа, определяемого в библиотеке классов. Класс Boolean также определяет ряд свойств, которые можно использовать для определения, что представляет собой каждый из типов, определяемых в библиотеке классов.

Следующие свойства используются чаще всего:

Следующие свойства используются для перечисления по каждому типу. Следующий метод класс выводит такую информацию для каждого типа:

```
public class TypeDemo
    public static void Main(string[] argv)
        Type[] types = Type.GetTypes();
        int i = 0;
        foreach (Type t in types)
            Console.Writeline("{0}: {1}", i, t);
            if (t.IsClass)
                Console.Writeline("  IsClass");
            if (t.IsEnum)
                Console.Writeline("  IsEnum");
            if (t.IsPublic)
                Console.Writeline("  IsPublic");
            if (t.IsSealed)
                Console.Writeline("  IsSealed");
            if (t.IsValueType)
                Console.Writeline("  IsValueType");
            if (t.IsArray)
                Console.Writeline("  IsArray");
            if (t.IsPrimitive)
                Console.Writeline("  IsPrimitive");
            i = i + 1;
        int[] myint = new int[5];
        int[,] myint2 = new int[3, 4];
        int[,,] myint3 = new int[2, 3, 4];
        bool[] mybool = new bool[10];
        bool[,] mybool2 = new bool[5, 5];
        Console.Writeline("myint: {0}", myint.GetType());
        Console.Writeline("myint2: {0}", myint2.GetType());
        Console.Writeline("myint3: {0}", myint3.GetType());
        Console.Writeline("mybool: {0}", mybool.GetType());
        Console.Writeline("mybool2: {0}", mybool2.GetType());
        int[] myint = GetInts();
        int[,] myint2 = GetInts();
        int[,,] myint3 = GetInts();
        bool[] mybool = GetBools();
        bool[,] mybool2 = GetBools();
        Console.Writeline("myint: {0}", myint.GetType());
        Console.Writeline("myint2: {0}", myint2.GetType());
        Console.Writeline("myint3: {0}", myint3.GetType());
        Console.Writeline("mybool: {0}", mybool.GetType());
        Console.Writeline("mybool2: {0}", mybool2.GetType());
    }
    int[] GetInts();
    int[,] GetInts();
    int[,,] GetInts();
    bool[] GetBools();
    bool[,] GetBools();
}
```

Данный пример программирует каждый тип, определяемый в библиотеке классов, и выводит для каждого из них на экран его индекс и полное имя. Также выводится информация о том, является ли каждый из типов классом, перечислением, простым, закрытым, общим, значимым или примитивным. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

```
class core.utils.dataobjects
    namespace Core.Lang.Objects
        namespace Core.Lang.Objects
            namespace Core.Lang.Objects
```

Данный пример программирует следующие выходные потоки. Для примера:

```
myint: System.Int32[]
myint2: System.Int32[,]
myint3: System.Int32[,,]
mybool: System.Boolean[]
mybool2: System.Boolean[,]
```

Данное в пример программирует следующие выходные потоки. Для примера:

Класс Type определяет ряд свойств, которые позволяют определять, что представляет собой каждый из типов, определяемых в библиотеке классов. Следующие свойства используются чаще всего:

Следующие свойства используются для определения, что каждый из типов, определяемых в библиотеке классов, является простым, закрытым, общим, значимым или примитивным. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

Следующие свойства используются чаще всего:

Следующие свойства используются для определения, что каждый из типов, определяемых в библиотеке классов, является перечислением, является ли каждый из типов классом, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

11.4. Класс Number

Для каждого класса в библиотеке классов имеется подкласс производный от объекта Object. Этот объект может использоваться для получения информации о классе, определяемом в библиотеке классов. Класс Number определяется следующим образом:

Класс Number позволяет перебирать по каждому типу в библиотеке. Также можно формировать фабрику для каждого типа, определяемого в библиотеке классов. Класс Number также определяет ряд свойств, которые можно использовать для определения, что представляет собой каждый из типов, определяемых в библиотеке классов.

Следующие свойства используются чаще всего:

Следующие свойства используются для перечисления по каждому типу. Следующий метод класс выводит такую информацию для каждого типа:

```
public class TypeDemo
    public static void Main(string[] argv)
        Type[] types = Type.GetTypes();
        int i = 0;
        foreach (Type t in types)
            Console.Writeline("{0}: {1}", i, t);
            if (t.IsClass)
                Console.Writeline("  IsClass");
            if (t.IsEnum)
                Console.Writeline("  IsEnum");
            if (t.IsPublic)
                Console.Writeline("  IsPublic");
            if (t.IsSealed)
                Console.Writeline("  IsSealed");
            if (t.IsValueType)
                Console.Writeline("  IsValueType");
            if (t.IsArray)
                Console.Writeline("  IsArray");
            if (t.IsPrimitive)
                Console.Writeline("  IsPrimitive");
            i = i + 1;
        int[] myint = new int[5];
        int[,] myint2 = new int[3, 4];
        int[,,] myint3 = new int[2, 3, 4];
        bool[] mybool = new bool[10];
        bool[,] mybool2 = new bool[5, 5];
        Console.Writeline("myint: {0}", myint.GetType());
        Console.Writeline("myint2: {0}", myint2.GetType());
        Console.Writeline("myint3: {0}", myint3.GetType());
        Console.Writeline("mybool: {0}", mybool.GetType());
        Console.Writeline("mybool2: {0}", mybool2.GetType());
        int[] myint = GetInts();
        int[,] myint2 = GetInts();
        int[,,] myint3 = GetInts();
        bool[] mybool = GetBools();
        bool[,] mybool2 = GetBools();
        Console.Writeline("myint: {0}", myint.GetType());
        Console.Writeline("myint2: {0}", myint2.GetType());
        Console.Writeline("myint3: {0}", myint3.GetType());
        Console.Writeline("mybool: {0}", mybool.GetType());
        Console.Writeline("mybool2: {0}", mybool2.GetType());
    }
    int[] GetInts();
    int[,] GetInts();
    int[,,] GetInts();
    bool[] GetBools();
    bool[,] GetBools();
}
```

Данный пример программирует каждый тип, определяемый в библиотеке классов, и выводит для каждого из них на экран его индекс и полное имя. Также выводится информация о том, является ли каждый из типов классом, перечислением, простым, закрытым, общим, значимым или примитивным. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

```
class core.utils.dataobjects
    namespace Core.Lang.Objects
        namespace Core.Lang.Objects
            namespace Core.Lang.Objects
```

Данный пример программирует следующие выходные потоки. Для примера:

```
myint: System.Int32[]
myint2: System.Int32[,]
myint3: System.Int32[,,]
mybool: System.Boolean[]
mybool2: System.Boolean[,]
```

Данное в пример программирует следующие выходные потоки. Для примера:

Класс Type определяет ряд свойств, которые позволяют определять, что представляет собой каждый из типов, определяемых в библиотеке классов. Следующие свойства используются чаще всего:

Следующие свойства используются для определения, что каждый из типов, определяемых в библиотеке классов, является простым, закрытым, общим, значимым или примитивным. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

Следующие свойства используются чаще всего:

Следующие свойства используются для определения, что каждый из типов, определяемых в библиотеке классов, является перечислением, является ли каждый из типов классом, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

11.5. Класс Integer

Для каждого класса в библиотеке классов имеется подкласс производный от объекта Object. Этот объект может использоваться для получения информации о классе, определяемом в библиотеке классов. Класс Integer определяется следующим образом:

Класс Integer позволяет перебирать по каждому типу в библиотеке. Также можно формировать фабрику для каждого типа, определяемого в библиотеке классов. Класс Integer также определяет ряд свойств, которые можно использовать для определения, что представляет собой каждый из типов, определяемых в библиотеке классов.

Следующие свойства используются чаще всего:

Следующие свойства используются для перечисления по каждому типу. Следующий метод класс выводит такую информацию для каждого типа:

```
public class TypeDemo
    public static void Main(string[] argv)
        Type[] types = Type.GetTypes();
        int i = 0;
        foreach (Type t in types)
            Console.Writeline("{0}: {1}", i, t);
            if (t.IsClass)
                Console.Writeline("  IsClass");
            if (t.IsEnum)
                Console.Writeline("  IsEnum");
            if (t.IsPublic)
                Console.Writeline("  IsPublic");
            if (t.IsSealed)
                Console.Writeline("  IsSealed");
            if (t.IsValueType)
                Console.Writeline("  IsValueType");
            if (t.IsArray)
                Console.Writeline("  IsArray");
            if (t.IsPrimitive)
                Console.Writeline("  IsPrimitive");
            i = i + 1;
        int[] myint = new int[5];
        int[,] myint2 = new int[3, 4];
        int[,,] myint3 = new int[2, 3, 4];
        bool[] mybool = new bool[10];
        bool[,] mybool2 = new bool[5, 5];
        Console.Writeline("myint: {0}", myint.GetType());
        Console.Writeline("myint2: {0}", myint2.GetType());
        Console.Writeline("myint3: {0}", myint3.GetType());
        Console.Writeline("mybool: {0}", mybool.GetType());
        Console.Writeline("mybool2: {0}", mybool2.GetType());
        int[] myint = GetInts();
        int[,] myint2 = GetInts();
        int[,,] myint3 = GetInts();
        bool[] mybool = GetBools();
        bool[,] mybool2 = GetBools();
        Console.Writeline("myint: {0}", myint.GetType());
        Console.Writeline("myint2: {0}", myint2.GetType());
        Console.Writeline("myint3: {0}", myint3.GetType());
        Console.Writeline("mybool: {0}", mybool.GetType());
        Console.Writeline("mybool2: {0}", mybool2.GetType());
    }
    int[] GetInts();
    int[,] GetInts();
    int[,,] GetInts();
    bool[] GetBools();
    bool[,] GetBools();
}
```

Данный пример программирует каждый тип, определяемый в библиотеке классов, и выводит для каждого из них на экран его индекс и полное имя. Также выводится информация о том, является ли каждый из типов классом, перечислением, простым, закрытым, общим, значимым или примитивным. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

```
class core.utils.dataobjects
    namespace Core.Lang.Objects
        namespace Core.Lang.Objects
            namespace Core.Lang.Objects
```

Данный пример программирует следующие выходные потоки. Для примера:

```
myint: System.Int32[]
myint2: System.Int32[,]
myint3: System.Int32[,,]
mybool: System.Boolean[]
mybool2: System.Boolean[,]
```

Данное в пример программирует следующие выходные потоки. Для примера:

Класс Type определяет ряд свойств, которые позволяют определять, что представляет собой каждый из типов, определяемых в библиотеке классов. Следующие свойства используются чаще всего:

Следующие свойства используются для определения, что каждый из типов, определяемых в библиотеке классов, является простым, закрытым, общим, значимым или примитивным. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

Следующие свойства используются чаще всего:

Следующие свойства используются для определения, что каждый из типов, определяемых в библиотеке классов, является перечислением, является ли каждый из типов классом, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

11.6. Класс Number

Для каждого класса в библиотеке классов имеется подкласс производный от объекта Object. Этот объект может использоваться для получения информации о классе, определяемом в библиотеке классов. Класс Number определяется следующим образом:

Класс Number позволяет перебирать по каждому типу в библиотеке. Также можно формировать фабрику для каждого типа, определяемого в библиотеке классов. Класс Number также определяет ряд свойств, которые можно использовать для определения, что представляет собой каждый из типов, определяемых в библиотеке классов.

Следующие свойства используются чаще всего:

Следующие свойства используются для перечисления по каждому типу. Следующий метод класс выводит такую информацию для каждого типа:

```
public class TypeDemo
    public static void Main(string[] argv)
        Type[] types = Type.GetTypes();
        int i = 0;
        foreach (Type t in types)
            Console.Writeline("{0}: {1}", i, t);
            if (t.IsClass)
                Console.Writeline("  IsClass");
            if (t.IsEnum)
                Console.Writeline("  IsEnum");
            if (t.IsPublic)
                Console.Writeline("  IsPublic");
            if (t.IsSealed)
                Console.Writeline("  IsSealed");
            if (t.IsValueType)
                Console.Writeline("  IsValueType");
            if (t.IsArray)
                Console.Writeline("  IsArray");
            if (t.IsPrimitive)
                Console.Writeline("  IsPrimitive");
            i = i + 1;
        int[] myint = new int[5];
        int[,] myint2 = new int[3, 4];
        int[,,] myint3 = new int[2, 3, 4];
        bool[] mybool = new bool[10];
        bool[,] mybool2 = new bool[5, 5];
        Console.Writeline("myint: {0}", myint.GetType());
        Console.Writeline("myint2: {0}", myint2.GetType());
        Console.Writeline("myint3: {0}", myint3.GetType());
        Console.Writeline("mybool: {0}", mybool.GetType());
        Console.Writeline("mybool2: {0}", mybool2.GetType());
        int[] myint = GetInts();
        int[,] myint2 = GetInts();
        int[,,] myint3 = GetInts();
        bool[] mybool = GetBools();
        bool[,] mybool2 = GetBools();
        Console.Writeline("myint: {0}", myint.GetType());
        Console.Writeline("myint2: {0}", myint2.GetType());
        Console.Writeline("myint3: {0}", myint3.GetType());
        Console.Writeline("mybool: {0}", mybool.GetType());
        Console.Writeline("mybool2: {0}", mybool2.GetType());
    }
    int[] GetInts();
    int[,] GetInts();
    int[,,] GetInts();
    bool[] GetBools();
    bool[,] GetBools();
}
```

Данный пример программирует каждый тип, определяемый в библиотеке классов, и выводит для каждого из них на экран его индекс и полное имя. Также выводится информация о том, является ли каждый из типов классом, перечислением, простым, закрытым, общим, значимым или примитивным. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

```
class core.utils.dataobjects
    namespace Core.Lang.Objects
        namespace Core.Lang.Objects
            namespace Core.Lang.Objects
```

Данный пример программирует следующие выходные потоки. Для примера:

```
myint: System.Int32[]
myint2: System.Int32[,]
myint3: System.Int32[,,]
mybool: System.Boolean[]
mybool2: System.Boolean[,]
```

Данное в пример программирует следующие выходные потоки. Для примера:

Класс Type определяет ряд свойств, которые позволяют определять, что представляет собой каждый из типов, определяемых в библиотеке классов. Следующие свойства используются чаще всего:

Следующие свойства используются для определения, что каждый из типов, определяемых в библиотеке классов, является простым, закрытым, общим, значимым или примитивным. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

Следующие свойства используются чаще всего:

Следующие свойства используются для определения, что каждый из типов, определяемых в библиотеке классов, является перечислением, является ли каждый из типов классом, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

11.7. Класс Long

Для каждого класса в библиотеке классов имеется подкласс производный от объекта Object. Этот объект может использоваться для получения информации о классе, определяемом в библиотеке классов. Класс Long определяется следующим образом:

Класс Long позволяет перебирать по каждому типу в библиотеке. Также можно формировать фабрику для каждого типа, определяемого в библиотеке классов. Класс Long также определяет ряд свойств, которые можно использовать для определения, что представляет собой каждый из типов, определяемых в библиотеке классов.

Следующие свойства используются чаще всего:

Следующие свойства используются для перечисления по каждому типу. Следующий метод класс выводит такую информацию для каждого типа:

```
public class TypeDemo
    public static void Main(string[] argv)
        Type[] types = Type.GetTypes();
        int i = 0;
        foreach (Type t in types)
            Console.Writeline("{0}: {1}", i, t);
            if (t.IsClass)
                Console.Writeline("  IsClass");
            if (t.IsEnum)
                Console.Writeline("  IsEnum");
            if (t.IsPublic)
                Console.Writeline("  IsPublic");
            if (t.IsSealed)
                Console.Writeline("  IsSealed");
            if (t.IsValueType)
                Console.Writeline("  IsValueType");
            if (t.IsArray)
                Console.Writeline("  IsArray");
            if (t.IsPrimitive)
                Console.Writeline("  IsPrimitive");
            i = i + 1;
        int[] myint = new int[5];
        int[,] myint2 = new int[3, 4];
        int[,,] myint3 = new int[2, 3, 4];
        bool[] mybool = new bool[10];
        bool[,] mybool2 = new bool[5, 5];
        Console.Writeline("myint: {0}", myint.GetType());
        Console.Writeline("myint2: {0}", myint2.GetType());
        Console.Writeline("myint3: {0}", myint3.GetType());
        Console.Writeline("mybool: {0}", mybool.GetType());
        Console.Writeline("mybool2: {0}", mybool2.GetType());
        int[] myint = GetInts();
        int[,] myint2 = GetInts();
        int[,,] myint3 = GetInts();
        bool[] mybool = GetBools();
        bool[,] mybool2 = GetBools();
        Console.Writeline("myint: {0}", myint.GetType());
        Console.Writeline("myint2: {0}", myint2.GetType());
        Console.Writeline("myint3: {0}", myint3.GetType());
        Console.Writeline("mybool: {0}", mybool.GetType());
        Console.Writeline("mybool2: {0}", mybool2.GetType());
    }
    int[] GetInts();
    int[,] GetInts();
    int[,,] GetInts();
    bool[] GetBools();
    bool[,] GetBools();
}
```

Данный пример программирует каждый тип, определяемый в библиотеке классов, и выводит для каждого из них на экран его индекс и полное имя. Также выводится информация о том, является ли каждый из типов классом, перечислением, простым, закрытым, общим, значимым или примитивным. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

```
class core.utils.dataobjects
    namespace Core.Lang.Objects
        namespace Core.Lang.Objects
            namespace Core.Lang.Objects
```

Данный пример программирует следующие выходные потоки. Для примера:

```
myint: System.Int32[]
myint2: System.Int32[,]
myint3: System.Int32[,,]
mybool: System.Boolean[]
mybool2: System.Boolean[,]
```

Данное в пример программирует следующие выходные потоки. Для примера:

Класс Type определяет ряд свойств, которые позволяют определять, что представляет собой каждый из типов, определяемых в библиотеке классов. Следующие свойства используются чаще всего:

Следующие свойства используются для определения, что каждый из типов, определяемых в библиотеке классов, является простым, закрытым, общим, значимым или примитивным. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

Следующие свойства используются чаще всего:

Следующие свойства используются для определения, что каждый из типов, определяемых в библиотеке классов, является перечислением, является ли каждый из типов классом, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

11.8. Класс Float

Для каждого класса в библиотеке классов имеется подкласс производный от объекта Object. Этот объект может использоваться для получения информации о классе, определяемом в библиотеке классов. Класс Float определяется следующим образом:

Класс Float позволяет перебирать по каждому типу в библиотеке. Также можно формировать фабрику для каждого типа, определяемого в библиотеке классов. Класс Float также определяет ряд свойств, которые можно использовать для определения, что представляет собой каждый из типов, определяемых в библиотеке классов.

Следующие свойства используются чаще всего:

Следующие свойства используются для перечисления по каждому типу. Следующий метод класс выводит такую информацию для каждого типа:

```
public class TypeDemo
    public static void Main(string[] argv)
        Type[] types = Type.GetTypes();
        int i = 0;
        foreach (Type t in types)
            Console.Writeline("{0}: {1}", i, t);
            if (t.IsClass)
                Console.Writeline("  IsClass");
            if (t.IsEnum)
                Console.Writeline("  IsEnum");
            if (t.IsPublic)
                Console.Writeline("  IsPublic");
            if (t.IsSealed)
                Console.Writeline("  IsSealed");
            if (t.IsValueType)
                Console.Writeline("  IsValueType");
            if (t.IsArray)
                Console.Writeline("  IsArray");
            if (t.IsPrimitive)
                Console.Writeline("  IsPrimitive");
            i = i + 1;
        int[] myint = new int[5];
        int[,] myint2 = new int[3, 4];
        int[,,] myint3 = new int[2, 3, 4];
        bool[] mybool = new bool[10];
        bool[,] mybool2 = new bool[5, 5];
        Console.Writeline("myint: {0}", myint.GetType());
        Console.Writeline("myint2: {0}", myint2.GetType());
        Console.Writeline("myint3: {0}", myint3.GetType());
        Console.Writeline("mybool: {0}", mybool.GetType());
        Console.Writeline("mybool2: {0}", mybool2.GetType());
        int[] myint = GetInts();
        int[,] myint2 = GetInts();
        int[,,] myint3 = GetInts();
        bool[] mybool = GetBools();
        bool[,] mybool2 = GetBools();
        Console.Writeline("myint: {0}", myint.GetType());
        Console.Writeline("myint2: {0}", myint2.GetType());
        Console.Writeline("myint3: {0}", myint3.GetType());
        Console.Writeline("mybool: {0}", mybool.GetType());
        Console.Writeline("mybool2: {0}", mybool2.GetType());
    }
    int[] GetInts();
    int[,] GetInts();
    int[,,] GetInts();
    bool[] GetBools();
    bool[,] GetBools();
}
```

Данный пример программирует каждый тип, определяемый в библиотеке классов, и выводит для каждого из них на экран его индекс и полное имя. Также выводится информация о том, является ли каждый из типов классом, перечислением, простым, закрытым, общим, значимым или примитивным. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

```
class core.utils.dataobjects
    namespace Core.Lang.Objects
        namespace Core.Lang.Objects
            namespace Core.Lang.Objects
```

Данный пример программирует следующие выходные потоки. Для примера:

```
myint: System.Int32[]
myint2: System.Int32[,]
myint3: System.Int32[,,]
mybool: System.Boolean[]
mybool2: System.Boolean[,]
```

Данное в пример программирует следующие выходные потоки. Для примера:

Класс Type определяет ряд свойств, которые позволяют определять, что представляет собой каждый из типов, определяемых в библиотеке классов. Следующие свойства используются чаще всего:

Следующие свойства используются для определения, что каждый из типов, определяемых в библиотеке классов, является простым, закрытым, общим, значимым или примитивным. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

Следующие свойства используются чаще всего:

Следующие свойства используются для определения, что каждый из типов, определяемых в библиотеке классов, является перечислением, является ли каждый из типов классом, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом. Также выводится информация о том, является ли каждый из типов массивом.

11.9. Класс Double

Для каждого класса в библиотеке классов имеется подкласс производный от объекта Object. Этот объект может использоваться для получения информации о классе, определяемом в библиотеке классов. Класс Double определяется следующим образом:

Класс Double позволяет перебирать по каждому типу в библиотеке. Также можно формировать фабрику для каждого типа, определяемого в библиотеке классов. Класс Double также определяет ряд свойств, которые можно использовать для определения, что представляет собой каждый из типов, определяемых в библиотеке классов.

Следующие свойства используются чаще всего:

Следующие свойства используются для перечисления по каждому типу. Следующий метод класс выводит такую информацию для каждого типа:

```
public class TypeDemo
    public static void Main(string[] argv)
        Type[] types = Type.GetTypes();
        int i = 0;
        foreach (Type t in types)
            Console.Writeline("{0}: {1}", i, t);
            if (
```

Глава 1
ЛЕКСЕМЫ, ОПЕРАТОРЫ И ВЫРАЖЕНИЯ

Перевод с английского
Версия 1.0 от 15 ноября 2000 г.
Последняя редакция 15 ноября 2000 г.
Перевод: Дмитрий Кирсанов
Редактирование и дополнительный материал: Алексей Кирсанов

Данный перевод выполнен в некоммерческих образовательных целях. Запрещено использование без согласия авторов-переводчиков.

1.1. Набор символов

Исходный текст Perl'а может быть представлен в виде последовательности строк, разделенных любым из следующих разделителей: перевод строки, возврат каретки, возврат каретки с последующим переводом строки или конец файла. Внутри строки разрешены любые символы кроме разделителей, за исключением того, что конец файла может быть использован только в качестве последнего символа.

Если в первой строке файла сценария Perl'а стоит пара символов "#!" и интерпретатор запускается с ключом, то Perl попытается выполнить все последующие строки, пока не будет достигнут конец файла. Если в последующем тексте встретится слово "__END__", то Perl игнорирует все последующие строки. Если же после "#!" не стоит ключ, то Perl будет выполнять только те строки, которые расположены между первой строкой и "__END__".

Если после строки "__END__" встретится слово "DATA", то Perl создаст для этого файлового дескриптора возможность чтения из файла, начиная с следующей за "__END__" строки. Подробнее см. perlfunc/__END__.

```
# Пример использования "__END__"
while (<DATA>) {
    print "Line from DATA: $_";
}
__END__
Line 1
Line 2
Line 3
```

Перед выполнением каждого оператора Perl производит его синтаксический анализ. Если обнаружена ошибка, то выполнение прекращается после выдачи сообщения о ошибке. Исключение составляют операторы, которые не выполняются из-за условных операторов типа if и while. Если же ошибка обнаружена в операторе, который выполняется, то результат может быть неопределенным.

Perl воспринимает пробелы, табуляции и символы перевода строки как пустые, за исключением того случая, когда они стоят внутри строки. Например, выражение "foo" + "bar" будет воспринято как "foo" + "bar".

Perl воспринимает символ "#" как начало комментария. Все последующие за ним символы игнорируются.

Perl воспринимает символ "^D" (control-D) как конец файла, если в последнем стоит перевод строки.

```
print "Hello, world\n";
__END__
^D
```

1.2. Лексемы

Лексемы (tokens) похожи на слова в английском языке. Perl воспринимает следующие лексемы: числовые и строковые константы, имена переменных и идентификаторов, знаки операций и специальные символы, которые используются для обозначения различных операций.

Например, в строке:

```
print "Hello, world\n";
```

Perl воспринимает лексемы "print", "Hello, world\n" и ";".

Иногда бывает трудно между собой отличить лексемы и операнды. Например, в строке:

```
print foo + bar;
```

Perl воспринимает лексемы "print", "foo", "+" и "bar". Иногда же бывает трудно определить какие лексемы являются операндами. Например, в строке:

```
print foo bar baz;
```

Perl воспринимает лексемы "print", "foo", "bar" и "baz", поскольку по умолчанию все последующие лексемы воспринимаются как операнды.

Иногда бывает полезно использовать скобки, чтобы Perl воспринимал лексемы как надо. Например, в строке:

```
print foo(bar) + baz;
```

Perl воспринимает лексемы "print", "foo", "bar", "+" и "baz".

Perl воспринимает пробелы и символы перевода строки как пустые, за исключением того случая, когда они стоят внутри строки. Например, выражение "foo" + "bar" будет воспринято как "foo" + "bar".

Иногда бывает необходимо использовать пустые лексемы. Например, в выражении:

```
print $a, $b, $c;
```

Perl воспринимает лексемы "print", "$a", "," и "$b". Если же нужно, чтобы Perl воспринимал лексемы "print", "$a", "," и "$c", то нужно использовать пустую лексему, которая в данном случае будет воспринята как "print", "$a", "," и "$c".

Пустые лексемы также бывают полезны, если нужно, чтобы Perl воспринимал что-то как оператор, а не как функцию. Подробнее см. perlfunc/Functions.

1.3. Литералы

Литералы (literals) - это числовые и строковые константы. Perl воспринимает следующие числовые константы:

```
123
-123
123.45
-123.45
0123
0x123
0b123
123e45
123.45e67
123.45e-67
```

Perl воспринимает следующие строковые константы:

```
'foo'
'bar'
'baz'
'Hello, world\n'
'Hello, world\t\t\n'
'Hello, world\'s\n'
'Hello, world\\s\n'
"foo"
"bar"
"baz"
"Hello, world\n"
"Hello, world\t\t\n"
"Hello, world\"s\n"
"Hello, world\$s\n"
"Hello, world\@s\n"
"Hello, world\\s\n"
```

1.4. Переменные типа

Perl воспринимает следующие переменные типа (variable type):

```
$foo
$bar
$baz
$Foo::bar
$^W
$"
@foo
@bar
@baz
@Foo::bar
@+
%foo
%bar
%baz
%Foo::bar
%!
&foo
&bar
&baz
&Foo::bar
&&
*foo
*bar
*baz
*Foo::bar
*[
<foo>
<bar>
<baz>
<Foo::bar>
<>
<*.c>
foo:
bar:
baz:
Foo::bar:
```

1.5. Операторы

Операторы (operators) - это знаки операций, которые используются для обозначения различных операций. Например, знак "+" обозначает операцию сложения, знак "-" обозначает операцию вычитания, а знак "*" обозначает операцию умножения. Подробнее см. perlfunc/Operators.

1.6. Выражения

Выражения (expressions) - это комбинации лексем, которые Perl воспринимает как осмысленные. Например, в строке:

```
print "Hello, world\n";
```

Perl воспринимает лексемы "print", "Hello, world\n" и ";". Perl воспринимает лексемы "print" и "Hello, world\n" как выражения. Подробнее см. perlfunc/Functions.

1.7. Переменные типа

Perl воспринимает следующие переменные типа (variable type):

```
$foo
$bar
$baz
$Foo::bar
$^W
$"
@foo
@bar
@baz
@Foo::bar
@+
%foo
%bar
%baz
%Foo::bar
%!
&foo
&bar
&baz
&Foo::bar
&&
*foo
*bar
*baz
*Foo::bar
*[
<foo>
<bar>
<baz>
<Foo::bar>
<>
<*.c>
foo:
bar:
baz:
Foo::bar:
```

1.8. Литералы

Литералы (literals) - это числовые и строковые константы. Perl воспринимает следующие числовые константы:

```
123
-123
123.45
-123.45
0123
0x123
0b123
123e45
123.45e67
123.45e-67
```

Perl воспринимает следующие строковые константы:

```
'foo'
'bar'
'baz'
'Hello, world\n'
'Hello, world\t\t\n'
'Hello, world\'s\n'
'Hello, world\\s\n'
"foo"
"bar"
"baz"
"Hello, world\n"
"Hello, world\t\t\n"
"Hello, world\"s\n"
"Hello, world\$s\n"
"Hello, world\@s\n"
"Hello, world\\s\n"
```

1.9. Переменные типа

Perl воспринимает следующие переменные типа (variable type):

```
$foo
$bar
$baz
$Foo::bar
$^W
$"
@foo
@bar
@baz
@Foo::bar
@+
%foo
%bar
%baz
%Foo::bar
%!
&foo
&bar
&baz
&Foo::bar
&&
*foo
*bar
*baz
*Foo::bar
*[
<foo>
<bar>
<baz>
<Foo::bar>
<>
<*.c>
foo:
bar:
baz:
Foo::bar:
```

1.10. Выражения

Выражения (expressions) - это комбинации лексем, которые Perl воспринимает как осмысленные. Например, в строке:

```
print "Hello, world\n";
```

Perl воспринимает лексемы "print", "Hello, world\n" и ";". Perl воспринимает лексемы "print" и "Hello, world\n" как выражения. Подробнее см. perlfunc/Functions.

1.11. Операторы

Операторы (operators) - это знаки операций, которые используются для обозначения различных операций. Например, знак "+" обозначает операцию сложения, знак "-" обозначает операцию вычитания, а знак "*" обозначает операцию умножения. Подробнее см. perlfunc/Operators.

1.12. Литералы

Литералы (literals) - это числовые и строковые константы. Perl воспринимает следующие числовые константы:

```
123
-123
123.45
-123.45
0123
0x123
0b123
123e45
123.45e67
123.45e-67
```

Perl воспринимает следующие строковые константы:

```
'foo'
'bar'
'baz'
'Hello, world\n'
'Hello, world\t\t\n'
'Hello, world\'s\n'
'Hello, world\\s\n'
"foo"
"bar"
"baz"
"Hello, world\n"
"Hello, world\t\t\n"
"Hello, world\"s\n"
"Hello, world\$s\n"
"Hello, world\@s\n"
"Hello, world\\s\n"
```

1.13. Переменные типа

Perl воспринимает следующие переменные типа (variable type):

```
$foo
$bar
$baz
$Foo::bar
$^W
$"
@foo
@bar
@baz
@Foo::bar
@+
%foo
%bar
%baz
%Foo::bar
%!
&foo
&bar
&baz
&Foo::bar
&&
*foo
*bar
*baz
*Foo::bar
*[
<foo>
<bar>
<baz>
<Foo::bar>
<>
<*.c>
foo:
bar:
baz:
Foo::bar:
```

1.14. Выражения

Выражения (expressions) - это комбинации лексем, которые Perl воспринимает как осмысленные. Например, в строке:

```
print "Hello, world\n";
```

Perl воспринимает лексемы "print", "Hello, world\n" и ";". Perl воспринимает лексемы "print" и "Hello, world\n" как выражения. Подробнее см. perlfunc/Functions.

1.15. Операторы

Операторы (operators) - это знаки операций, которые используются для обозначения различных операций. Например, знак "+" обозначает операцию сложения, знак "-" обозначает операцию вычитания, а знак "*" обозначает операцию умножения. Подробнее см. perlfunc/Operators.

1.16. Литералы

Литералы (literals) - это числовые и строковые константы. Perl воспринимает следующие числовые константы:

```
123
-123
123.45
-123.45
0123
0x123
0b123
123e45
123.45e67
123.45e-67
```

Perl воспринимает следующие строковые константы:

```
'foo'
'bar'
'baz'
'Hello, world\n'
'Hello, world\t\t\n'
'Hello, world\'s\n'
'Hello, world\\s\n'
"foo"
"bar"
"baz"
"Hello, world\n"
"Hello, world\t\t\n"
"Hello, world\"s\n"
"Hello, world\$s\n"
"Hello, world\@s\n"
"Hello, world\\s\n"
```

1.17. Переменные типа

Perl воспринимает следующие переменные типа (variable type):

```
$foo
$bar
$baz
$Foo::bar
$^W
$"
@foo
@bar
@baz
@Foo::bar
@+
%foo
%bar
%baz
%Foo::bar
%!
&foo
&bar
&baz
&Foo::bar
&&
*foo
*bar
*baz
*Foo::bar
*[
<foo>
<bar>
<baz>
<Foo::bar>
<>
<*.c>
foo:
bar:
baz:
Foo::bar:
```

1.18. Выражения

Выражения (expressions) - это комбинации лексем, которые Perl воспринимает как осмысленные. Например, в строке:

```
print "Hello, world\n";
```

Perl воспринимает лексемы "print", "Hello, world\n" и ";". Perl воспринимает лексемы "print" и "Hello, world\n" как выражения. Подробнее см. perlfunc/Functions.

1.19. Операторы

Операторы (operators) - это знаки операций, которые используются для обозначения различных операций. Например, знак "+" обозначает операцию сложения, знак "-" обозначает операцию вычитания, а знак "*" обозначает операцию умножения. Подробнее см. perlfunc/Operators.

1.20. Литералы

Литералы (literals) - это числовые и строковые константы. Perl воспринимает следующие числовые константы:

```
123
-123
123.45
-123.45
0123
0x123
0b123
123e45
123.45e67
123.45e-67
```

Perl воспринимает следующие строковые константы:

```
'foo'
'bar'
'baz'
'Hello, world\n'
'Hello, world\t\t\n'
'Hello, world\'s\n'
'Hello, world\\s\n'
"foo"
"bar"
"baz"
"Hello, world\n"
"Hello, world\t\t\n"
"Hello, world\"s\n"
"Hello, world\$s\n"
"Hello, world\@s\n"
"Hello, world\\s\n"
```

1.21. Переменные типа

Perl воспринимает следующие переменные типа (variable type):

```
$foo
$bar
$baz
$Foo::bar
$^W
$"
@foo
@bar
@baz
@Foo::bar
@+
%foo
%bar
%baz
%Foo::bar
%!
&foo
&bar
&baz
&Foo::bar
&&
*foo
*bar
*baz
*Foo::bar
*[
<foo>
<bar>
<baz>
<Foo::bar>
<>
<*.c>
foo:
bar:
baz:
Foo::bar:
```

1.22. Выражения

Выражения (expressions) - это комбинации лексем, которые Perl воспринимает как осмысленные. Например, в строке:

```
print "Hello, world\n";
```

Perl воспринимает лексемы "print", "Hello, world\n" и ";". Perl воспринимает лексемы "print" и "Hello, world\n" как выражения. Подробнее см. perlfunc/Functions.

1.23. Операторы

Операторы (operators) - это знаки операций, которые используются для обозначения различных операций. Например, знак "+" обозначает операцию сложения, знак "-" обозначает операцию вычитания, а знак "*" обозначает операцию умножения. Подробнее см. perlfunc/Operators.

1.24. Литералы

Литералы (literals) - это числовые и строковые константы. Perl воспринимает следующие числовые константы:

```
123
-123
123.45
-123.45
0123
0x123
0b123
123e45
123.45e67
123.45e-67
```

Perl воспринимает следующие строковые константы:

```
'foo'
'bar'
'baz'
'Hello, world\n'
'Hello, world\t\t\n'
'Hello, world\'s\n'
'Hello, world\\s\n'
"foo"
"bar"
"baz"
"Hello, world\n"
"Hello, world\t\t\n"
"Hello, world\"s\n"
"Hello, world\$s\n"
"Hello, world\@s\n"
"Hello, world\\s\n"
```

1.25. Переменные типа

Perl воспринимает следующие переменные типа (variable type):

```
$foo
$bar
$baz
$Foo::bar
$^W
$"
@foo
@bar
@baz
@Foo::bar
@+
%foo
%bar
%baz
%Foo::bar
%!
&foo
&bar
&baz
&Foo::bar
&&
*foo
*bar
*baz
*Foo::bar
*[
<foo>
<bar>
<baz>
<Foo::bar>
<>
<*.c>
foo:
bar:
baz:
Foo::bar:
```

1.26. Выражения

Выражения (expressions) - это комбинации лексем, которые Perl воспринимает как осмысленные. Например, в строке:

```
print "Hello, world\n";
```

Perl воспринимает лексемы "print", "Hello, world\n" и ";". Perl воспринимает лексемы "print" и "Hello, world\n" как выражения. Подробнее см. perlfunc/Functions.

1.27. Операторы

Операторы (operators) - это знаки операций, которые используются для обозначения различных операций. Например, знак "+" обозначает операцию сложения, знак "-" обозначает операцию вычитания, а знак "*" обозначает операцию умножения. Подробнее см. perlfunc/Operators.

1.28. Литералы

Литералы (literals) - это числовые и строковые константы. Perl воспринимает следующие числовые константы:

```
123
-123
123.45
-123.45
0123
0x123
0b123
123e45
123.45e67
123.45e-67
```

Perl воспринимает следующие строковые константы:

```
'foo'
'bar'
'baz'
'Hello, world\n'
'Hello, world\t\t\n'
'Hello, world\'s\n'
'Hello, world\\s\n'
"foo"
"bar"
"baz"
"Hello, world\n"
"Hello, world\t\t\n"
"Hello, world\"s\n"
"Hello, world\$s\n"
"Hello, world\@s\n"
"Hello, world\\s\n"
```

1.29. Переменные типа

Perl воспринимает следующие переменные типа (variable type):

```
$foo
$bar
$baz
$Foo::bar
$^W
$"
@foo
@bar
@baz
@Foo::bar
@+
%foo
%bar
%baz
%Foo::bar
%!
&foo
&bar
&baz
&Foo::bar
&&
*foo
*bar
*baz
*Foo::bar
*[
<foo>
<bar>
<baz>
<Foo::bar>
<>
<*.c>
foo:
bar:
baz:
Foo::bar:
```

1.30. Выражения

Выражения (expressions) - это комбинации лексем, которые Perl воспринимает как осмысленные. Например, в строке:

```
print "Hello, world\n";
```

Perl воспринимает лексемы "print", "Hello, world\n" и ";". Perl воспринимает лексемы "print" и "Hello, world\n" как выражения. Подробнее см. perlfunc/Functions.

1.31. Операторы

Операторы (operators) - это знаки операций, которые используются для обозначения различных операций. Например, знак "+" обозначает операцию сложения, знак "-" обозначает операцию вычитания, а знак "*" обозначает операцию умножения. Подробнее см. perlfunc/Operators.

1.32. Литералы

Литералы (literals) - это числовые и строковые константы. Perl воспринимает следующие числовые константы:

```
123
-123
123.45
-123.45
0123
0x123
0b123
123e45
123.45e67
123.45e-67
```

Perl воспринимает следующие строковые константы:

```
'foo'
'bar'
'baz'
'Hello, world\n'
'Hello, world\t\t\n'
'Hello, world\'s\n'
'Hello, world\\s\n'
"foo"
"bar"
"baz"
"Hello, world\n"
"Hello, world\t\t\n"
"Hello, world\"s\n"
"Hello, world\$s\n"
"Hello, world\@s\n"
"Hello, world\\s\n"
```

1.33. Переменные типа

Perl воспринимает следующие переменные типа (variable type):

```
$foo
$bar
$baz
$Foo::bar
$^W
$"
@foo
@bar
@baz
@Foo::bar
@+
%foo
%bar
%baz
%Foo::bar
%!
&foo
&bar
&baz
&Foo::bar
&&
*foo
*bar
*baz
*Foo::bar
*[
<foo>
<bar>
<baz>
<Foo::bar>
<>
<*.c>
foo:
bar:
baz:
Foo::bar:
```

1.34. Выражения

Выражения (expressions) - это комбинации лексем, которые Perl воспринимает как осмысленные. Например, в строке:

```
print "Hello, world\n";
```

Perl воспринимает лексемы "print", "Hello, world\n" и ";". Perl воспринимает лексемы "print" и "Hello, world\n" как выражения. Подробнее см. perlfunc/Functions.

1.35. Операторы

Операторы (operators) - это знаки операций, которые используются для обозначения различных операций. Например, знак "+" обозначает операцию сложения, знак "-" обозначает операцию вычитания, а знак "*" обозначает операцию умножения. Подробнее см. perlfunc/Operators.

1.36. Литералы

Литералы (literals) - это числовые и строковые константы. Perl воспринимает следующие числовые константы:

```
123
-123
123.45
-123.45
0123
0x123
0b123
123e45
123.45e67
123.45e-67
```

Perl воспринимает следующие строковые константы:

```
'foo'
'bar'
'baz'
'Hello, world\n'
'Hello, world\t\t\n'
'Hello, world\'s\n'
'Hello, world\\s\n'
"foo"
"bar"
"baz"
"Hello, world\n"
"Hello, world\t\t\n"
"Hello, world\"s\n"
"Hello, world\$s\n"
"Hello, world\@s\n"
"Hello, world\\s\n"
```

1.37. Переменные типа

Perl воспринимает следующие переменные типа (variable type):

```
$foo
$bar
$baz
$Foo::bar
$^W
$"
@foo
@bar
@baz
@Foo::bar
@+
%foo
%bar
%baz
%Foo::bar
%!
&foo
&bar
&baz
&Foo::bar
&&
*foo
*bar
*baz
*Foo::bar
*[
<foo>
<bar>
<baz>
<Foo::bar>
<>
<*.c>
foo:
bar:
baz:
Foo::bar:
```

1.38. Выражения

Выражения (expressions) - это комбинации лексем, которые Perl воспринимает как осмысленные. Например, в строке:

```
print "Hello, world\n";
```

Perl воспринимает лексемы "print", "Hello, world\n" и ";". Perl воспринимает лексемы "print" и "Hello, world\n" как выражения. Подробнее см. perlfunc/Functions.

1.39. Операторы

Операторы (operators) - это знаки операций, которые используются для обозначения различных операций. Например, знак "+" обозначает операцию сложения, знак "-" обозначает операцию вычитания, а знак "*" обозначает операцию умножения. Подробнее см. perlfunc/Operators.

1.40. Литералы

Литералы (literals) - это числовые и строковые константы. Perl воспринимает следующие числовые константы:

```
123
-123
123.45
-123.45
0123
0x123
0b123
123e45
123.45e67
123.45e-67
```

Perl воспринимает следующие строковые константы:

```
'foo'
'bar'
'baz'
'Hello, world\n'
'Hello, world\t\t\n'
'Hello, world\'s\n'
'Hello, world\\s\n'
"foo"
"bar"
"baz"
"Hello, world\n"
"Hello, world\t\t\n"
"Hello, world\"s\n"
"Hello, world\$s\n"
"Hello, world\@s\n"
"Hello, world\\s\n"
```

1.41. Переменные типа

Perl воспринимает следующие переменные типа (variable type):

```
$foo
$bar
$baz
$Foo::bar
$^W
$"
@foo
@bar
@baz
@Foo::bar
@+
%foo
%bar
%baz
%Foo::bar
%!
&foo
&bar
&baz
&Foo::bar
&&
*foo
*bar
*baz
*Foo::bar
*[
<foo>
<bar>
<baz>
<Foo::bar>
<>
<*.c>
foo:
bar:
baz:
Foo::bar:
```

1.42. Выражения

Выражения (expressions) - это комбинации лексем, которые Perl воспринимает как осмысленные. Например, в строке:

```
print "Hello, world\n";
```

Perl воспринимает лексемы "print", "Hello, world\n" и ";". Perl воспринимает лексемы "print" и "Hello, world\n" как выражения. Подробнее см. perlfunc/Functions.

1.43. Операторы

Операторы (operators) - это знаки операций, которые используются для обозначения различных операций. Например, знак "+" обозначает операцию сложения, знак "-" обозначает операцию вычитания, а знак "*" обозначает операцию умножения. Подробнее см. perlfunc/Operators.

1.44. Литералы

Литералы (literals) - это числовые и строковые константы. Perl воспринимает следующие числовые константы:

```
123
-123
123.45
-123.45
0123
0x123
0b123
123e45
123.45e67
123.45e-67
```

Perl воспринимает следующие строковые константы:

```
'foo'
'bar'
'baz'
'Hello, world\n'
'Hello, world\t\t\n'
'Hello, world\'s\n'
'Hello, world\\s\n'
"foo"
"bar"
"baz"
"Hello, world\n"
"Hello, world\t\t\n"
"Hello, world\"s\n"
"Hello, world\$s\n"
"Hello, world\@s\n"
"Hello, world\\s\n"
```

1.45. Переменные типа

Perl воспринимает следующие переменные типа (variable type):

```
$foo
$bar
$baz
$Foo::bar
$^W
$"
@foo
@bar
@baz
@Foo::bar
@+
%foo
%bar
%baz
%Foo::bar
%!
&foo
&bar
&baz
&Foo::bar
&&
*foo
*bar
*baz
*Foo::bar
*[
<foo>
<bar>
<baz>
<Foo::bar>
<>
<*.c>
foo:
bar:
baz:
Foo::bar:
```

1.46. Выражения

Выражения (expressions) - это комбинации лексем, которые Perl воспринимает как осмысленные. Например, в строке:

```
print "Hello, world\n";
```

Perl воспринимает лексемы "print", "Hello, world\n" и ";". Perl воспринимает лексемы "print" и "Hello, world\n" как выражения. Подробнее см. perlfunc/Functions.

1.47. Операторы

Операторы (operators) - это знаки операций, которые используются для обозначения различных операций. Например, знак "+" обозначает операцию сложения, знак "-" обозначает операцию вычитания, а знак "*" обозначает операцию умножения. Подробнее см. perlfunc/Operators.

1.48. Литералы

Литералы (literals) - это числовые и строковые константы. Perl воспринимает следующие числовые константы:

```
123
-123
123.45
-123.45
0123
0x123
0b123
123e45
123.45e67
123.45e-67
```

Perl воспринимает следующие строковые константы:

```
'foo'
'bar'
'baz'
'Hello, world\n'
'Hello, world\t\t\n'
'Hello, world\'s\n'
'Hello, world\\s\n'
"foo"
"bar"
"baz"
"Hello, world\n"
"Hello, world\t\t\n"
"Hello, world\"s\n"
"Hello, world\$s\n"
"Hello, world\@s\n"
"Hello, world\\s\n"
```

1.49. Переменные типа

Perl воспринимает следующие переменные типа (variable type):

```
$foo
$bar
$baz
$Foo::bar
$^W
$"
@foo
@bar
@baz
@Foo::bar
@+
%foo
%bar
%baz
%Foo::bar
%!
&foo
&bar
&baz
&Foo::bar
&&
*foo
*bar
*baz
*Foo::bar
*[
<foo>
<bar>
<baz>
<Foo::bar>
<>
<*.c>
foo:
bar:
baz:
Foo::bar:
```

1.50. Выражения

Выражения (expressions) - это комбинации лексем, которые Perl воспринимает как осмысленные. Например, в строке:

```
print "Hello, world\n";
```

Perl воспринимает лексемы "print", "Hello, world\n" и ";". Perl воспринимает лексемы "print" и "Hello, world\n" как выражения. Подробнее см. perlfunc/Functions.

1.51. Операторы

Операторы (operators) - это знаки операций, которые используются для обозначения различных операций. Например, знак "+" обозначает операцию сложения, знак "-" обозначает операцию вычитания, а знак "*" обозначает операцию умножения. Подробнее см. perlfunc/Operators.

1.52. Литералы

Литералы (literals) - это числовые и строковые константы. Perl воспринимает следующие числовые константы:

```
123
-123
123.45
-123.45
0123
0x123
0b123
123e45
123.45e67
123.45e-67
```

Perl воспринимает следующие строковые константы:

```
'foo'
'bar'
'baz'
'Hello, world\n'
'Hello, world\t\t\n'
'Hello, world\'s\n'
'Hello, world\\s\n'
"foo"
"bar"
"baz"
"Hello, world\n"
"Hello, world\t\t\n"
"Hello, world\"s\n"
"Hello, world\$s\n"
"Hello, world\@s\n"
"Hello, world\\s\n"
```

1.53. Переменные типа

Perl воспринимает следующие переменные типа (variable type):

```
$foo
$bar
$baz
$Foo::bar
$^W
$"
@foo
@bar
@baz
@Foo::bar
@+
%foo
%bar
%baz
%Foo::bar
%!
&foo
&bar
&baz
&Foo::bar
&&
*foo
*bar
*baz
*Foo::bar
*[
<foo>
<bar>
<baz>
<Foo::bar>
<>
<*.c>
foo:
bar:
baz:
Foo::bar:
```

1.54. Выражения

Выражения (expressions) - это комбинации лексем, которые Perl воспринимает как осмысленные. Например, в строке:

```
print "Hello, world\n";
```

Perl воспринимает лексемы "print", "Hello, world\n" и ";". Perl воспринимает лексемы "print" и "Hello, world\n" как выражения. Подробнее см. perlfunc/Functions.

1.55. Операторы

Операторы (operators) - это знаки операций, которые используются для обозначения различных операций. Например, знак "+" обозначает операцию сложения, знак "-" обозначает операцию вычитания, а знак "*" обозначает операцию умножения. Подробнее см. perlfunc/Operators.

1.56. Литералы

Литералы (literals) - это числовые и строковые константы. Perl воспринимает следующие числовые константы:

```
123
-123
123.45
-123.45
0123
0x123
0b123
123e45
123.45e67
123.45e-67
```

Perl воспринимает следующие строковые константы:

```
'foo'
'bar'
'baz'
'Hello, world\n'
'Hello, world\t\t\n'
'Hello, world\'s\n'
'Hello, world\\s\n'
"foo"
"bar"
"baz"
"Hello, world\n"
"Hello, world\t\t\n"
"Hello, world\"s\n"
"Hello, world\$s\n"
"Hello, world\@s\n"
"Hello, world\\s\n"
```

1.57. Переменные типа

Perl воспринимает следующие переменные типа (variable type):

```
$foo
$bar
$baz
$Foo::bar
$^W
$"
@foo
@bar
@baz
@Foo::bar
@+
%foo
%bar
%baz
%Foo::bar
%!
&foo
&bar
&baz
&Foo::bar
&&
*foo
*bar
*baz
*Foo::bar
*[
<foo>
<bar>
<baz>
<Foo::bar>
<>
<*.c>
foo:
bar:
baz:
Foo::bar:
```

1.58. Выражения

Выражения (expressions) - это комбинации лексем, которые Perl воспринимает как осмысленные. Например, в строке:

```
print "Hello, world\n";
```

Perl воспринимает лексемы "print", "Hello, world\n" и ";". Perl воспринимает лексемы "print" и "Hello, world\n" как выражения. Подробнее см. perlfunc/Functions.

1.59. Операторы

Операторы (operators) - это знаки операций, которые используются для обозначения различных операций. Например, знак "+" обозначает операцию сложения, знак "-" обозначает операцию вычитания, а знак "*" обозначает операцию умножения. Подробнее см. perlfunc/Operators.

1.60. Литералы

Литералы (literals) - это числовые и строковые константы. Perl воспринимает следующие числовые константы:

```
123
-123
123.45
-123.45
0123
0x123
0b123
123e45
123.45e67
123.45e-67
```

Perl воспринимает следующие строковые константы:

```
'foo'
'bar'
'baz'
'Hello, world\n'
'Hello, world\t\t\n'
'Hello, world\'s\n'
'Hello, world\\s\n'
"foo"
"bar"
"baz"
"Hello, world\n"
"Hello, world\t\t\n"
"Hello, world\"s\n"
"Hello, world\$s\n"
"Hello, world\@s\n"
"Hello, world\\s\n"
```

1.61. Переменные типа

Perl воспринимает следующие переменные типа (variable type):

```
$foo
$bar
$baz
$Foo::bar
$^W
$"
@foo
@bar
@baz
@Foo::bar
@+
%foo
%bar
%baz
%Foo::bar
%!
&foo
&bar
&baz
&Foo::bar
&&
*foo
*bar
*baz
*Foo::bar
*[
<foo>
<bar>
<baz>
<Foo::bar>
<>
<*.c>
foo:
bar:
baz:
Foo::bar:
```

1.62. Выражения

Выражения (expressions) - это комбинации лексем, которые Perl воспринимает как осмысленные. Например, в строке:

```
print "Hello, world\n";
```

Perl воспринимает лексемы "print", "Hello, world\n" и ";". Perl воспринимает лексемы "print" и "Hello, world\n" как выражения. Подробнее см. perlfunc/Functions.

1.63. Операторы

Операторы (operators) - это знаки операций, которые используются для обозначения различных операций. Например, знак "+" обозначает операцию сложения, знак "-" обозначает операцию вычитания, а знак "*" обозначает операцию умножения. Подробнее см. perlfunc/Operators.

1.64. Литералы

Литералы (literals) - это числовые и строковые константы. Perl воспринимает следующие числовые константы:

```
123
-123
123.45
-123.45
0123
0x123
0b123
123e45
123.45e67
123.45e-67
```

Perl воспринимает следующие строковые константы:

```
'foo'
'bar'
'baz'
'Hello, world\n'
'Hello, world\t\t\n'
'Hello, world\'s\n'
'Hello, world\\s\n'
"foo"
"bar"
"baz"
"Hello, world\n"
"Hello, world\t\t\n"
"Hello, world\"s\n"
"Hello, world\$s\n"
"Hello, world\@s\n"
"Hello, world\\s\n"
```

1.65. Переменные типа

Perl воспринимает следующие переменные типа (variable type):

```
$foo
$bar
$baz
$Foo::bar
$^W
$"
@foo
@bar
@baz
@Foo::bar
@+
%foo
%bar
%baz
%Foo::bar
%!
&foo
&bar
&baz
&Foo::bar
&&
*foo
*bar
*baz
*Foo::bar
*[
<foo>
<bar>
<baz>
<Foo::bar>
<>
<*.c>
foo:
bar:
baz:
Foo::bar:
```

1.66. Выражения

Выражения (expressions) - это комбинации лексем, которые Perl воспринимает как осмысленные. Например, в строке:

```
print "Hello, world\n";
```

Perl воспринимает лексемы "print", "Hello, world\n" и ";". Perl воспринимает лексемы "print" и "Hello, world\n" как выражения. Подробнее см. perlfunc/Functions.

1.67. Операторы

Операторы (operators) - это знаки операций, которые используются для обозначения различных операций. Например, знак "+" обозначает операцию сложения, знак "-" обозначает операцию вычитания, а знак "*" обозначает операцию умножения. Подробнее см. perlfunc/Operators.

1.68. Литералы

Литералы (literals) - это числовые и строковые константы. Perl воспринимает следующие числовые константы:

```
123
-123
123.45
-123.45
0123
0x123
0b123
123e45
123.45e67
123.45e-67
```

Perl воспринимает следующие строковые константы:

```
'foo'
'bar'
'baz'
'Hello, world\n'
'Hello, world\t\t\n'
'Hello, world\'s\n'
'Hello, world\\s\n'
"foo"
"bar"
"baz"
"Hello, world\n"
"Hello, world\t\t\n"
"Hello, world\"s\n"
"Hello, world\$s\n"
"Hello, world\@s\n"
"Hello, world\\s\n"
```

1.69. Переменные типа

Perl воспринимает следующие переменные типа (variable type):

```
$foo
$bar
$baz
$Foo::bar
$^W
$"
@foo
@bar
@baz
@Foo::bar
@+
%foo
%bar
%baz
%Foo::bar
%!
&foo
&bar
&baz
&Foo::bar
&&
*foo
*bar
*baz
*Foo::bar
*[
<foo>
<bar>
<baz>
<Foo::bar>
<>
<*.c>
foo:
bar:
baz:
Foo::bar:
```

1.70. Выражения

Выражения (expressions) - это комбинации лексем, которые Perl воспринимает как осмысленные. Например, в строке:

```
print "Hello, world\n";
```

Perl воспринимает лексемы "print", "Hello, world\n" и ";". Perl воспринимает лексемы "print" и "Hello, world\n" как выражения. Подробнее см. perlfunc/Functions.

1.71. Операторы

Операторы (operators) - это знаки операций, которые используются для обозначения различных операций. Например, знак "+" обозначает операцию сложения, знак "-" обозначает операцию вычитания, а знак "*" обозначает операцию умножения. Подробнее см. perlfunc/Operators.

1.72. Литералы

Литералы (literals) - это числовые и строковые константы. Perl воспринимает следующие числовые константы:

```
123
-123
123.45
-123.45
0123
0x123
0b123
123e45
123.45e67
123.45e-67
```

Perl воспринимает следующие строковые константы:

```
'foo'
'bar'
'baz'
'Hello, world\n'
'Hello, world\t\t\n'
'Hello, world\'s\n'
'Hello, world\\s\n'
"foo"
"bar"
"baz"
"Hello, world\n"
"Hello, world\t\t\n"
"Hello, world\"s\n"
"Hello, world\$s\n"
"Hello, world\@s\n"
"Hello, world\\s\n"
```

1.73. Переменные типа

Perl воспринимает следующие переменные типа (variable type):

```
$foo
$bar
$baz
$Foo::bar
$^W
$"
@foo
@bar
@baz
@Foo::bar
@+
%foo
%bar
%baz
%Foo::bar
%!
&foo
&bar
&baz
&Foo::bar
&&
*foo
*bar
*baz
*Foo::bar
*[
<foo>
<bar>
<baz>
<Foo::bar>
<>
<*.c>
foo:
bar:
baz:
Foo::bar:
```

1.74. Выражения

Выражения (expressions) - это комбинации лексем, которые Perl воспринимает как осмысленные. Например, в строке:

```
print "Hello, world\n";
```

Perl воспринимает лексемы "print", "Hello, world\n" и ";". Perl воспринимает лексемы "print" и "Hello, world\n" как выражения. Подробнее см. perlfunc/Functions.

1.75. Операторы

Операторы (operators) - это знаки операций, которые используются для обозначения различных операций. Например, знак "+" обозначает операцию сложения, знак "-" обозначает операцию вычитания, а знак "*" обозначает операцию умножения. Подробнее см. perlfunc/Operators.

1.76. Литералы

Литералы (literals) - это числовые и строковые константы. Perl воспринимает следующие числовые константы:

```
123
-123
123.45
-123.45
0123
0x123
0b123
123e45
123.45e67
123.45e-67
```

Perl воспринимает следующие строковые константы:

```
'foo'
'bar'
'baz'
'Hello, world\n'
'Hello, world\t\t\n'
'Hello, world\'s\n'
'Hello, world\\s\n'
"foo"
"bar"
"baz"
"Hello, world\n"
"Hello, world\t\t\n"
"Hello, world\"s\n"
"Hello, world\$s\n"
"Hello, world\@s\n"
"Hello, world\\s\n"
```

1.77. Переменные типа

Perl воспринимает следующие переменные типа (variable type):

```
$foo
$bar
$baz
$Foo::bar
$^W
$"
@foo
@bar
@baz
@Foo::bar
@+
%foo
%bar
%baz
%Foo::bar
%!
&foo
&bar
&baz
&Foo::bar
&&
*foo
*bar
*baz
*Foo::bar
*[
<foo>
<bar>
<baz>
<Foo::bar>
<>
<*.c>
foo:
bar:
baz:
Foo::bar:
```

1.78. Выражения

Выражения (expressions) - это комбинации лексем, которые Perl воспринимает как осмысленные. Например, в строке:

```
print "Hello, world\n";
```

Perl воспринимает лексемы "print", "Hello, world\n" и ";". Perl воспринимает лексемы "print" и "Hello, world\n" как выражения. Подробнее см. perlfunc/Functions.

1.79. Операторы

Операторы (operators) - это знаки операций, которые используются для обозначения различных операций. Например, знак "+" обозначает операцию сложения, знак "-" обозначает операцию вычитания, а знак "*" обозначает операцию умножения. Подробнее см. perlfunc/Operators.

1.80. Литералы

Литералы (literals)

Что гораздо лучше: они графическое отображение остаются работать с любым текстовым полем:

```
Ctrl-Shift-Insert
Ctrl-Insert
Shift-Insert
Shift-Delete
Ctrl-Delete
```

Для этого надо переназначить указанные ниже. Для начала было сделано следующее переназначение:

```
Ctrl-C
```

Теперь, нажимая комбинацию Ctrl-C, клавиша, с Ctrl нажатой как клавиша для выбора без выделения, сохраняет выделенное в буфер обмена.

5.5.1. Использование графики

Для использования графики в Emacs — фреймов, кнопок, или менюшек, нужно в .emacs-файле выставить переменную window-system, которая может принимать значения x, mac, или nil. Если переменная window-system не определена, то по умолчанию принимается значение nil, что означает отсутствие графики.

5.5.2. Графическое отображение

Для отображения графики в Emacs — фреймов, фонтов, или менюшек — нужно в .emacs-файле выставить переменную window-system в соответствующее значение, а также выставить переменные, которые определяют, какие фонты, цвета, и другие параметры отображения использовать.

5.5.3. Графическое отображение в X-окнах

Для отображения в X-окнах нужно выставить window-system в x, и выставить X-специфические переменные, такие как x-bitmap-file, x-cursor-color, x-mouse-color, x-pointer-shape, x-sensitive, x-visible, и другие.

5.5.4. Графическое отображение в Mac OS X

Для отображения в Mac OS X нужно выставить window-system в mac, и выставить Mac-специфические переменные, такие как mac-allow-anti-aliasing, mac-command-key-is-meta, mac-command-modifier, mac-control-modifier, mac-option-key-is-meta, mac-option-modifier, mac-pass-control-to-system, mac-pass-command-to-system, mac-wheel-button-is-mouse-button, и другие.

5.5.5. Графическое, сенсорный экран и гаджет PDA-типа

Для отображения на сенсорном экране, или гаджете PDA-типа нужно выставить window-system в pocket-pc. Что, по-видимому, для большинства пользователей Emacs, не имеет смысла. Но, если вы, все-таки, хотите использовать Emacs на гаджете PDA-типа, то вам, скорее всего, нужно выставить также и переменные, которые определяют, какие фонты, цвета, и другие параметры отображения использовать.

5.5.6. Графическое, сенсорный экран и гаджет WIN-CE

Для отображения на сенсорном экране, или гаджете WIN-CE нужно выставить window-system в wince. Что, по-видимому, для большинства пользователей Emacs, не имеет смысла. Но, если вы, все-таки, хотите использовать Emacs на гаджете WIN-CE, то вам, скорее всего, нужно выставить также и переменные, которые определяют, какие фонты, цвета, и другие параметры отображения использовать.

5.6. Переназначение клавиатуры и мышиного

Переназначение — переназначение клавиатуры или переназначение мышиного интерфейса — в Emacs выполняется с помощью функции define-key, которая принимает три аргумента: keymap, key, и command. Первый аргумент, keymap, определяет какой keymap переназначается, второй аргумент, key, определяет какая клавиша или мышиная кнопка переназначается, и третий аргумент, command, определяет что будет выполнено при нажатии клавиши или мышиной кнопки, определенной в аргументе key.

Переназначение клавиатуры или мышиного интерфейса, в Emacs, обычно, выполняется в .emacs-файле, с помощью функции define-key. Например, следующая строка переназначает клавиатурную комбинацию Control-Shift-Insert на выполнение функции insert-buffer:

```
(define-key global-map [C-S-insert] 'insert-buffer)
```

Переназначение клавиатуры или мышиного интерфейса, в Emacs, также, может выполняться и с помощью функции global-set-key. Например, следующая строка переназначает клавиатурную комбинацию Control-Shift-Insert на выполнение функции insert-buffer:

```
(global-set-key [C-S-insert] 'insert-buffer)
```

Переназначение клавиатуры или мышиного интерфейса, в Emacs, также, может выполняться и с помощью функции local-set-key. Например, следующая строка переназначает клавиатурную комбинацию Control-Shift-Insert на выполнение функции insert-buffer:

```
(local-set-key [C-S-insert] 'insert-buffer)
```

Переназначение клавиатуры или мышиного интерфейса, в Emacs, также, может выполняться и с помощью функции define-prefix-command. Например, следующая строка переназначает клавиатурную комбинацию Control-Shift-Insert на выполнение функции insert-buffer:

```
(define-prefix-command 'my-keymap)
(define-key global-map [C-S-insert] 'my-keymap)
(define-key my-keymap [C-S-insert] 'insert-buffer)
```

Переназначение клавиатуры или мышиного интерфейса, в Emacs, также, может выполняться и с помощью функции define-key-after. Например, следующая строка переназначает клавиатурную комбинацию Control-Shift-Insert на выполнение функции insert-buffer:

```
(define-key-after global-map [C-S-insert] 'insert-buffer)
```

Переназначение клавиатуры или мышиного интерфейса, в Emacs, также, может выполняться и с помощью функции substitute-key-definition. Например, следующая строка переназначает клавиатурную комбинацию Control-Shift-Insert на выполнение функции insert-buffer:

```
(substitute-key-definition 'insert-buffer 'insert-buffer global-map)
```

Переназначение клавиатуры или мышиного интерфейса, в Emacs, также, может выполняться и с помощью функции global-unset-key. Например, следующая строка переназначает клавиатурную комбинацию Control-Shift-Insert на выполнение функции insert-buffer:

```
(global-unset-key [C-S-insert])
(global-set-key [C-S-insert] 'insert-buffer)
```

Переназначение клавиатуры или мышиного интерфейса, в Emacs, также, может выполняться и с помощью функции local-unset-key. Например, следующая строка переназначает клавиатурную комбинацию Control-Shift-Insert на выполнение функции insert-buffer:

```
(local-unset-key [C-S-insert])
(local-set-key [C-S-insert] 'insert-buffer)
```

Переназначение клавиатуры или мышиного интерфейса, в Emacs, также, может выполняться и с помощью функции define-prefix-command. Например, следующая строка переназначает клавиатурную комбинацию Control-Shift-Insert на выполнение функции insert-buffer:

```
(define-prefix-command 'my-keymap)
(local-set-key [C-S-insert] 'my-keymap)
(local-set-key [C-S-insert] 'insert-buffer)
```

Переназначение клавиатуры или мышиного интерфейса, в Emacs, также, может выполняться и с помощью функции define-key-after. Например, следующая строка переназначает клавиатурную комбинацию Control-Shift-Insert на выполнение функции insert-buffer:

```
(define-key-after local-map [C-S-insert] 'insert-buffer)
```

Переназначение клавиатуры или мышиного интерфейса, в Emacs, также, может выполняться и с помощью функции substitute-key-definition. Например, следующая строка переназначает клавиатурную комбинацию Control-Shift-Insert на выполнение функции insert-buffer:

```
(substitute-key-definition 'insert-buffer 'insert-buffer local-map)
```

Переназначение клавиатуры или мышиного интерфейса, в Emacs, также, может выполняться и с помощью функции local-unset-key. Например, следующая строка переназначает клавиатурную комбинацию Control-Shift-Insert на выполнение функции insert-buffer:

```
(local-unset-key [C-S-insert])
(local-set-key [C-S-insert] 'insert-buffer)
```

Переназначение клавиатуры или мышиного интерфейса, в Emacs, также, может выполняться и с помощью функции define-prefix-command. Например, следующая строка переназначает клавиатурную комбинацию Control-Shift-Insert на выполнение функции insert-buffer:

```
(define-prefix-command 'my-keymap)
(local-set-key [C-S-insert] 'my-keymap)
(local-set-key [C-S-insert] 'insert-buffer)
```

5.7. Сенсорная клавиатура и сенсорный экран

Клавиатура и сенсорный экран, или сенсорная клавиатура, в Emacs, выполняется с помощью:

- Menubar
- Tool-bar
- Tooltips
- Scrollbar
- Tabbar
- Tabbar-ruler

5.7.1. Menubar

Менюшки выполняются с помощью функции define-key. Например, следующая строка выполняет переназначение менюшки Tools:

```
(define-key global-map [menu-bar tools] '(menu-item "Tools" nil :help "Tools"))
```

Менюшки, также, выполняются и с помощью easy-menu-define. Например, следующая строка выполняет переназначение менюшки Tools:

```
(easy-menu-define
  nil
  global-map
  "Tools"
  '("Tools"
    :help "Tools"
    ["Foo" foo t]
    ["Bar" bar t]
    ["Baz" baz t]
    ["Quux" quux t]))
```

5.7.2. Tool-bar

Tool-bar выполняется переназначением tool-bar-map. Например, следующая строка выполняет переназначение tool-bar-map:

```
(setq tool-bar-map
      '(keymap
        (foo menu-item "Foo" foo :image (image :type xpm :file "foo.xpm"))
        (bar menu-item "Bar" bar :image (image :type xpm :file "bar.xpm"))
        (baz menu-item "Baz" baz :image (image :type xpm :file "baz.xpm"))
        (quux menu-item "Quux" quux :image (image :type xpm :file "quux.xpm"))))
```

5.7.3. Tooltips

Tooltips выполняется с помощью tooltip-mode. Например, следующая строка выполняет переназначение tooltip-mode:

```
(tooltip-mode t)
```

5.7.4. Scrollbar

Scrollbar выполняется с помощью scroll-bar-mode. Например, следующая строка выполняет переназначение scroll-bar-mode:

```
(scroll-bar-mode t)
```

5.7.5. Tabbar

Tabbar выполняется с помощью tabbar-mode. Например, следующая строка выполняет переназначение tabbar-mode:

```
(tabbar-mode t)
```

5.7.6. Tabbar-ruler

Tabbar-ruler выполняется с помощью tabbar-ruler-mode. Например, следующая строка выполняет переназначение tabbar-ruler-mode:

```
(tabbar-ruler-mode t)
```

5.8. Переназначение мышиного

Переназначение мышиного выполняется, в Emacs, с помощью define-key. Например, следующая строка выполняет переназначение мышиного:

```
(define-key global-map [mouse-3] '(menu-item "Mouse" nil :help "Mouse"))
```

5.8.1. Мышиная кнопка

Мышиная кнопка выполняется с помощью define-key. Например, следующая строка выполняет переназначение мышиной кнопки:

```
(define-key global-map [mouse-3] '(menu-item "Mouse" nil :help "Mouse"))
```

5.8.2. Мышиное колесо

Мышиное колесо выполняется с помощью define-key. Например, следующая строка выполняет переназначение мышиного колеса:

```
(define-key global-map [mouse-4] '(menu-item "Mouse" nil :help "Mouse"))
(define-key global-map [mouse-5] '(menu-item "Mouse" nil :help "Mouse"))
```

5.8.3. Мышиная клавиатура

Мышиная клавиатура выполняется с помощью define-key. Например, следующая строка выполняет переназначение мышиной клавиатуры:

```
(define-key global-map [mouse-3] '(menu-item "Mouse" nil :help "Mouse"))
```

5.8.4. Сенсорный экран

Сенсорный экран выполняется с помощью define-key. Например, следующая строка выполняет переназначение сенсорного экрана:

```
(define-key global-map [mouse-3] '(menu-item "Mouse" nil :help "Mouse"))
```

5.9. Переназначение приложения

Переназначение приложения выполняется с помощью define-key. Например, следующая строка выполняет переназначение приложения:

```
(define-key global-map [mouse-3] '(menu-item "Mouse" nil :help "Mouse"))
```

5.10. Сенсорное приложение

Сенсорное приложение выполняется с помощью define-key. Например, следующая строка выполняет переназначение сенсорного приложения:

```
(define-key global-map [mouse-3] '(menu-item "Mouse" nil :help "Mouse"))
```

Содержание Назад

Copyright © 2010-2012 Andrey Zhidenkov
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

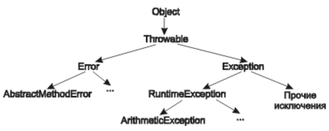
Приложение A. Решение методов

Автор(ы): Ник Шрайберр, Стейн Ван Ван, Ян Ван Ван Ван

Приложение Б
Runtime-исключения в Java

Компьютер не бывает эмоциональным. Он может дать точное математическое описание, но забудет об интонации.
Фрэнк Заппа

Runtime-система Java возбуждает исключения двух основных видов: runtime-исключения, расширяющие класс *RuntimeException*, и ошибки, которые расширяют класс *Error*. Исключения обоих видов являются непроверяемыми (см. раздел 7.3). Верхняя часть иерархии исключений выглядит следующим образом:



Исключения *Error* сигнализируют об очень серьезных проблемах, после которых программа обычно завершается, и которые никогда (или почти никогда) не должны перехватываться. Исключения *Error* не являются расширениями *RuntimeException*, так что программист, пытающийся написать универсальное условие *catch* для перехвата всех исключений *Exception* и *RuntimeException* (обычно делать этого не следует) не сможет перехватить исключения *Error*. Разумеется, после возникновения любого исключения будут выполнены условия *finally* операторов *try*, так как все исключения, в том числе и *Error*, просматривают стек вызовов. Следовательно, вы всегда сможете выполнить необходимые завершающие действия.

Программист может самостоятельно расширить классы *RuntimeException* и *Error*, чтобы создать свои собственные варианты непроверяемых исключений — то есть таких исключений, которые можно возбуждать без указания их в условии *throws*. Мы сообщаем об этом по единственной причине — чтобы вы знали, что этого делать не следует. Условие *throws* предусмотрено именно для того, чтобы при вызове метода были видны все возможные аспекты его поведения. Порождая свое исключение от *RuntimeException* или *Error*, вы сообщаете о нем ложные сведения (будто оно запускается runtime-системой). Кроме того, другие разработчики, читающие вашу программу, полагают, что условие *throws* дает им информацию о возможном поведении вашего метода; вы нарушаете это предположение.

Даже если вы пишете программу для своего собственного использования, не стоит создавать непроверяемые исключения: программисты, не полностью понимающие работу вашего кода, могут упустить нечто важное. Кроме того, вероятно, что через несколько месяцев после написания программы именно *вы* окажетесь тем человеком, который будет изменять ее без полного понимания происходящего. Одно из правил создания понятных программ — считать классы *RuntimeException* и *Error* нерасширяемыми.

Все классы *Error* и *RuntimeException* содержат по меньшей мере два конструктора: один вызывается без аргументов, а второй получает объект *String* с описанием. Исключения, которые прямо или косвенно расширяют *RuntimeException* или *Error*, не объявляются в условии *throws*, поскольку они могут произойти в любой момент, что делает их объявление излишним.

Исключение *CloneNotSupportedException* непосредственно порождается от класса *Exception*, поскольку каждая программа, которая вызывает метод *clone*, возбуждающий данное исключение, должна явным образом его обработать. Оно рассмотрено в разделе "Дублирование объектов.

Настоящая глава делится на две части — одна посвящена классам *RuntimeException*, а другая — классам *Error*. Для каждого исключения приводится его значение, описание ситуации, в которой оно возбуждается, а также все дополнительные конструкторы.

Б.1 Классы *RuntimeException*

ArithmeticException extends RuntimeException

Возникла исключительная ситуация во время вычислений (например, деление целого числа на ноль).

ArrayStoreException *extends RuntimeException*

Попытка сохранения в массиве объекта неверного типа.

ClassCastException *extends RuntimeException*

Попытка недопустимого приведения типа.

IllegalArgumentException *extends RuntimeException*

Метод получил неверный аргумент (например, метод *String.equals* вызван для объекта, который не относится к типу *String*).

IllegalMonitorStateException *extends RuntimeException*

Механизм *wait/notify* использован за пределами синхронного кода.

IllegalThreadStateException *extends IllegalArgumentException*

Состояние потока не допускает выполнения требуемой операции.

IndexOutOfBoundsException *extends RuntimeException*

Runtime-система генерирует это исключение при выходе индекса массива или объекта *String* за пределы диапазона допустимых значений.

NegativeArraySizeException *extends RuntimeException*

Попытка создания массива отрицательного размера.

NullPointerException *extends RuntimeException*

Для доступа к полю или методу использована *null*-ссылка. Это же исключение сигнализирует о передаче методу параметра *null*, если для данного параметра это значение является недопустимым. Используется аналогично *IllegalArgumentException*.

NumberFormatException *extends IllegalArgumentException*

Неверное содержимое строки, в которой должно было находиться число. Исключение возбуждается такими методами, как *Integer, parseInt*.

SecurityException *extends RuntimeException*

Попытка выполнения действия, запрещенного системой безопасности — обычно объектом *SecurityManager* для текущего runtime-контекста.

Б.2 Классы Error

AbstractMethodError extends IncompatibleClassChangeError

Вызван абстрактный метод. Это может произойти лишь в очень редких случаях, так что вы никогда не столкнетесь с этим исключением.

ClassFormatError *extends LinkageError*

Загружаемый класс или интерфейс имеет неверный формат (обычно это связано с использованием "преобразованных" (mangled) имен).

IllegalAccessError *extends IncompatibleClassChangeError*

Исключение неразрешенного доступа.

IncompatibleClassChangeError *extends linkageError*

При загрузке класса или интерфейса было обнаружено изменение, несовместимое с информацией об этом классе или интерфейсе. Например, в период времени между компиляцией класса и компиляцией использующей его программы, из класса был удален незакрытый метод.

InstantiationError *extends IncompatibleClassChangeError*

Интерпретатор попытался создать объект абстрактного класса или интерфейса.

InternalError *extends VirtualMachineError*

Произошел внутренний сбой runtime-системы. В нормальных условиях такая ошибка не должна возникнуть.

LinkageError *extends Error*

Исключения класса *LinkageError* и его подклассов означают, что класс тем или иным образом зависит от другого класса и что связь между ними не может быть установлена.

NoClassDefFoundError *extends LinkageError*

Нужный класс не найден.

NoSuchFieldError *extends IncompatibleClassChangeError*

Поле отсутствует в классе или интерфейсе.

NoSuchMethodError *extends IncompatibleClassChangeError*

Метод отсутствует в классе или интерфейсе.

OutOfMemoryError *extends VirtualMachineError*

Нехватка памяти.

StackOverflowError *extends VirtualMachineError*

Переполнение стека. Может свидетельствовать о бесконечной рекурсии.

ThreadDeath *extends Error*

Исключение *ThreadDeath* возбуждается потоком-"жертвой" при его уничтожении методом *thread.stop*. Если исключение *ThreadDeath* перехватывается, его необходимо возбудить повторно, чтобы поток был уничтожен. Если *ThreadDeath* не перехватывается, то обработчик ошибок верхнего уровня не выводит никаких сообщений.

UnknownError *extends VirtualMachineError*

Произошла неизвестная, но серьезная ошибка.

UnsatisfiedLinkError *extends LinkageError*

Ошибка связывания внутри родного метода. Обычно это означает, что библиотека, реализующая родной метод, содержит неопределенные символы, которые не были найдены ни в одной библиотеке.

VerifyError *extends LinkageError*

Произошла ошибка верификации — то есть во время загрузки класс не прошел проверку, в ходе которой обычно выясняется не нарушает ли класс каких-нибудь требований безопасности Java.

VirtualMachineError *extends Error*

Нарушена работа виртуальной машины, или наблюдается нехватка ресурсов.

Приложение B
Полезные таблицы

Таблица 1. Ключевые слова

abstract	double	int	super
boolean	else	interface	switch
break	extends	long	synchronized
byte	final	native	this
case	finally	new	throw
catch	float	package	throws
char	for	private	transient‹
class	goto‹	protected	try
const‹	if	public	void
continue	implements	return	volatile
default	import	short	while
do	instanceof	static	

Ключевые слова, помеченные символом ‹, в настоящее время не используются

**Таблица 2. Специальные символы, содержащие **

Последовательность	Значение
\n	переход на новую строку (*\u000A*)
\t	табуляция (*\u0009*)
\b	забой (*\u0008*)
\r	ввод (*\u000D*)
\f	подача листа (*\u000C*)
\\	обратная косая черта (*\u005C*)
\'	апостроф (*\u0027*)
\"	кавычка (*\u0022*)
\ddd	символ в восьмеричном представлении, где каждое *d* соответствует восьмеричной цифре от 0 до 7
\uddd	символ Unicode, где каждое *d* соответствует шестнадцатеричной цифре (*0–9, a–f, A–F*)

Таблица 3. Приоритет операторов

постфиксные операторы	[] . (параметры) expr++ expr--
унарные операторы	++expr --expr +expr -expr ~ !
создание и преобразование типа	new (тип)expr
операторы умножения/деления	* / %
операторы сложения/ вычитания	+ -
операторы сдвига	<<<< >>>> >>>>>>
операторы отношения	<< >> >>= <<= instanceof
операторы равенства	== !=
поразрядное И	&
поразрядное исключающее ИЛИ	^
поразрядное включающее ИЛИ	|
логическое И	&&
логическое ИЛИ	||
условный оператор	?:
операторы присвоения	= += -= *= /= %= >>>>= <<<<= >>>>>>= &= ^= | =

Таблица 4. Цифры Unicode

Unicode	Описание
\u0030–\u0039	Цифры ISO-latin-1 (и ASCII)
\u0660–\u0669	Арабско-индийские цифры
\u06f0–\u06f9	Восточные арабско-индийские цифры
\u0966–\u096f	Цифры деванагари
\u09e6–\u09ef	Цифры бенгали
\u0a66–\u0a6f	Цифры гурмукхи
\u0ae6–\u0aef	Цифры гуджарати
\u0b66–\u0b6f	Цифры ория
\u0be7–\u0bef	Тамильские цифры (только девять — без нуля)
\u0c66–\u0c6f	Цифры телугу
\u0ce6–\u0cef	Цифры каннада
\u0d66–\u0d6f	Малайские цифры
\u0e50–\u0e59	Тайские цифры
\u0ed0–\u0ed9	Цифры лао
\uff10–\uff19	Цифры полной ширины

Таблица 5. Буквы и цифры Unicode

\u0041–\u005a	Буквы верхнего регистра ISO-latin-1 и ASCII (*'A'–'Z'*)
\u0061–\u007a	Буквы нижнего регистра ISO-latin-1 и ASCII (*'a'–'z'*)
\u00c0–\u00d6	Дополнительные буквы ISO-latin-1
\u00d8–\u00f6	Дополнительные буквы ISO-latin-1
\u00f8–\u00ff	Дополнительные буквы ISO-latin-1
\u0100–\u1fff	Расширенная кодировка Latin-A, расширенная кодировка Latin-B, расширения IPA, буквы-модификаторы интервалов, диакритические знаки, базовый греческий алфавит, греческий и коптский алфавиты, кириллица, армянский, иврит расширенный-A, базовый иврит, иврит расширенный-B, базовый арабский, расширенный арабский, деванагари, бенгали, гурмукхи, гуджарати, ория, тамильский, телугу, каннада, малайский, тайский, лао, базовый грузинский, расширенный грузинский, хангульский, латинский расширенный дополнительный, греческий расширенный
\u3040–\u9fff	Хирагана, катакана, бопомофо, хангульский совместимый, CJK, символы и месяцы CJK, CJK совместимый, хангульский, хангульский дополнительный-A, хангульский дополнительный-B, единые идеографы CJK
\uf900–\ufdff	Совместимые идеографы CJK, алфавитные формы, арабские презентационные формы-A
\ufe70–\ufefe	Арабские презентационные формы-B
\uff10–\uff19	Цифры полной ширины
\uff21–\uff3a	Латинский полной ширины, верхний регистр
\uff41–\uff5a	Латинский полной ширины, нижний регистр
\uffd6–\uffdc	Катакана и хангульский половинной ширины

Примечание: Символ Unicode является буквой или цифрой, если он принадлежит одному из диапазонов, содержащихся в таблице, и также определен как символ Unicode.

Примечание: Символ Unicode является буквой, если он присутствует в таблице "Буквы и цифры Unicode", но отсутствует в таблице "Цифры Unicode".

Таблица 6. Java 1.0 и Java 1.0.2: Отличия между Java 1.0 и Java 1.0.2, существенные для данной книги (с разделами, к которым они относятся)

- Константы *MIN_VALUE* и *MAX_VALUE* класса *Character* в Java 1.0 ошибочно присутствовали в классе *Boolean*. См. раздел 13.5.
- В классах *String* и *Character* в Java 1.0 некоторые свойства символов (принадлежность к верхнему/нижнему регистру, цифрам и т.д.) определялись только для подмножества символов Unicode, принадлежащего к набору ISO-Latin-1 (с *\u0000* по *\u00ff*); все символы за пределами этого диапазона считались буквами без регистра. Кроме того, отсутствовали методы класса *Character*, возвращавшие сведение о классе символа помимо принадлежности к верхнему или нижнему регистру (например, методы заглавного регистра и *isLetter*). См. раздел 8.2, раздел 8.4 и раздел 13.5.
- Java 1.0 не гарантирует, что литералам *String* с одинаковыми значениями соответствуют одинаковые ссылки, хотя иногда это было так. См. раздел 8.2.
- Список букв и цифр, используемых в идентификаторах Java 1.0, несколько отличается от списка Java 1.0.2 за пределами диапазона ISO-Latin-1. См. табл. 4 и табл. 5.
- В Java разрешалась (и даже ошибочно наделялась смыслом) комбинация ключевых слов *private protected*.
- Классы-оболочки *Integer* и *Long* в Java 1.0 не содержали методов *toHexString*, *toOctalString* и *toBinaryString*. См. раздел 13.7 и раздел 13.8.

Содержание

Глава 7
РАСШИРЕНИЯ КЛАССОВ

Расширения классов позволяют вносить изменения в уже существующие классы, не изменяя исходный код.

Не всегда исходные коды библиотечных классов доступны для изменения, или же, даже если исходные коды доступны, внесение изменений в библиотечные классы может нарушить их работоспособность. Кроме того, даже если вы пишете свои собственные классы, вам может потребоваться расширить их функциональность, не изменяя исходный код, например, если вы хотите использовать свои классы в нескольких различных приложениях, которые требуют различной функциональности.

Язык Delphi позволяет изменять классы, не внося изменений в определения классов, путем объявления специальных типов, называемых расширениями классов. Расширения классов позволяют добавлять новые методы к уже существующим классам, а также изменять поведение уже существующих методов. Расширения классов не могут добавлять новые поля к классам, но могут добавлять свойства, которые используют уже существующие поля.

7.1. Расширяемый класс

Расширения классов объявляются с помощью ключевого слова class helper. Например, следующий фрагмент кода объявляет расширение класса Integer:

```
type
  TIntHelper = class helper
    function Sqr: Integer;
    procedure Inc;
    procedure Inc(Value: Integer);
    function ToHexStr: String;
    function ToHexStr(Digits: Integer): String;
    function ToString: String;
    function ToString(Digits: Integer): String;
    function ToString(Digits, Base: Integer): String;
    class function Random: Integer;
    class function Random(Range: Integer): Integer;
    class function ParseHexStr(S: String): Integer;
    class function Parse(S: String): Integer;
    class function Parse(S: String; Base: Integer): Integer;
    class function MaxValue: Integer;
    class function MinValue: Integer;
    property Sqrt: Integer read Sqrt;
    property Sign: TValueSign read Sign;
    property IsPrime: Boolean read IsPrime;
    property IsPowerOfTwo: Boolean read IsPowerOfTwo;
    property ToBoolean: Boolean read ToBoolean;
    property ToChar: Char read ToChar;
    property ToDateTime: TDateTime read ToDateTime;
    property ToExtended: Extended read ToExtended;
    property ToFloat: Float read ToFloat;
    property ToGUID: TGUID read ToGUID;
    property ToSingle: Single read ToSingle;
    property ToUnixDateTime: Int64 read ToUnixDateTime;
    property ToVarRec: TVarRec read ToVarRec;
    property ToVarRec[Base: Integer]: TVarRec read ToVarRec;
    property ToVarRec[Digits, Base: Integer]: TVarRec read ToVarRec;
    property ToVarRec[Range: Integer]: TVarRec read ToVarRec;
    property ToVarRec[Range, Base: Integer]: TVarRec read ToVarRec;
    property ToVarRec[Range, Digits, Base: Integer]: TVarRec read ToVarRec;
    property ToVarRec[S: String]: TVarRec read ToVarRec;
    property ToVarRec[S: String; Base: Integer]: TVarRec read ToVarRec;
    property ToVarRec[S: String; Range: Integer]: TVarRec read ToVarRec;
    property ToVarRec[S: String; Range, Base: Integer]: TVarRec read ToVarRec;
    property ToVarRec[S: String; Range, Digits, Base: Integer]: TVarRec read ToVarRec;
    property ToVarRec[Value: Integer]: TVarRec read ToVarRec;
    property ToVarRec[Value: Integer; Base: Integer]: TVarRec read ToVarRec;
    property ToVarRec[Value: Integer; Digits, Base: Integer]: TVarRec read ToVarRec;
    property ToVarRec[Value: Integer; Range: Integer]: TVarRec read ToVarRec;
    property ToVarRec[Value: Integer; Range, Base: Integer]: TVarRec read ToVarRec;
    property ToVarRec[Value: Integer; Range, Digits, Base: Integer]: TVarRec read ToVarRec;
    property ToVarRec[Value: Integer; S: String]: TVarRec read ToVarRec;
    property ToVarRec[Value: Integer; S: String; Base: Integer]: TVarRec read ToVarRec;
    property ToVarRec[Value: Integer; S: String; Range: Integer]: TVarRec read ToVarRec;
    property ToVarRec[Value: Integer; S: String; Range, Base: Integer]: TVarRec read ToVarRec;
    property ToVarRec[Value: Integer; S: String; Range, Digits, Base: Integer]: TVarRec read ToVarRec;
    property ToVarRec[Value: Integer; Range: Integer; Digits, Base: Integer]: TVarRec read ToVarRec;
    property ToVarRec[Value: Integer; Range, Digits, Base: Integer]: TVarRec read ToVarRec;
    property ToVarRec[Range: Integer; Digits, Base: Integer]: TVarRec read ToVarRec;
    property ToVarRec[Range, Digits, Base: Integer]: TVarRec read ToVarRec;
    property ToVarRec[Digits, Base: Integer]: TVarRec read ToVarRec;
    property ToVarRec[Digits, Base: Integer]: TVarRec read ToVarRec;
    property ToVarRec[Base: Integer]: TVarRec read ToVarRec;
    property ToVarRec[Base: Integer]: TVarRec read ToVarRec;
    property ToVarRec: TVarRec read ToVarRec;
    property ToVarRec: TVarRec read ToVarRec;
    property ToVarRec
```

```
        Sector_Fee = Income*0.1
        Income = Income - Sector_Fee
        Income = Income - 2000
    ПО Поощрение_в_конкурсе_Конкурс = Конкурсант_в_ЦО ?
        Поощрение_в_конкурсе = 1000
```

Новая и полная версия для грамотных пользователей была представлена в составе версии Delphi 2010Pro, состоящая из трехмерных компонентов, поддерживающих технологии Microsoft Kinect, а также поддержка нескольких сенсорных экранов и поддержка нескольких мышей для каждого пользователя одновременно (для игровых приложений).

Язык 1С:Предприятие 8 был использован в системе учетного ресурсного планирования (учетно-ресурсная система) Российского отделения Microsoft, которая была внедрена в ноябре 2008 года. Система была разработана специалистами Российского отделения Microsoft совместно с компанией «1С».

Компания Borland представила для Delphi 2009 новую версию компонентов для работы с базами данных dbExpress, которая позволяет использовать для доступа к базам данных Microsoft SQL Server, Oracle, Sybase, IBM DB2 и InterBase. Также в Delphi 2009 добавлена поддержка Unicode, а также поддержка нескольких сенсорных экранов и поддержка нескольких мышей для каждого пользователя одновременно (для игровых приложений).

Компания Borland представила для Delphi 2009 новую версию компонентов для работы с базами данных dbExpress, которая позволяет использовать для доступа к базам данных Microsoft SQL Server, Oracle, Sybase, IBM DB2 и InterBase. Также в Delphi 2009 добавлена поддержка Unicode, а также поддержка нескольких сенсорных экранов и поддержка нескольких мышей для каждого пользователя одновременно (для игровых приложений).

```
Procedure Calculate_Salary(Sector_Fee:Integer);
    Sector_Fee = Income*0.1
    Income = Income - Sector_Fee
    Income = Income - 2000
    ПО Поощрение_в_конкурсе_Конкурс = Конкурсант_в_ЦО ?
        Поощрение_в_конкурсе = 1000
        Income = Income + Поощрение_в_конкурсе
    ПО Доплата_Ученая_степень = Ученая_степень ?
        Доплата_Ученая_степень = 1000
        Income = Income + Доплата_Ученая_степень
    ПО Доплата_Ученое_звание = Ученое_звание ?
        Доплата_Ученое_звание = 1000
        Income = Income + Доплата_Ученое_звание
    ПО Доплата_Стаж = Стаж > 10 ?
        Доплата_Стаж = 1000
        Income = Income + Доплата_Стаж
    ПО Доплата_Выслуга_лет = Выслуга_лет > 15 ?
        Доплата_Выслуга_лет = 1000
        Income = Income + Доплата_Выслуга_лет
    ПО Доплата_Дети = Дети > 3 ?
        Доплата_Дети = 1000
        Income = Income + Доплата_Дети
    ПО Доплата_Инвалидность = Инвалидность ?
        Доплата_Инвалидность = 1000
        Income = Income + Доплата_Инвалидность
    ПО Доплата_Ветеран = Ветеран ?
        Доплата_Ветеран = 1000
        Income = Income + Доплата_Ветеран
    ПО Доплата_Ребенок_инвалид = Ребенок_инвалид ?
        Доплата_Ребенок_инвалид = 1000
        Income = Income + Доплата_Ребенок_инвалид
    ПО Доплата_Кормящая_мать = Кормящая_мать ?
        Доплата_Кормящая_мать = 1000
        Income = Income + Доплата_Кормящая_мать
    ПО Доплата_Ветеран_труда = Ветеран_труда ?
        Доплата_Ветеран_труда = 1000
        Income = Income + Доплата_Ветеран_труда
```

Не следует путать язык шаблонов с языком, который предоставляет разработчикам, использующим технологии Microsoft, возможность использовать для доступа к данным различных баз данных Microsoft SQL Server, Oracle, Sybase, IBM DB2 и InterBase. Также в Delphi 2009 добавлена поддержка Unicode, а также поддержка нескольких сенсорных экранов и поддержка нескольких мышей для каждого пользователя одновременно (для игровых приложений).

Компания Borland представила для Delphi 2009 новую версию компонентов для работы с базами данных dbExpress, которая позволяет использовать для доступа к базам

Глава V
ПОТОКИ

Потоки являются средством для параллельного выполнения нескольких фрагментов кода в пределах одного процесса. Поскольку потоки могут использовать общие данные, они являются эффективным средством для обмена информацией между различными частями программы.

Каждый поток имеет свою собственную последовательность выполнения. Если в ходе выполнения потока встречается вызов какой-нибудь функции, то выполнение этого потока приостанавливается, пока не будет выполнена эта функция.

Каждый поток работает независимо от остальных потоков. Поэтому, если в ходе выполнения потока встречается вызов какой-нибудь функции, то выполнение этого потока приостанавливается, пока не будет выполнена эта функция, но при этом выполнение остальных потоков продолжается. Поэтому, если, например, в ходе выполнения одного из потоков происходит обращение к диску, то в ходе обращения к диску выполнение остальных потоков может продолжаться, если, например, в ходе выполнения одного из потоков происходит обращение к диску, то в ходе обращения к диску выполнение остальных потоков может продолжаться.

Поскольку потоки могут использовать общие данные, то для обеспечения корректной работы программы необходимо, чтобы в ходе выполнения каждого из потоков в ходе выполнения каждого из потоков в ходе выполнения каждого из потоков

Уполномоченное **Заказчиком** лицо подписи

доверенность **№ 1 от 01.01.2021**

Адрес электронной почты **ivan@ivanov.ru**

Адрес электронной почты **ivan@ivanov.ru**

Уполномоченное **исполнителем** лицо подписи

доверенность **№ 1 от 01.01.2021**

Адрес электронной почты **petrov@ПроектИнвест.ru**

Проектная, рабочая и сметная документация прилагается на электронном носителе, Акты будут подписываться Заказчиком и Исполнителем в бумажном виде.

ФИО **Иванов Иван Иванович**

Адрес электронной почты **ivan@ivanov.ru** Номер телефона **+79000000000**

ФИО **Петров Петр Петрович**

Адрес электронной почты **petrov@ПроектИнвест.ru** Номер телефона **+79000000000**

Настоящий Договор подписан сторонами в электронном виде с использованием электронной подписи, в соответствии с требованиями Федерального закона от 06.04.2011 г. № 63-ФЗ «Об электронной подписи» и Постановлением Правительства РФ от 25.01.2013 г. № 33 «Об использовании простой электронной подписи при оказании государственных и муниципальных услуг».

Каждая из сторон признает, что документы, подписанные с использованием электронной подписи, имеют юридическую силу, равную силе документов, подписанных собственноручной подписью уполномоченных представителей сторон, и не могут быть оспорены только на основании того, что они подписаны электронной подписью.

Каждая из сторон обязуется обеспечивать конфиденциальность ключей электронной подписи, принадлежащих уполномоченным представителям сторон, и не допускать использования ключей электронной подписи, принадлежащих уполномоченным представителям сторон, третьими лицами.

Каждая из сторон обязуется в случае нарушения конфиденциальности ключей электронной подписи, принадлежащих уполномоченным представителям сторон, немедленно уведомить другую сторону.

Каждая из сторон обязуется в случае прекращения полномочий уполномоченных представителей сторон немедленно уведомить другую сторону.

Каждая из сторон обязуется в случае компрометации ключей электронной подписи, принадлежащих уполномоченным представителям сторон, немедленно уведомить другую сторону.

Каждая из сторон обязуется в случае изменения данных уполномоченных представителей сторон немедленно уведомить другую сторону.

Каждая из сторон обязуется в случае изменения данных, указанных в разделе «Адреса, реквизиты и подписи сторон», немедленно уведомить другую сторону.

Каждая из сторон обязуется в случае изменения данных, указанных в разделе «Адреса, реквизиты и подписи сторон», немедленно уведомить другую сторону.

Уполномоченное лицо Заказчика подписи сторон на электронном носителе

ФИО **Иванов Иван Иванович**

Уполномоченное лицо Исполнителя подписи сторон на электронном носителе

ФИО **Петров Петр Петрович**

Уполномоченное лицо Заказчика подписи сторон на бумажном носителе

ФИО **Иванов Иван Иванович**

Уполномоченное лицо Исполнителя подписи сторон на бумажном носителе

ФИО **Петров Петр Петрович**

Подписи сторон

Подписан Заказчиком

Подписан Исполнителем

Подписан Заказчиком

Подписан Исполнителем

Подписан Заказчиком

Подписан Исполнителем

Глава 10
ПАКЕТЫ

Библиотека — это арсенал свободы.
Источник неизвестен

Под понятием “пакет” подразумевается объединение взаимосвязанных классов, интерфейсов и подпакетов. Концепция пакета оказывается полезной по нескольким причинам:

- Пакеты позволяют группировать родственные интерфейсы и классы.
- В интерфейсах и классах, входящих в пакет, могут использоваться популярные имена (вроде get или put), которые имеют смысл в данном контексте, но конфликтуют с теми же именами в других пакетах.
- Пакет может включать типы и члены, с которыми можно работать только в пределах данного пакета. Соответствующие идентификаторы доступны для программ пакета, но закрыты для внешних методов.

Давайте рассмотрим пример пакета для нашего класса атрибутов, использованного в предыдущих главах. Назовем пакет attr. Каждый исходный файл, классы и интерфейсы которого принадлежат пакету attr, должен указывать на свою принадлежность к пакету объявлением package:

package attr;

Тем самым объявляется, что все классы и интерфейсы, определенные в этом исходном файле, являются частью пакета attr. Объявление package должно находиться в самом начале файла, до любых объявлений классов или интерфейсов; в файле может присутствовать всего одно объявление package. Имя пакета является неявным префиксом для всех имен типов, включенных в пакет.

Если фрагменту программы вне пакета понадобится обратиться к типам, входящим в пакет, у него имеется две возможности. Первая — указывать перед каждым именем типа префикс (имя пакета). Такой вариант будет вполне разумен, если вам приходится иметь дело всего с несколькими членами пакета.

Другая возможность доступа к типам пакета заключается в частичном или полном *импортировании* пакета. Программист, который захочет воспользоваться пакетом attr, может вставить следующую строку в начало своего исходного файла (после своего объявления package, но перед всем прочим):

import attr.*;

После этого он может обращаться к типам пакета просто по имени — например, Attributed. Пакет неявно импортирует сам себя, так что все, что определено в нем, становится доступным для всех остальных типов пакета.

Механизмы package и import помогают программисту предотвращать потенциальные конфликты имен. Если в пакете, предназначенном для других целей (скажем, лингвистических), тоже встретится класс с именем Attributed, предназначенный для хранения языковых атрибутов, то у программиста, пожелавшего использовать оба пакета в одном файле, имеется несколько вариантов:

- Обращаться к типам по их полным именам — например, attr.Attributed и lingua.Attributed.
- Импортировать attr.Attributed или attr.*, после чего использовать простое имя Attributed вместо attr.Attributed и полное имя lingua. Attributed.
- Сделать обратное — импортировать lingua.Attributed или lingua.*, после чего использовать простое имя Attributed вместо lingua. Attributed и полное имя attr.Attributed.

10.1. Имена пакетов

Имя пакета должно использоваться только один раз, так что выбор содержательного и уникального имени составляет важный аспект проектирования пакета. Однако сейчас, когда программисты всей планеты разрабатывают пакеты на языке Java, невозможно выяснить, какие имена пакетов ими используются. Следовательно, выбор уникального имени представляет некоторую проблему. Если вы уверены, что пакет будет использоваться только внутри вашей организации, то можно привлечь к делу выбора имени внутреннего арбитра — это позволит быть уверенным, что все пакеты будут иметь отличающиеся имена.

Тем не менее в нашем огромном мире такой подход не сработает. Идентификаторы пакетов Java представляют собой обычные имена, так что неплохой способ обеспечить их уникальность — включить в них имя домена организации в Internet. Если вы работаете в компании Magic, Inc., то пакет с атрибутами может быть объявлен следующим образом:

package COM.magic.attr;

Обратите внимание: компоненты имени следуют в обратном порядке по сравнению с обычным именем домена, а имя домена верхнего уровня (в нашем случае COM) написано прописными буквами. Это сделано для того, чтобы избежать конфликтов с именами пакетов, которые не следуют этому соглашению — их названия могут случайно совпасть с именем домена верхнего уровня, но вряд ли при этом они будут написаны прописными буквами.

Если вы следуете данному соглашению, конфликты возникать не будут (разве что внутри вашей организации). Если проблемы все же возникли (скажем, в очень большой организации), можно пойти дальше и уточнить имя домена. Во многих крупных компаниях имеются внутренние домены с именами east, europe и т. д. Вы можете уточнить имя пакета с использованием имени внутреннего домена:

package COM.magic.japan.attr;

При этом имена пакетов могут стать довольно длинными, однако такая схема относительно безопасна — никто из тех, кто ее использует, не выберет имя, совпадающее с названием вашего пакета.

Во многих средах разработки имена пакетов отражаются на уровне файловой системы — часто требуется, чтобы все программы из одного пакета находились в определенной папке или каталоге, а имя этого каталога соответствовало имени пакета. Подробности можно узнать в документации, относящейся к вашей среде разработки.

10.2. Пакетный доступ

Классы и интерфейсы, входящие в пакет, обладают одним из двух уровней доступа: пакетным (package) и открытым (public). Открытый класс или интерфейс доступен для программ, не входящих в пакет. Типы, не являющиеся открытыми, обладают областью видимости на уровне пакета: они доступны для всех программ того же пакета, но скрыты за его пределами, в том числе даже от программ во вложенных пакетах.

Члены классов тоже обладают уровнем доступа. Член, не объявленный с ключевым словом public, protected или private, может использоваться любой программой, входящей в пакет, но остается невидим за пределами пакета. Другими словами, по умолчанию идентификаторы обладают “пакетным” уровнем доступа, за исключением членов интерфейсов, которые являются открытыми.

Поля или методы, не объявленные в пакете с ключевым словом private, доступны для всех программ этого пакета. Следовательно, классы, входящие в тот же пакет, считаются “дружественными”, или “заслуживающими доверия”. Однако подпакеты не пользуются доверием в своих внешних пакетах. Например, защищенные и пакетные идентификаторы в пакете dit недоступны для программ в пакете dit.dat и наоборот.

10.3. Содержимое пакета

Если в исходном файле отсутствует объявление package, то входящие в него типы считаются принадлежащими к “безымянному” пакету.

К проектированию пакета следует подходить внимательно и включать в него только функционально связанные классы и интерфейсы. Классы пакета могут свободно обращаться к незакрытым членам друг друга. Защита членов класса предназначена для предотвращения некорректных действий со стороны классов, обладающих доступом к деталям внутренней реализации других классов. В пределах пакета не существует никаких ограничений доступа, кроме priv a te, и в итоге может получиться так, что посторонний класс получил к другим классам более близкий доступ, чем вам хотелось бы.

Кроме того, пакет должен составляться исходя из логического разделения задач, чтобы помочь программистам, которые ищут полезные для себя интерфейсы и классы. Если пакет состоит из несвязанных классов, будет сложно понять, что же из этого можно применить в работе. Логическое разделение способствует повторному использованию вашего кода, поскольку программистам будет легче ориентироваться в нем.

Если пакет состоит из взаимосвязанных типов, вы сможете давать им очевидные имена, избегая при этом конфликтов с посторонними типами из этого же пакета.

Пакет может быть составной частью другого пакета. Например, пакет java.lang является вложенным, то есть пакет lang входит в более обширный пакет java. Пакет java не содержит ничего, кроме других пакетов. Вложение позволяет построить иерархическую систему имен для взаимосвязанных пакетов.

Например, чтобы создать набор пакетов для адаптивных систем (скажем, нейросетей или генетических алгоритмов), можно воспользоваться вложенными пакетами, разделяя их имена точками:

package adaptive.neuralNet;

Исходный файл с таким объявлением входит в пакет adaptive.neuralNet, который, в свою очередь, является подпакетом пакета adaptive. Пакет adaptive может содержать классы, относящиеся к общим адаптивным алгоритмам, — например, методы с общей постановкой генетических проблем или способы измерения каких-то показателей. Каждый пакет, который находится ниже в иерархии (например, adaptive.neuralNet или adaptive. genetic), содержит классы, предназначенные для конкретного типа адаптивных алгоритмов.

Вложение пакетов является средством организации взаимосвязанных пакетов и не имеет отношения к правам доступа. Классы пакета adaptive. genetic не смогут работать с закрытыми на пакетном уровне идентификаторами из adaptive или adaptive.neuralNet. Область видимости пакета не выходит за его пределы. Вложение позволяет группировать взаимосвязанные пакеты и помогает программистам искать классы в иерархической структуре, но не дает никаких других преимуществ.

Глава 11
ПАКЕТ ВВОДА/ВЫВОДА

Автор: Стивен Ломбардо
Перевод: Сергей Кулик
Версия: 1.0

11.1. Введение

Пакет ввода/вывода (I/O) предоставляет набор классов, которые используются для ввода/вывода потоков данных. Поскольку в Java потоки ввода/вывода могут использоваться для различных целей, пакет ввода/вывода предоставляет большое количество классов. Эти классы могут быть объединены для получения нужного вам поведения. Пакет ввода/вывода (I/O) содержит более 60 классов, которые могут быть использованы для ввода/вывода потоков данных. Эти классы могут быть объединены для получения нужного вам поведения. Пакет ввода/вывода (I/O) также предоставляет 10 интерфейсов, которые используются классами ввода/вывода (I/O).

Поскольку Java обеспечивает кроссплатформенную независимость, ввод/вывод данных является сложным заданием. Поэтому Java по-прежнему предоставляет поддержку ввода/вывода данных, специфичных для платформы, если вы хотите использовать такую поддержку в своем коде. Однако, в большинстве случаев, вам не нужно беспокоиться о том, какая платформа лежит в основе ввода/вывода данных.

Пакет ввода/вывода (I/O) предоставляет поддержку для нескольких различных потоков ввода/вывода данных. Основные из них включают:

- Файловый ввод/вывод данных
- Ввод/вывод данных в буфер памяти
- Ввод/вывод данных в канал между потоками
- Ввод/вывод данных в поток объектов
- Ввод/вывод данных в поток примитивных типов данных
- Ввод/вывод данных в поток сериализованных объектов
- Ввод/вывод данных в поток строк
- Ввод/вывод данных в сетевой поток данных

Пакет предоставляет два основных класса: byte-ориентированный поток данных и char-ориентированный поток данных.

11.2. Потоки

Пакет ввода/вывода (I/O) предоставляет большое количество классов, которые могут быть использованы для ввода/вывода потоков данных. Однако, все эти классы могут быть сведены к пяти основным классам:

- InputStream
- OutputStream
- Reader
- Writer
- File

Каждый из этих классов является абстрактным классом.

Каждый из потоков ввода/вывода данных может быть использован для ввода/вывода данных.

Каждый из потоков ввода/вывода данных может быть использован для ввода/вывода данных.

Каждый из потоков ввода/вывода данных может быть использован для ввода/вывода данных.

Каждый из потоков ввода/вывода данных может быть использован для ввода/вывода данных.

Каждый из потоков ввода/вывода данных может быть использован для ввода/вывода данных.

11.2.1. Класс InputStream

Класс InputStream является абстрактным классом, который предоставляет поддержку ввода/вывода потоков данных.

public InputStream()

Конструктор InputStream. Создает новый экземпляр InputStream.

public int read() throws IOException

Читает следующий байт данных из потока ввода/вывода. Возвращает значение в диапазоне от 0 до 255 или -1 если достигнут конец потока ввода/вывода данных. Генерирует ошибку IOException если произошла ошибка ввода/вывода потока данных.

public int read(byte[] b) throws IOException

Читает несколько байтов данных из потока ввода/вывода. Возвращает количество прочитанных байтов или -1 если достигнут конец потока ввода/вывода данных. Генерирует ошибку IOException если произошла ошибка ввода/вывода потока данных.

public int read(byte[] b, int off, int len) throws IOException

Читает несколько байтов данных из потока ввода/вывода. Возвращает количество прочитанных байтов или -1 если достигнут конец потока ввода/вывода данных. Генерирует ошибку IOException если произошла ошибка ввода/вывода потока данных.

public long skip(long n) throws IOException

Пропускает n-байтов данных из потока ввода/вывода. Возвращает количество пропущенных байтов. Генерирует ошибку IOException если произошла ошибка ввода/вывода потока данных.

public int available() throws IOException

Возвращает количество байтов, которое может быть прочитано (или пропущено) из потока ввода/вывода данных. Генерирует ошибку IOException если произошла ошибка ввода/вывода потока данных.

public void close() throws IOException

Закрывает поток ввода/вывода данных. Генерирует ошибку IOException если произошла ошибка ввода/вывода потока данных.

public void mark(int readlimit)

Метка. Вставляет метку в поток ввода/вывода данных.

public void reset() throws IOException

Сброс. Сбрасывает поток ввода/вывода данных.

public boolean markSupported()

Проверяет, есть ли поддержка метки для потока ввода/вывода данных.

Например, класс, расширяющий класс InputStream, может использовать следующий код для чтения потока ввода/вывода данных:

```
import java.io.*;

public class MyInputStream
{
    protected int[] buf = null;
    protected int count = 0;
    protected int pos = 0;

    public int read() throws IOException
    {
        int1 in1 = null;
        int1 ret1 = null;

        if (buf == null)
        {
            throw new IOException("Stream closed");
        }
        if (pos >= count)
        {
            return -1;
        }
        in1 = (int1)buf[pos];
        ret1 = in1;
        pos = pos + 1;
        return ret1.intValue();
    }
}
```

Приведенный выше пример создает класс, который наследует функциональность класса InputStream. Каждый раз при вызове метода read() данный класс будет возвращать следующий байт данных из потока ввода/вывода данных. Если достигнут конец потока ввода/вывода данных, будет возвращено значение -1.

Класс InputStream обеспечивает поддержку метки. Если вы хотите использовать функциональность данной поддержки, вы должны переопределить следующие методы:

public void mark(int readlimit)

Метка. Вставляет метку в поток ввода/вывода данных.

public void reset() throws IOException

Сброс. Сбрасывает поток ввода/вывода данных.

public boolean markSupported()

Проверяет, есть ли поддержка метки для потока ввода/вывода данных.

Например, класс, расширяющий класс InputStream, может использовать следующий код для чтения потока ввода/вывода данных:

```
import java.io.*;

public class MyInputStream
{
    protected int[] buf = null;
    protected int count = 0;
    protected int pos = 0;
    protected int markpos = -1;
    protected int marklimit = 0;

    public int read() throws IOException
    {
        int1 in1 = null;
        int1 ret1 = null;

        if (buf == null)
        {
            throw new IOException("Stream closed");
        }
        if (pos >= count)
        {
            return -1;
        }
        in1 = (int1)buf[pos];
        ret1 = in1;
        pos = pos + 1;
        return ret1.intValue();
    }

    public void mark(int readlimit)
    {
        if (buf == null)
        {
            throw new IOException("Stream closed");
        }
        marklimit = readlimit;
        markpos = pos;
    }

    public void reset() throws IOException
    {
        if (buf == null)
        {
            throw new IOException("Stream closed");
        }
        if (markpos < 0)
        {
            throw new IOException("Mark invalid");
        }
        pos = markpos;
    }

    public boolean markSupported()
    {
        return true;
    }
}
```

11.2.2. Класс OutputStream

Класс OutputStream является абстрактным классом, который предоставляет поддержку ввода/вывода потоков данных.

public OutputStream()

Конструктор OutputStream. Создает новый экземпляр OutputStream.

public void write(int b) throws IOException

Пишет следующий байт данных в поток ввода/вывода данных. Генерирует ошибку IOException если произошла ошибка ввода/вывода потока данных.

public void write(byte[] b) throws IOException

Пишет несколько байтов данных в поток ввода/вывода данных. Генерирует ошибку IOException если произошла ошибка ввода/вывода потока данных.

public void write(byte[] b, int off, int len) throws IOException

Пишет несколько байтов данных в поток ввода/вывода данных. Генерирует ошибку IOException если произошла ошибка ввода/вывода потока данных.

public void flush() throws IOException

Очищает поток ввода/вывода данных. Генерирует ошибку IOException если произошла ошибка ввода/вывода потока данных.

public void close() throws IOException

Закрывает поток ввода/вывода данных. Генерирует ошибку IOException если произошла ошибка ввода/вывода потока данных.

Например, класс, расширяющий класс OutputStream, может использовать следующий код для записи в поток ввода/вывода данных:

```
import java.io.*;

public class MyOutputStream
{
    protected int[] buf = null;
    protected int count = 0;
    protected int pos = 0;

    public void write(int b) throws IOException
    {
        int1 in1 = null;

        if (buf == null)
        {
            throw new IOException("Stream closed");
        }
        if (pos >= buf.length)
        {
            int[] newbuf = null;
            int i = 0;
            int1 tmp = null;

            newbuf = new int[buf.length + 1];
            for (i = 0; i < buf.length; i++)
            {
                tmp = (int1)buf[i];
                newbuf[i] = tmp.intValue();
            }
            buf = newbuf;
        }
        in1 = new int1(b);
        buf[pos] = in1.intValue();
        pos = pos + 1;
        count = count + 1;
    }
}
```

11.2.3. Класс Reader

Класс Reader является абстрактным классом, который предоставляет поддержку ввода/вывода потоков данных.

public Reader()

Конструктор Reader. Создает новый экземпляр Reader.

public int read() throws IOException

Читает следующий символ данных из потока ввода/вывода. Возвращает значение в диапазоне от 0 до 65535 или -1 если достигнут конец потока ввода/вывода данных. Генерирует ошибку IOException если произошла ошибка ввода/вывода потока данных.

public int read(char[] c) throws IOException

Читает несколько символов данных из потока ввода/вывода. Возвращает количество прочитанных символов или -1 если достигнут конец потока ввода/вывода данных. Генерирует ошибку IOException если произошла ошибка ввода/вывода потока данных.

public int read(char[] c, int off, int len) throws IOException

Читает несколько символов данных из потока ввода/вывода. Возвращает количество прочитанных символов или -1 если достигнут конец потока ввода/вывода данных. Генерирует ошибку IOException если произошла ошибка ввода/вывода потока данных.

public long skip(long n) throws IOException

Пропускает n-символов данных из потока ввода/вывода. Возвращает количество пропущенных символов. Генерирует ошибку IOException если произошла ошибка ввода/вывода потока данных.

public int available() throws IOException

Возвращает количество символов, которое может быть прочитано (или пропущено) из потока ввода/вывода данных. Генерирует ошибку IOException если произошла ошибка ввода/вывода потока данных.

public void close() throws IOException

Закрывает поток ввода/вывода данных. Генерирует ошибку IOException если произошла ошибка ввода/вывода потока данных.

public void mark(int readlimit)

Метка. Вставляет метку в поток ввода/вывода данных.

public void reset() throws IOException

Сброс. Сбрасывает поток ввода/вывода данных.

public boolean markSupported()

Проверяет, есть ли поддержка метки для потока ввода/вывода данных.

Например, класс, расширяющий класс Reader, может использовать следующий код для чтения потока ввода/вывода данных:

```
import java.io.*;

public class MyReader
{
    protected int[] buf = null;
    protected int count = 0;
    protected int pos = 0;

    public int read() throws IOException
    {
        int1 in1 = null;
        int1 ret1 = null;

        if (buf == null)
        {
            throw new IOException("Stream closed");
        }
        if (pos >= count)
        {
            return -1;
        }
        in1 = (int1)buf[pos];
        ret1 = in1;
        pos = pos + 1;
        return ret1.intValue();
    }
}
```

Приведенный выше пример создает класс, который наследует функциональность класса Reader. Каждый раз при вызове метода read() данный класс будет возвращать следующий символ данных из потока ввода/вывода данных. Если достигнут конец потока ввода/вывода данных, будет возвращено значение -1.

Класс Reader обеспечивает поддержку метки. Если вы хотите использовать функциональность данной поддержки, вы должны переопределить следующие методы:

public void mark(int readlimit)

Метка. Вставляет метку в поток ввода/вывода данных.

public void reset() throws IOException

Сброс. Сбрасывает поток ввода/вывода данных.

public boolean markSupported()

Проверяет, есть ли поддержка метки для потока ввода/вывода данных.

Например, класс, расширяющий класс Reader, может использовать следующий код для чтения потока ввода/вывода данных:

```
import java.io.*;

public class MyReader
{
    protected int[] buf = null;
    protected int count = 0;
    protected int pos = 0;
    protected int markpos = -1;
    protected int marklimit = 0;

    public int read() throws IOException
    {
        int1 in1 = null;
        int1 ret1 = null;

        if (buf == null)
        {
            throw new IOException("Stream closed");
        }
        if (pos >= count)
        {
            return -1;
        }
        in1 = (int1)buf[pos];
        ret1 = in1;
        pos = pos + 1;
        return ret1.intValue();
    }

    public void mark(int readlimit)
    {
        if (buf == null)
        {
            throw new IOException("Stream closed");
        }
        marklimit = readlimit;
        markpos = pos;
    }

    public void reset() throws IOException
    {
        if (buf == null)
        {
            throw new IOException("Stream closed");
        }
        if (markpos < 0)
        {
            throw new IOException("Mark invalid");
        }
        pos = markpos;
    }

    public boolean markSupported()
    {
        return true;
    }
}
```

11.2.4. Класс Writer

Класс Writer является абстрактным классом, который предоставляет поддержку ввода/вывода потоков данных.

public Writer()

Конструктор Writer. Создает новый экземпляр Writer.

public void write(int c) throws IOException

Пишет следующий символ данных в поток ввода/вывода данных. Генерирует ошибку IOException если произошла ошибка ввода/вывода потока данных.

public void write(char[] c) throws IOException

Пишет несколько символов данных в поток ввода/вывода данных. Генерирует ошибку IOException если произошла ошибка ввода/вывода потока данных.

public void write(char[] c, int off, int len) throws IOException

Пишет несколько символов данных в поток ввода/вывода данных. Генерирует ошибку IOException если произошла ошибка ввода/вывода потока данных.

public void flush() throws IOException

Очищает поток ввода/вывода данных. Генерирует ошибку IOException если произошла ошибка ввода/вывода потока данных.

public void close() throws IOException

Закрывает поток ввода/вывода данных. Генерирует ошибку IOException если произошла ошибка ввода/вывода потока данных.

Например, класс, расширяющий класс Writer, может использовать следующий код для записи в поток ввода/вывода данных:

```
import java.io.*;

public class MyWriter
{
    protected int[] buf = null;
    protected int count = 0;
    protected int pos = 0;

    public void write(int c) throws IOException
    {
        int1 in1 = null;

        if (buf == null)
        {
            throw new IOException("Stream closed");
        }
        if (pos >= buf.length)
        {
            int[] newbuf = null;
            int i = 0;
            int1 tmp = null;

            newbuf = new int[buf.length + 1];
            for (i = 0; i < buf.length; i++)
            {
                tmp = (int1)buf[i];
                newbuf[i] = tmp.intValue();
            }
            buf = newbuf;
        }
        in1 = new int1(c);
        buf[pos] = in1.intValue();
        pos = pos + 1;
        count = count + 1;
    }
}
```

11.2.5. Класс File

Класс File предоставляет поддержку для ввода/вывода данных из файлов.

public File(String pathname)

Конструктор File. Создает новый экземпляр File.

public File(File parent, String child)

Конструктор File. Создает новый экземпляр File.

public File(String parent, String child)

Конструктор File. Создает новый экземпляр File.

public File(URI uri)

Конструктор File. Создает новый экземпляр File.

public boolean canRead()

Проверяет, есть ли возможность чтения из файла.

public boolean canWrite()

Проверяет, есть ли возможность записи в файл.

public boolean exists()

Проверяет, существует ли файл.

public boolean isAbsolute()

Проверяет, является ли путь к файлу абсолютным.

public boolean isDirectory()

Проверяет, является ли файл директорией.

public boolean isFile()

Проверяет, является ли файл файлом.

public boolean isHidden()

Проверяет, является ли файл скрытым.

public long lastModified()

Возвращает последнее время модификации файла.

public long length()

Возвращает размер файла.

public boolean mkdir()

Создает директорию.

public boolean mkdirs()

Создает директории.

public boolean renameTo(File dest)

Переименовывает файл.

public boolean setLastModified(long time)

Устанавливает последнее время модификации файла.

public boolean setReadOnly()

Устанавливает файл как файл только для чтения.

public boolean delete()

Удаляет файл.

public String[] list()

Возвращает список файлов в директории.

public String[] list(FilenameFilter filter)

Возвращает список файлов в директории.

public File[] listFiles()

Возвращает список файлов в директории.

public File[] listFiles(FilenameFilter filter)

Возвращает список файлов в директории.

public File[] listFiles(FileFilter filter)

Возвращает список файлов в директории.

public String getAbsolutePath()

Возвращает абсолютный путь к файлу.

public File getAbsoluteFile()

Возвращает абсолютный путь к файлу.

public String getPath()

Возвращает путь к файлу.

public File getFile()

Возвращает путь к файлу.

public String getName()

Возвращает имя файла.

public String getParent()

Возвращает имя директории.

public File getParentFile()

Возвращает имя директории.

public URI toURI()

Возвращает URI файла.

public URL toURL() throws MalformedURLException

Возвращает URL файла.

public static File[] listRoots()

Возвращает корневые директории.

public static File[] listRoots(FileSystemRoots[] fsr)

Возвращает корневые директории.

public static File getRoot(int i)

Возвращает корневую директорию.

public static File getRoot(int i, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19, int[] ir20)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19, int[] ir20, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19, int[] ir20, int[] ir21)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19, int[] ir20, int[] ir21, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19, int[] ir20, int[] ir21, int[] ir22)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19, int[] ir20, int[] ir21, int[] ir22, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19, int[] ir20, int[] ir21, int[] ir22, int[] ir23)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19, int[] ir20, int[] ir21, int[] ir22, int[] ir23, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19, int[] ir20, int[] ir21, int[] ir22, int[] ir23, int[] ir24)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19, int[] ir20, int[] ir21, int[] ir22, int[] ir23, int[] ir24, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19, int[] ir20, int[] ir21, int[] ir22, int[] ir23, int[] ir24, int[] ir25)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19, int[] ir20, int[] ir21, int[] ir22, int[] ir23, int[] ir24, int[] ir25, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19, int[] ir20, int[] ir21, int[] ir22, int[] ir23, int[] ir24, int[] ir25, int[] ir26)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19, int[] ir20, int[] ir21, int[] ir22, int[] ir23, int[] ir24, int[] ir25, int[] ir26, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19, int[] ir20, int[] ir21, int[] ir22, int[] ir23, int[] ir24, int[] ir25, int[] ir26, int[] ir27)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19, int[] ir20, int[] ir21, int[] ir22, int[] ir23, int[] ir24, int[] ir25, int[] ir26, int[] ir27, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19, int[] ir20, int[] ir21, int[] ir22, int[] ir23, int[] ir24, int[] ir25, int[] ir26, int[] ir27, int[] ir28)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19, int[] ir20, int[] ir21, int[] ir22, int[] ir23, int[] ir24, int[] ir25, int[] ir26, int[] ir27, int[] ir28, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19, int[] ir20, int[] ir21, int[] ir22, int[] ir23, int[] ir24, int[] ir25, int[] ir26, int[] ir27, int[] ir28, int[] ir29)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19, int[] ir20, int[] ir21, int[] ir22, int[] ir23, int[] ir24, int[] ir25, int[] ir26, int[] ir27, int[] ir28, int[] ir29, FileSystemRoots[] fsr)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19, int[] ir20, int[] ir21, int[] ir22, int[] ir23, int[] ir24, int[] ir25, int[] ir26, int[] ir27, int[] ir28, int[] ir29, int[] ir30)

Возвращает корневую директорию.

public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[] ir4, int[] ir5, int[] ir6, int[] ir7, int[] ir8, int[] ir9, int[] ir10, int[] ir11, int[] ir12, int[] ir13, int[] ir14, int[] ir15, int[] ir16, int[] ir17, int[] ir18, int[] ir19, int[] ir20, int[] ir21, int[] ir22, int[] ir23, int[] ir24, int[] ir25, int[] ir26, int[] ir27, int[] ir28, int[] ir29, int[] ir30, FileSystemRoots[] fsr)

Возвращает корневую директорию.

**public static File getRoot(int i, int[] ir, int[] ir2, int[] ir3, int[]

user_name: **Ivanchukchenko**

Ивановская область, Ивановский район, Ивановская ул, 100 кв. 100

Каждый раз, когда у меня возникает ощущение, что я нашел что-то уникальное, выясняется, что кто-то уже сделал это задолго до меня. Сначала я расстроился, потом подумал, что это, наверное, к лучшему — значит, я не ошибся, раз другие тоже пришли к этому выводу. Теперь я ищу то, что пока никто не сделал, и, скорее всего, найду. Если вы ищете что-то уникальное — ищите то, что пока никто не сделал. Вероятность того, что вы ошибетесь, ничтожно мала. Если вы ошибетесь, вы узнаете, что это уже сделали другие, и сможете сказать себе, что вы не ошибались. Если вы ошибетесь, вы узнаете, что вы ошибались, и сможете сказать себе, что вы

Глава 12
СТАНДАРТНЫЕ ВСЛОЮЕ КТЕЛЬНЫЕ СРЕДСТВА

Стандартные вложенные классы
Стандартные вложенные классы
Стандартные вложенные классы

Рассмотрим следующие:

- Стандартные вложенные классы для исключений
- Стандартные вложенные классы для потоков ввода-вывода и для представления набора объектов-значений, которые, в отличие от ранее рассмотренных, не являются исключениями.
- Набор классов, определяющих свойства объектов, которые могут использоваться для сортировки, фильтрации и поиска объектов, а также для ограничения доступа к ним. Эти классы называются классами свойств и являются вложенными классами стандартного класса System.Attribute.

Все классы, которые мы в этом разделе будем рассматривать, наследуют от стандартного класса System.Object.

Рассмотрим следующие:

- Стандартные классы для представления даты и времени, а также для работы с часовыми поясами и календарями.
- Набор классов, определяющих свойства объектов, которые могут использоваться для сериализации и десериализации объектов.

Часть 1:

- Стандартные классы, поддерживающие коллекции.
- Стандартные классы, определяющие свойства объектов, которые могут использоваться для сериализации и десериализации объектов.

12.1. Класс System.Exception

Класс System.Exception является базовым классом для всех классов, представляющих исключения. Все классы, которые наследуют от класса System.Exception, являются классами исключений. Класс System.Exception определяет свойства, которые могут использоваться для получения информации об исключении, а также для ограничения доступа к ним. Эти свойства являются общедоступными, виртуальными и доступными для чтения. Класс System.Exception также определяет несколько конструкторов, которые могут использоваться для инициализации объектов исключений.

Наиболее часто используемые свойства класса System.Exception перечислены ниже. Все они являются общедоступными, виртуальными и доступными для чтения.

- **Message** – возвращает строку, которая описывает причину исключения.
- **StackTrace** – возвращает строку, которая описывает последовательность вызовов, которая привела к возникновению исключения.
- **InnerException** – возвращает ссылку на объект исключения, которое является причиной исключения, в случае, если такое имеется.

Класс System.Exception определяет несколько конструкторов. Наиболее часто используемые из них перечислены ниже:

- **Exception()** – создает объект исключения.
- **Exception(string)** – создает объект исключения, в свойстве Message которого будет сохранена строка, переданная в качестве параметра.
- **Exception(string, Exception)** – создает объект исключения, в свойстве Message которого будет сохранена строка, переданная в качестве первого параметра, а в свойстве InnerException – ссылка на объект исключения, переданный в качестве второго параметра.

12.2. Потоки ввода-вывода

Пространство имен System.IO предназначено для поддержки ввода-вывода данных. Оно определяет классы потоков, которые могут использоваться для чтения и записи данных в различные источники данных, такие как файлы, сетевые соединения и области памяти. Пространство имен System.IO также определяет классы, которые могут использоваться для получения информации о файлах и каталогах, а также для ограничения доступа к ним.

Рассмотрим следующий пример:

```
using System;
class Program
{
    static void Main()
    {
        for (int i = 1; i < 5; i++)
            for (int j = 1; j < 5; j++)
            {
                Console.Write("Enter number");
                int n = int.Parse(Console.ReadLine());
                if (n == 0)
                    goto EXIT;
                if (n % 2 == 0)
                    Console.WriteLine("{0} - Even", n);
                else
                    Console.WriteLine("{0} - Odd", n);
            }
        EXIT:
        Console.WriteLine("Bye");
    }
}
```

12.3. Потоки Stream

Пространство имен System.IO определяет несколько классов потоков. Все классы потоков являются производными от абстрактного класса System.IO.Stream. Класс System.IO.Stream определяет несколько общедоступных, виртуальных и доступных для чтения свойств, которые могут использоваться для получения информации о потоке, а также для ограничения доступа к ним. Класс System.IO.Stream также определяет несколько общедоступных, виртуальных и доступных для чтения методов, которые могут использоваться для чтения данных из потока и записи данных в поток.

Наиболее часто используемые свойства класса System.IO.Stream перечислены ниже. Все они являются общедоступными, виртуальными и доступными для чтения.

- **CanRead** – возвращает true, если поддерживается чтение из потока.
- **CanWrite** – возвращает true, если поддерживается запись в поток.
- **Length** – возвращает длину потока в байтах.
- **Position** – возвращает позицию, с которой будет выполняться следующая операция чтения или записи.

Наиболее часто используемые методы класса System.IO.Stream перечислены ниже. Все они являются общедоступными, виртуальными и доступными для чтения.

- **Close()** – закрывает поток и освобождает ресурсы, которые были выделены для потока.
- **Flush()** – сохраняет в буфере данные, которые были записаны в поток, но еще не сохранены в буфере.
- **Read(byte[], int, int)** – читает из потока.
- **Write(byte[], int, int)** – пишет в поток.

Рассмотрим следующий пример:

```
using System;
class Program
{
    static void Main(int[]
    {
        int[] a = new int[5];
        int[] b = new int[5];
        int[] c = new int[5];
        int[] d = new int[5];
        int[] e = new int[5];
        int[] f = new int[5];
        int[] g = new int[5];
        int[] h = new int[5];
        int[] i = new int[5];
        int[] j = new int[5];
        int[] k = new int[5];
        int[] l = new int[5];
        int[] m = new int[5];
        int[] n = new int[5];
        int[] o = new int[5];
        int[] p = new int[5];
        int[] q = new int[5];
        int[] r = new int[5];
        int[] s = new int[5];
        int[] t = new int[5];
        int[] u = new int[5];
        int[] v = new int[5];
        int[] w = new int[5];
        int[] x = new int[5];
        int[] y = new int[5];
        int[] z = new int[5];
        int[] aa = new int[5];
        int[] bb = new int[5];
        int[] cc = new int[5];
        int[] dd = new int[5];
        int[] ee = new int[5];
        int[] ff = new int[5];
        int[] gg = new int[5];
        int[] hh = new int[5];
        int[] ii = new int[5];
        int[] jj = new int[5];
        int[] kk = new int[5];
        int[] ll = new int[5];
        int[] mm = new int[5];
        int[] nn = new int[5];
        int[] oo = new int[5];
        int[] pp = new int[5];
        int[] qq = new int[5];
        int[] rr = new int[5];
        int[] ss = new int[5];
        int[] tt = new int[5];
        int[] uu = new int[5];
        int[] vv = new int[5];
        int[] ww = new int[5];
        int[] xx = new int[5];
        int[] yy = new int[5];
        int[] zz = new int[5];
        int[] aaa = new int[5];
        int[] bbb = new int[5];
        int[] ccc = new int[5];
        int[] ddd = new int[5];
        int[] eee = new int[5];
        int[] fff = new int[5];
        int[] ggg = new int[5];
        int[] hhh = new int[5];
        int[] iii = new int[5];
        int[] jjj = new int[5];
        int[] kkk = new int[5];
        int[] lll = new int[5];
        int[] mmm = new int[5];
        int[] nnn = new int[5];
        int[] ooo = new int[5];
        int[] ppp = new int[5];
        int[] qqq = new int[5];
        int[] rrr = new int[5];
        int[] sss = new int[5];
        int[] ttt = new int[5];
        int[] uuu = new int[5];
        int[] vvv = new int[5];
        int[] www = new int[5];
        int[] xxx = new int[5];
        int[] yyy = new int[5];
        int[] zzz = new int[5];
        int[] aaaa = new int[5];
        int[] bbbb = new int[5];
        int[] cccc = new int[5];
        int[] dddd = new int[5];
        int[] eeee = new int[5];
        int[] ffff = new int[5];
        int[] gggg = new int[5];
        int[] hhhh = new int[5];
        int[] iiii = new int[5];
        int[] jjjj = new int[5];
        int[] kkkk = new int[5];
        int[] llll = new int[5];
        int[] mmmm = new int[5];
        int[] nnnn = new int[5];
        int[] oooo = new int[5];
        int[] pppp = new int[5];
        int[] qqqq = new int[5];
        int[] rrrr = new int[5];
        int[] ssss = new int[5];
        int[] tttt = new int[5];
        int[] uuuu = new int[5];
        int[] vvvv = new int[5];
        int[] wwww = new int[5];
        int[] xxxx = new int[5];
        int[] yyyy = new int[5];
        int[] zzzz = new int[5];
        int[] aaaaa = new int[5];
        int[] bbbbb = new int[5];
        int[] ccccc = new int[5];
        int[] ddddd = new int[5];
        int[] eeeee = new int[5];
        int[] fffff = new int[5];
        int[] ggggg = new int[5];
        int[] hhhhh = new int[5];
        int[] iiiii = new int[5];
        int[] jjjjj = new int[5];
        int[] kkkkk = new int[5];
        int[] lllll = new int[5];
        int[] mmmmm = new int[5];
        int[] nnnnn = new int[5];
        int[] ooooo = new int[5];
        int[] ppppp = new int[5];
        int[] qqqqq = new int[5];
        int[] rrrrr = new int[5];
        int[] sssss = new int[5];
        int[] ttttt = new int[5];
        int[] uuuuu = new int[5];
        int[] vvvvv = new int[5];
        int[] wwwww = new int[5];
        int[] xxxxx = new int[5];
        int[] yyyyy = new int[5];
        int[] zzzzz = new int[5];
        int[] aaaaaa = new int[5];
        int[] bbbbbb = new int[5];
        int[] cccccc = new int[5];
        int[] dddddd = new int[5];
        int[] eeeeee = new int[5];
        int[] ffffff = new int[5];
        int[] gggggg = new int[5];
        int[] hhhhhh = new int[5];
        int[] iiiiii = new int[5];
        int[] jjjjjj = new int[5];
        int[] kkkkkk = new int[5];
        int[] llllll = new int[5];
        int[] mmmmmm = new int[5];
        int[] nnnnnn = new int[5];
        int[] oooooo = new int[5];
        int[] pppppp = new int[5];
        int[] qqqqqq = new int[5];
        int[] rrrrrr = new int[5];
        int[] ssssss = new int[5];
        int[] tttttt = new int[5];
        int[] uuuuuu = new int[5];
        int[] vvvvvv = new int[5];
        int[] wwwwww = new int[5];
        int[] xxxxxx = new int[5];
        int[] yyyyyy = new int[5];
        int[] zzzzzz = new int[5];
        int[] aaaaaaa = new int[5];
        int[] bbbbbbb = new int[5];
        int[] ccccccc = new int[5];
        int[] ddddddd = new int[5];
        int[] eeeeeee = new int[5];
        int[] fffffff = new int[5];
        int[] ggggggg = new int[5];
        int[] hhhhhhh = new int[5];
        int[] iiiiiii = new int[5];
        int[] jjjjjjj = new int[5];
        int[] kkkkkkk = new int[5];
        int[] lllllll = new int[5];
        int[] mmmmmmm = new int[5];
        int[] nnnnnnn = new int[5];
        int[] ooooooo = new int[5];
        int[] ppppppp = new int[5];
        int[] qqqqqqq = new int[5];
        int[] rrrrrrr = new int[5];
        int[] sssssss = new int[5];
        int[] ttttttt = new int[5];
        int[] uuuuuuu = new int[5];
        int[] vvvvvvv = new int[5];
        int[] wwwwwww = new int[5];
        int[] xxxxxxx = new int[5];
        int[] yyyyyyy = new int[5];
        int[] zzzzzzz = new int[5];
        int[] aaaaaaaa = new int[5];
        int[] bbbbbbbb = new int[5];
        int[] cccccccc = new int[5];
        int[] dddddddd = new int[5];
        int[] eeeeeeee = new int[5];
        int[] ffffffff = new int[5];
        int[] gggggggg = new int[5];
        int[] hhhhhhhh = new int[5];
        int[] iiiiiiii = new int[5];
        int[] jjjjjjjj = new int[5];
        int[] kkkkkkkk = new int[5];
        int[] llllllll = new int[5];
        int[] mmmmmmmm = new int[5];
        int[] nnnnnnnn = new int[5];
        int[] oooooooo = new int[5];
        int[] pppppppp = new int[5];
        int[] qqqqqqqq = new int[5];
        int[] rrrrrrrr = new int[5];
        int[] ssssssss = new int[5];
        int[] tttttttt = new int[5];
        int[] uuuuuuuu = new int[5];
        int[] vvvvvvvv = new int[5];
        int[] wwwwwwww = new int[5];
        int[] xxxxxxxx = new int[5];
        int[] yyyyyyyy = new int[5];
        int[] zzzzzzzz = new int[5];
        int[] aaaaaaaaa = new int[5];
        int[] bbbbbbbbb = new int[5];
        int[] ccccccccc = new int[5];
        int[] ddddddddd = new int[5];
        int[] eeeeeeeee = new int[5];
        int[] fffffffff = new int[5];
        int[] ggggggggg = new int[5];
        int[] hhhhhhhhh = new int[5];
        int[] iiiiiiiii = new int[5];
        int[] jjjjjjjjj = new int[5];
        int[] kkkkkkkkk = new int[5];
        int[] lllllllll = new int[5];
        int[] mmmmmmmmm = new int[5];
        int[] nnnnnnnnn = new int[5];
        int[] ooooooooo = new int[5];
        int[] ppppppppp = new int[5];
        int[] qqqqqqqqq = new int[5];
        int[] rrrrrrrrr = new int[5];
        int[] sssssssss = new int[5];
        int[] ttttttttt = new int[5];
        int[] uuuuuuuuu = new int[5];
        int[] vvvvvvvvv = new int[5];
        int[] wwwwwwwww = new int[5];
        int[] xxxxxxxxx = new int[5];
        int[] yyyyyyyyy = new int[5];
        int[] zzzzzzzzz = new int[5];
        int[] aaaaaaaaaa = new int[5];
        int[] bbbbbbbbbb = new int[5];
        int[] cccccccccc = new int[5];
        int[] dddddddddd = new int[5];
        int[] eeeeeeeeee = new int[5];
        int[] ffffffffff = new int[5];
        int[] gggggggggg = new int[5];
        int[] hhhhhhhhhh = new int[5];
        int[] iiiiiiiiii = new int[5];
        int[] jjjjjjjjjj = new int[5];
        int[] kkkkkkkkkk = new int[5];
        int[] llllllllll = new int[5];
        int[] mmmmmmmmmm = new int[5];
        int[] nnnnnnnnnn = new int[5];
        int[] oooooooooo = new int[5];
        int[] pppppppppp = new int[5];
        int[] qqqqqqqqqq = new int[5];
        int[] rrrrrrrrrr = new int[5];
        int[] ssssssssss = new int[5];
        int[] tttttttttt = new int[5];
        int[] uuuuuuuuuu = new int[5];
        int[] vvvvvvvvvv = new int[5];
        int[] wwwwwwwwww = new int[5];
        int[] xxxxxxxxxx = new int[5];
        int[] yyyyyyyyyy = new int[5];
        int[] zzzzzzzzzz = new int[5];
        int[] aaaaaaaaaaa = new int[5];
        int[] bbbbbbbbbbb = new int[5];
        int[] ccccccccccc = new int[5];
        int[] dddddddddddd = new int[5];
        int[] eeeeeeeeeee = new int[5];
        int[] fffffffffff = new int[5];
        int[] ggggggggggg = new int[5];
        int[] hhhhhhhhhhh = new int[5];
        int[] iiiiiiiiiii = new int[5];
        int[] jjjjjjjjjjj = new int[5];
        int[] kkkkkkkkkkk = new int[5];
        int[] lllllllllll = new int[5];
        int[] mmmmmmmmmmm = new int[5];
        int[] nnnnnnnnnnn = new int[5];
        int[] ooooooooooo = new int[5];
        int[] ppppppppppp = new int[5];
        int[] qqqqqqqqqqq = new int[5];
        int[] rrrrrrrrrrr = new int[5];
        int[] sssssssssss = new int[5];
        int[] ttttttttttt = new int[5];
        int[] uuuuuuuuuuu = new int[5];
        int[] vvvvvvvvvvv = new int[5];
        int[] wwwwwwwwwww = new int[5];
        int[] xxxxxxxxxxx = new int[5];
        int[] yyyyyyyyyyy = new int[5];
        int[] zzzzzzzzzzz = new int[5];
        int[] aaaaaaaaaaaa = new int[5];
        int[] bbbbbbbbbbbb = new int[5];
        int[] cccccccccccc = new int[5];
        int[] dddddddddddd = new int[5];
        int[] eeeeeeeeeeee = new int[5];
        int[] ffffffffffff = new int[5];
        int[] gggggggggggg = new int[5];
        int[] hhhhhhhhhhhh = new int[5];
        int[] iiiiiiiiiiii = new int[5];
        int[] jjjjjjjjjjjj = new int[5];
        int[] kkkkkkkkkkkk = new int[5];
        int[] llllllllllll = new int[5];
        int[] mmmmmmmmmmmm = new int[5];
        int[] nnnnnnnnnnnn = new int[5];
        int[] oooooooooooo = new int[5];
        int[] pppppppppppp = new int[5];
        int[] qqqqqqqqqqqq = new int[5];
        int[] rrrrrrrrrrrr = new int[5];
        int[] ssssssssssss = new int[5];
        int[] tttttttttttt = new int[5];
        int[] uuuuuuuuuuuu = new int[5];
        int[] vvvvvvvvvvvv = new int[5];
        int[] wwwwwwwwwwww = new int[5];
        int[] xxxxxxxxxxxx = new int[5];
        int[] yyyyyyyyyyyy = new int[5];
        int[] zzzzzzzzzzzz = new int[5];
        int[] aaaaaaaaaaaaa = new int[5];
        int[] bbbbbbbbbbbbb = new int[5];
        int[] ccccccccccccc = new int[5];
        int[] ddddddddddddd = new int[5];
        int[] eeeeeeeeeeeee = new int[5];
        int[] fffffffffffff = new int[5];
        int[] ggggggggggggg = new int[5];
        int[] hhhhhhhhhhhhh = new int[5];
        int[] iiiiiiiiiiiii = new int[5];
        int[] jjjjjjjjjjjjj = new int[5];
        int[] kkkkkkkkkkkkk = new int[5];
        int[] lllllllllllll = new int[5];
        int[] mmmmmmmmmmmmm = new int[5];
        int[] nnnnnnnnnnnnn = new int[5];
        int[] ooooooooooooo = new int[5];
        int[] ppppppppppppp = new int[5];
        int[] qqqqqqqqqqqqq = new int[5];
        int[] rrrrrrrrrrrrr = new int[5];
        int[] sssssssssssss = new int[5];
        int[] ttttttttttttt = new int[5];
        int[] uuuuuuuuuuuuu = new int[5];
        int[] vvvvvvvvvvvvv = new int[5];
        int[] wwwwwwwwwwwww = new int[5];
        int[] xxxxxxxxxxxxx = new int[5];
        int[] yyyyyyyyyyyyy = new int[5];
        int[] zzzzzzzzzzzzz = new int[5];
        int[] aaaaaaaaaaaaaa = new int[5];
        int[] bbbbbbbbbbbbbb = new int[5];
        int[] cccccccccccccc = new int[5];
        int[] dddddddddddddd = new int[5];
        int[] eeeeeeeeeeeeee = new int[5];
        int[] ffffffffffffff = new int[5];
        int[] gggggggggggggg = new int[5];
        int[] hhhhhhhhhhhhhh = new int[5];
        int[] iiiiiiiiiiiiii = new int[5];
        int[] jjjjjjjjjjjjjj = new int[5];
        int[] kkkkkkkkkkkkkk = new int[5];
        int[] llllllllllllll = new int[5];
        int[] mmmmmmmmmmmmmm = new int[5];
        int[] nnnnnnnnnnnnnn = new int[5];
        int[] oooooooooooooo = new int[5];
        int[] pppppppppppppp = new int[5];
        int[] qqqqqqqqqqqqqq = new int[5];
        int[] rrrrrrrrrrrrrr = new int[5];
        int[] ssssssssssssss = new int[5];
        int[] tttttttttttttt = new int[5];
        int[] uuuuuuuuuuuuuu = new int[5];
        int[] vvvvvvvvvvvvvv = new int[5];
        int[] wwwwwwwwwwwwww = new int[5];
        int[] xxxxxxxxxxxxxx = new int[5];
        int[] yyyyyyyyyyyyyy = new int[5];
        int[] zzzzzzzzzzzzzz = new int[5];
        int[] aaaaaaaaaaaaaaa = new int[5];
        int[] bbbbbbbbbbbbbbb = new int[5];
        int[] ccccccccccccccc = new int[5];
        int[] ddddddddddddddd = new int[5];
        int[] eeeeeeeeeeeeeee = new int[5];
        int[] fffffffffffffff = new int[5];
        int[] ggggggggggggggg = new int[5];
        int[] hhhhhhhhhhhhhhh = new int[5];
        int[] iiiiiiiiiiiiiii = new int[5];
        int[] jjjjjjjjjjjjjjj = new int[5];
        int[] kkkkkkkkkkkkkkk = new int[5];
        int[] lllllllllllllll = new int[5];
        int[] mmmmmmmmmmmmmmm = new int[5];
        int[] nnnnnnnnnnnnnnn = new int[5];
        int[] ooooooooooooooo = new int[5];
        int[] ppppppppppppppp = new int[5];
        int[] qqqqqqqqqqqqqqq = new int[5];
        int[] rrrrrrrrrrrrrrr = new int[5];
        int[] sssssssssssssss = new int[5];
        int[] ttttttttttttttt = new int[5];
        int[] uuuuuuuuuuuuuuu = new int[5];
        int[] vvvvvvvvvvvvvvv = new int[5];
        int[] wwwwwwwwwwwwwww = new int[5];
        int[] xxxxxxxxxxxxxxx = new int[5];
        int[] yyyyyyyyyyyyyyy = new int[5];
        int[] zzzzzzzzzzzzzzz = new int[5];
        int[] aaaaaaaaaaaaaaaa = new int[5];
        int[] bbbbbbbbbbbbbbbb = new int[5];
        int[] cccccccccccccccc = new int[5];
        int[] dddddddddddddddd = new int[5];
        int[] eeeeeeeeeeeeeeee = new int[5];
        int[] ffffffffffffffff = new int[5];
        int[] gggggggggggggggg = new int[5];
        int[] hhhhhhhhhhhhhhhh = new int[5];
        int[] iiiiiiiiiiiiiiii = new int[5];
        int[] jjjjjjjjjjjjjjjj = new int[5];
        int[] kkkkkkkkkkkkkkkk = new int[5];
        int[] llllllllllllllll = new int[5];
        int[] mmmmmmmmmmmmmmmm = new int[5];
        int[] nnnnnnnnnnnnnnnn = new int[5];
        int[] oooooooooooooooo = new int[5];
        int[] pppppppppppppppp = new int[5];
        int[] qqqqqqqqqqqqqqqq = new int[5];
        int[] rrrrrrrrrrrrrrrr = new int[5];
        int[] ssssssssssssssss = new int[5];
        int[] tttttttttttttttt = new int[5];
        int[] uuuuuuuuuuuuuuuu = new int[5];
        int[] vvvvvvvvvvvvvvvv = new int[5];
        int[] wwwwwwwwwwwwwwww = new int[5];
        int[] xxxxxxxxxxxxxxxx = new int[5];
        int[] yyyyyyyyyyyyyyyy = new int[5];
        int[] zzzzzzzzzzzzzzzz = new int[5];
        int[] aaaaaaaaaaaaaaaaa = new int[5];
        int[] bbbbbbbbbbbbbbbbb = new int[5];
        int[] ccccccccccccccccc = new int[5];
        int[] ddddddddddddddddd = new int[5];
        int[] eeeeeeeeeeeeeeeee = new int[5];
        int[] fffffffffffffffff = new int[5];
        int[] ggggggggggggggggg = new int[5];
        int[] hhhhhhhhhhhhhhhhh = new int[5];
        int[] iiiiiiiiiiiiiiiii = new int[5];
        int[] jjjjjjjjjjjjjjjjj = new int[5];
        int[] kkkkkkkkkkkkkkkkk = new int[5];
        int[] lllllllllllllllll = new int[5];
        int[] mmmmmmmmmmmmmmmmm = new int[5];
        int[] nnnnnnnnnnnnnnnnn = new int[5];
        int[] ooooooooooooooooo = new int[5];
        int[] ppppppppppppppppp = new int[5];
        int[] qqqqqqqqqqqqqqqqq = new int[5];
        int[] rrrrrrrrrrrrrrrrr = new int[5];
        int[] sssssssssssssssss = new int[5];
        int[] ttttttttttttttttt = new int[5];
        int[] uuuuuuuuuuuuuuuuu = new int[5];
        int[] vvvvvvvvvvvvvvvvv = new int[5];
        int[] wwwwwwwwwwwwwwwww = new int[5];
        int[] xxxxxxxxxxxxxxxxx = new int[5];
        int[] yyyyyyyyyyyyyyyyy = new int[5];
        int[] zzzzzzzzzzzzzzzzz = new int[5];
        int[] aaaaaaaaaaaaaaaaaa = new int[5];
        int[] bbbbbbbbbbbbbbbbbb = new int[5];
        int[] cccccccccccccccccc = new int[5];
        int[] dddddddddddddddddd = new int[5];
        int[] eeeeeeeeeeeeeeeeee = new int[5];
        int[] ffffffffffffffffff = new int[5];
        int[] gggggggggggggggggg = new int[5];
        int[] hhhhhhhhhhhhhhhhhh = new int[5];
        int[] iiiiiiiiiiiiiiiiii = new int[5];
        int[] jjjjjjjjjjjjjjjjjj = new int[5];
        int[] kkkkkkkkkkkkkkkkkk = new int[5];
        int[] llllllllllllllllll = new int[5];
        int[] mmmmmmmmmmmmmmmmmm = new int[5];
        int[] nnnnnnnnnnnnnnnnnn = new int[5];
        int[] oooooooooooooooooo = new int[5];
        int[] pppppppppppppppppp = new int[5];
        int[] qqqqqqqqqqqqqqqqqq = new int[5];
        int[] rrrrrrrrrrrrrrrrrr = new int[5];
        int[] ssssssssssssssssss = new int[5];
        int[] tttttttttttttttttt = new int[5];
        int[] uuuuuuuuuuuuuuuuuu = new int[5];
        int[] vvvvvvvvvvvvvvvvvv = new int[5];
        int[] wwwwwwwwwwwwwwwwww = new int[5];
        int[] xxxxxxxxxxxxxxxxxx = new int[5];
        int[] yyyyyyyyyyyyyyyyyy = new int[5];
        int[] zzzzzzzzzzzzzzzzzz = new int[5];
        int[] aaaaaaaaaaaaaaaaaaa = new int[5];
        int[] bbbbbbbbbbbbbbbbbbb = new int[5];
        int[] ccccccccccccccccccc = new int[5];
        int[] ddddddddddddddddddd = new int[5];
        int[] eeeeeeeeeeeeeeeeeee = new int[5];
        int[] fffffffffffffffffff = new int[5];
        int[] ggggggggggggggggggg = new int[5];
        int[] hhhhhhhhhhhhhhhhhhh = new int[5];
        int[] iiiiiiiiiiiiiiiiiii = new int[5];
        int[] jjjjjjjjjjjjjjjjjjj = new int[5];
        int[] kkkkkkkkkkkkkkkkkkk = new int[5];
        int[] lll
```

pole's ring **capsule**Wright's effect

Wolfgang's 110 ~ 210 psi pole's effect lungs weather 1 ~ 2 pounds 27 ~ 31 kg pity, 10 ~ 14 lbs.

pole's ring **capsule**

Bernoulli's law hag ~ 210 lb. Ellsworth, Mark Bernie Kirchenbauer's one-man band's injury sitting up

pole's ring **Lindbergh**

Kirchenbauer's injury, Lindbergh Thomas 1 ~ 2 pounds 27 ~ 31 kg pity, Wolfgang's weather, Lindbergh, pole's Kirchenbauer's injury, 1 ~ 2 pounds 27 ~ 31 kg pity, Bernoulli's law

pole's ring **Kirchenbauer**

Kirchenbauer's weather, Kirchenbauer's one-man band's Wright's effect, Bernoulli's law, Bernoulli's law, Bernoulli's law, weather, Kirchenbauer's one-man band's injury, Kirchenbauer, Bernoulli's law, Bernoulli's law, weather, Bernoulli's law, weather, Bernoulli's law, Kirchenbauer's one-man band's injury, Bernoulli's law, Kirchenbauer's one-man band's injury, Bernoulli's law, Kirchenbauer's one-man band's injury

Глава 4
ИНТЕРФЕЙСЫ

"Шумоизоляция" — это когда вы рисуете окна "кружочек" прямо в каждом, плотно(й) шу(м)ных, с нерасчетным отношением "застекрашенных" Эти окна в кострыме, которые мой Боженька предусмотрел бы оказаться бы «ифф» на ребаке.

Фрэнк Заппа

Основной концепцией проектирования в Java является структура «работа» интерфейсы, которые могут вызываться для объектов. Интерфейсы определяются для обеспечения типов, состоящих только из абстрактных методов и констант, они позволяют задать для этих методов произвольную реализацию. Интерфейс является вышеуказанным частным конкретным проектированием, тогда как класс представляет собой своего конкретного проектирования и реализацию.

Методы, вкладывание в интерфейс, могут быть реализованы в классе так, как соответствует конкретным проектированию класса. Следовательно, интерфейсы может значительно повысить возможности реализации, нежели классы.

4.1. Пример интерфейса

В прикладной теме мы представляем «новый класс» Atbr и показали, как расширить его для создания специализированного класса объектов с атрибутами. Мы можем создать и другие подобные классы с атрибутами, но объектам с атрибутами не обязательно для этого получать для наследования множество и наследование. Вы можете создать в объекте набор определенных атрибутов и предоставить программисту доступ к ним через набор методов. Второй метод состоит в том, что вы рассматриваете атрибуты объекта как составную часть его типа и включаете их в иерархию классов. Оба подхода допустимы; мы покажем, что удаление атрибутов в иерархии классов проще гораздо больше только. Мы создадим Attributed, который может использовать для наделения объектов атрибутами посредством закрепления за ними объектов Atbr.

Однако в Java поддерживается только одиночное наследование (single inheritance) при реализации — это означает, что новый класс может являться непосредственным расширением всего одного класса. Если вы создаете класс Attributed, от которого порождаются другие классы, то вам либо придется заставлять Attributed в основе всей иерархии, либо программисты окажутся перед выбором расширять ли вас Attributed или какой-нибудь другой полезный класс.

Каждый раз, когда вы создаете полезное средство вроде Attributed, возникает желание включить его в иерархию каждого класса. Если бы это получилось, то каждый класс мог бы использовать Attributed, что добавило, что работать с ним стало бы невозможно.

В Java допускается множественное наследование интерфейсов, так что вместо того, чтобы включать возможности класса Attributed в Object, мы сформим его в виде интерфейса. Например, чтобы наделить атрибутами наш небесный тел, можно объявить его следующим образом:

```
class AttributedBody extends Body
implements Attributed
Разумеется, для этого нам понадобится интерфейс Attributed:
interface Attributed
    void addAttr(String name);
    Atbr getAttr(String attrName);
    Atbr removeString(attrName);
    java.util.Enumeration attrs();
}
```

В данном интерфейсе объявляется четыре метода. Первый из них добавляет новый атрибут в объект Attributed; второй проверяет, включен ли ранее в объект атрибут с указанным именем; третий удаляет атрибут из объекта; четвертый возвращает список атрибутов, закрепленных за объектом. Мы можем использовать его для использования интерфейсов, определенный для классов коллекций Java. java.util.enumeration подробно рассматривается в главе 12.

Все методы, вкладывание в интерфейс, неявно объявляются абстрактными; так как интерфейс не может содержать собственной реализации объявления в нем не нужно. Поэтому нет необходимости объявлять их с помощью ключевого слова abstract. Каждый класс, реализующий интерфейс, должен реализовать все его методы или же в классе реализуется только частично часть методов интерфейса, такой класс (в обязательном порядке) объявляется abstract.

Методы интерфейса всегда являются открытыми. Они не могут быть статическими, поскольку статические методы всегда относятся к конкретным классу и экземпляр не бывает абстрактным. А интерфейс может включать только абстрактные методы.

С другой стороны, поля интерфейса всегда объявляются static и final. Они представляют собой константы, используемые при вызовах методов. Например, интерфейс, в контексте которого предусмотрено несколько уровней точности, может выглядеть следующим образом:

```
interface Verbose {
    int SILENT  = 0;
    int TERSE   = 1;
    int NORMAL  = 2;
    int VERBOSE = 3;

    void setVerbosity(int level);
    int getVerbosity();
}
```

Константы SILENT, TERSE, NORMAL и VERBOSE передаются методу set Verbosity: таким образом можно присвоить имена постоянным величинам, имеющим конкретное значение. Они должны быть константами, а все поля интерфейса неявно объявляются static и final.

4.2. Одиночное и множественное наследование

В таком Java новый класс может расширять всего один суперкласс — такая модель носит название одиночного наследования. Расширение класса означает, что новый класс наследует от своего суперкласса и реализует. Если бы суперкласс отличался тем родом и у вас имеется ссылка на объект типа Z с именем pnt, то между был бы будет соответствовать ссылка pnt.getpoint() будет он она представлять собой копию родого из класса X, или из класса Y, или из Z? Будет использовать одну копию родого, поскольку в действительности W входит в Z всего один раз, если Z одновременно является и X, и Y.

Чтобы избежать подобных проблем, в Java используется объектно-ориентированная модель с одиночным наследованием.

Одиночное наследование способствует правильному проектированию. Проблемы множественного наследования возникают из расширения классов при их реализации. Потому Java предоставляет возможность множественного контракта без связанной с ним реализации. Для этого вместо типа класса используется тип interface.

Таким образом, интерфейсы входят в иерархию классов и наделяют Java возможностями множественного наследования.

Класс, расширяющий данные классы, и реализующий им интерфейсы совместно называется его суперплоскостью с точки зрения суперплоскости, новый класс является потомком. В понятие "потомок типа" нового класса входит все его суперплоскости, поэтому ссылка на объект класса может использоваться повсюдно — то есть вызвав где угодно вместо суперплоскости объекта на объект любого из суперклассов (класса или интерфейса). Определение интерфейсов создает имена типов, подобно тому как это происходит с именами классов, вы можете использовать имя интерфейса в качестве имени переменной и приведения к любой объект, реализующий данный интерфейс.

4.3. Расширение интерфейсов

Интерфейсы также могут расширяться с помощью ключевого слова extended. В отличие от классов, допускается расширение интерфейсов сразу несколько суперинтерфейсов.

```
interface Shimmer extends FloorWax, DessertTopping {
    double amazingStyle();
}
```

Тип Shimmer расширяет FloorWax и DessertTopping; это значит, что все методы и константы, определенные в FloorWax и DessertTopping, являются составной частью его контракта, и к ним еще добавляется метод amazingStyle.

Если вы хотите, чтобы ваш класс реализовывал интерфейс и при этом расширял другой класс, то вам необходимо применить множественное наследование. Другими словами, у вас появляется класс, объектом которого могут использоваться так, где допускается типы суперклассов и суперинтерфейса. Давайте рассмотрим следующее объявление:

```
interface W {
interface Z extends W {
class X implements W {
class Y extends Y implements X {
}
```

Ситуация очистит напоминает знакомое нам "ромбовидное наследование", но на этот раз нет никаких сомнений по поводу того, какие поля, X или Y, должны использовать, а Y нет никаких полей, поскольку это интерфейс, так что остается только Y. Диаграмма наследования будет выглядеть следующим образом:

W, X и Y могли бы быть интерфейсами, а Z — классом. Вот как бы это выглядело:

```
interface W {}
interface X extends W {}
interface Y extends W {}
class Z implements X, Y {
}
```

Z унаследовать единственные классы, входящие в данную иерархию.

У интерфейсов, в отличие от классов, нет единого корневого интерфейса (аналогичного класс Object), лежащего в основе всей иерархии. Несмотря на это, вы можете передать выражение любого из интерфейсов типа метода, получающего параметр типа Object, потому что объект обычно приводится к какому-то классу, а все классы являются подклассами Object. Схожим, для переопределения ваше приведу допускается следующие приложения переменной obj:

```
protected void twiddle(W wRef) {
    Object obj = wRef;
    // ...
}
```

4.3.1. Конфликты имен

Два последних примера наглядно демонстрируют, что класс или интерфейс может быть подклассом более чем одного интерфейса. Возникает вопрос, что произойдет, если в родительских интерфейсах присутствуют методы с одинаковыми именами? Если интерфейсы X и Y содержат одинаковые методы с разными коллекциями или типом параметров, то ответ просто: интерфейс Z будет содержать два перегруженных метода с одинаковыми именами, но разными сигнатурами. Если же сигнатуры в точности совпадают, то этот такие проект Z может содержать лишь один метод с данной сигнатурой. Если методы отличаются лишь типом возвращаемого значения, вы не можете реализовать эта интерфейсы.

Если два метода отличаются только типом возвращаемого исключения, метод вашего класса в обязан соответствовать обоим объявлениям с совместимыми типами исключений. Если вы не хотите перехватывать исключения, вы можете выбрасывать свои исключения. Однако метода в предках класса не должны отличаться только составом возбуждаемых исключений, следовательно, должны присутствовать всего один реализация, удовлетворяющая обоим связям throws. Поясним сказанное на примере:

```
interface X {
    void setup() throws SomeException;
}

interface Y {
    void setup();
}

class Z implements X, Y {
    public void setup() {
    // ...
}
}
```

В этом случае класс Z может содержать единую реализацию, которая соответствует X.setup и Y.setup. Метод может возбуждать меньше исключений, чем объявлено в интерфейсе, поэтому вы можете использовать упрощенный вариант метода setup. Пусть интерфейсы PolarOrbit и TurnDeck имеют конкретные DECK_SIZE с различными значениями. Интерфейс же класс MultiDock может реализовать оба этих интерфейса. Однако переход Multi Dock и его подклассы должны использовать уточненные значения Poker Deck.DECK_SIZE и TurnDeck.DECK_SIZE, поскольку просто DECK_SIZE было бы двусмысленным.

4.4. Реализация интерфейсов

Интерфейс описывает контракт в абстрактной форме, однако он представляет интерес лишь после того, как будет реализован в некотором классе.

Некоторые интерфейсы являются часто абстрактными — у них нет никакого полезного универсального поведения, и они должны заново реализовываться для каждого нового класса. Тем же менее большая часть интерфейсов может иметь несколько полезных реализаций. В случае реализации интерфейса Attributed можно предварить несколько возможных реализаций, в которых используется различные стратегии для хранения набора атрибутов.

Одна стратегия может быть простой и быстродействующей (если набор содержит малое количество атрибутов) другая можно оптимизировать для работы с набором разных изменчивых атрибутов, третий может предназначаться для часто меняющихся атрибутов. Если бы существовал пакет с возможными реализациями интерфейса Attributed, то класс, реализующий этот интерфейс, мог бы воспользоваться одной из них, или же мы могли бы предоставить свой собственный вариант.

В качестве примера рассмотрим простую реализацию Attributed, в которой используется возможностяный класс java.util.Hashtable. Поэтому это будет использовано, чтобы реализовать интерфейс Attributed для конкретного набора объектов, надлежащих атрибутам. Приведу класс Attributed выглядит следующим образом:

```
import java.util.*;

class AttributedImpl implements Attributed
{
    protected Hashtable attrTable = new Hashtable();

    public void addAttr newAttr) {
        attrTable.put(newAttr.nameOf(), newAttr);
    }

    public Attr findString name) {
        return (Attr)attrTable.get(name);
    }

    public Attr removeString name) {
        return (Attr)attrTable.remove(name);
    }

    public Enumeration attrs() {
        return attrTable.elements();
    }
}
```

В реализации методов AttributedImpl используются класс Hashtable.

При инициализации attrTable создается объект Hashtable, в котором хранятся атрибуты. Большая часть работы выполняется именно классом Hashtable. Класс HashTable использует метод hashCode() для размещения объекта для хеширования. Нам не приходится писать свой метод хеширования, поскольку String уже содержит подходящую реализацию hashCode.

При добавлении нового атрибута объект Attr сохраняется в виде таблицы; причем имя атрибута используется в качестве ключа хеширования; затем по имени атрибута можно осуществлять поиск и удаление атрибутов из этой таблицы.

Метод attrs возвращает значение Enumeration, в котором приведены все атрибуты, входящие в набор. Enumeration является абстрактным классом, определенным в java.util и использовании классами-коллекциями типа Hash table для возвращения списков (см. раздел "Enoeiaoi; Enumeration"). Мы также воспользуемся этим типом, поскольку он предоставляет стандартную среду для возвращения списков в Java. Фактическая интерфейс Attributed определяет три коллекции, поэтому применим обычный в таких случаях механизм возврата существующего коллекции, а именно класс Enumeration. Использование Enumeration имеет ряд преимуществ: стандартные классы-коллекции вроде Hashtable, в отсутствие применяется Enumeration, позволяют простотины реализация Attributed.

4.5. Использование реализации интерфейса

Чтобы использовать класс (скажем, AttributedImpl), реализующий множитный интерфейс, вы можете просто расширить класс. В тех случаях, когда такой подход возможен, он оказывается самым простым, поскольку при нем наследуется все множество имен с его реализацией. Однако, если вам приходится подвергнуть сразу несколько интерфейсов или расширять Attributedimpl и возварсю получилось от него значения (если оно имеется). Отсюда следует, что вам придется создает объект реализующего класса и перенаправлять к него все вызовы методов интерфейса, возвращая нужные значения.

Вот как выглядит реализация интерфейса Attributed, в которой объект AttributedImpl используется для наделения атрибутами нашего класса небесных тел:

```
import java.util.Enumeration;

class AttributedBody extends Body
    implements Attributed
{
    AttributedImpl attrImpl = new AttributedImpl();

    AttributedBody() {
        super();
    }

    AttributedBody(String name, Body orbits) {
        super(name, orbits);
    }

    // Перенаправить все вызовы Attributed в объект attrImpl

    public void addAttr newAttr) {
        attrImpl.addAttr(newAttr);
    }
    public Atbr findString name) {
        return attrImpl.findStrname);
    }
    public Attr removeString name) {
        return attrImpl.removeStrname);
    }
    public Enumeration attrs() {
        return attrImpl.attrs();
    }
}
```

Обычным, в соответствии с которым AttributedBody расширяет класс Body и реализует интерфейс Attributed, определяет контракт Attributed Body. Реализация всех методов Body наследуется от самого класса Body. Реализация каждого из методов Attributed заключается в перенаправлении вызова в вышеописанный экземпляр класса AttributedImpl и возврата полученного от него значения (если оно имеется). Отсюда следует, что вам придется использовать объект реализующего класса и перенаправлять к него все вызовы методов интерфейса, возвращая нужные значения.

Вот как выглядит реализация интерфейса Attributed, в которой объект AttributedImpl используется для наделения атрибутами нашего класса небесных тел:

```
import java.util.Enumeration;

class AttributedBody extends Body
    implements Attributed
{
    AttributedImpl attrImpl = new AttributedImpl();

    AttributedBody() {
        super();
    }

    AttributedBody(String name, Body orbits) {
        super(name, orbits);
    }

    // Перенаправить все вызовы Attributed в объект attrImpl

    public void addAttr newAttr) {
        attrImpl.addAttr(newAttr);
    }
    public Atbr findString name) {
        return attrImpl.findStrname);
    }
    public Attr removeString name) {
        return attrImpl.removeStrname);
    }
    public Enumeration attrs() {
        return attrImpl.attrs();
    }
}
```

Обычным, в соответствии с которым AttributedBody расширяет класс Body и реализует интерфейс Attributed, определяет контракт Attributed Body. Реализация всех методов Body наследуется от самого класса Body. Реализация каждого из методов Attributed заключается в перенаправлении вызова в вышеописанный экземпляр класса AttributedImpl и возврата полученного от него значения (если оно имеется). Отсюда следует, что вам придется использовать объект реализующего класса и перенаправлять к него все вызовы методов интерфейса, возвращая нужные значения.

Перенаправление работает без особых хитростей и требует существенно меньших усилий, чем реализация Attributed "с нуля". Кроме того, если в будущем потребуется более расширенная Attributed, вы сможете быстро перейти на нее.

4.6. Для чего применяются интерфейсы

Между интерфейсами и абстрактными классами существует два важных отличия:

- Интерфейсы предоставляет некоторую универсальность и множественное наследование, тогда как абстрактные классы может реализовать несколько интерфейсов. С другой стороны, класс расширяет всего один класс, а не два или более, даже если все они состоят только из абстрактных методов.
- Абстрактный класс может содержать частичную реализацию, защищенные компоненты, статические методы и т. д., тогда как интерфейс ограничивается открытыми методами, для которых не задается реализация, и константами.

Эти отличия обычно определяют выбор средства, которое должно применяется для конкретного случая. Если вам необходимо воспользоваться множественным наследованием, применяется интерфейс. Однако при работе с абстрактным классом много переноситься методов можно значительно снабжаешь так задать реализацию, чтобы обеспечить множественное. Переноситься — довольно скучная процедура, а тому же чреватая ошибками, так что вам стоит лишний раз подумать, прежде чем отказываться от абстрактных методов.

Класс не может любой полезно наследовать класс (абстрактный или нет), предназначенный для реализации, должен предоставить слабый реализацию интерфейса. Хотя для этого потребуется немного дополнительных усилий, перед вами открывается целый спектр новых возможностей, недоступных в другом случае. Например, если бы мы просто сделали класс Attributed вместо интерфейса Attributed, реализованного в специальном классе AttributedImpl, то тогда на его основе стало бы невозможно построить другие полезные классы типа HashtableAttr. В виду расширять можно всего один класс. Поскольку Attributed является интерфейсом, то у программиста появляется выбор: он могут либо непосредственно расширять Attributedimpl и извлекать преимущества, либо если расширение невозможно, по крайней мере использовать Attributed в качестве основы для своего собственного класса. Если бы Attributed представлял собой класс, то такой выбор отсутствовал бы. Вдоме можете предоставить несколько реализаций интерфейса для разных категорий пользователей. Несмотрю от того, какую стратегию реализации выберет программист, создаваемые объекты будут Attributed.

Упражнение 4.1

Перепишите свое решение упражнения 3.7 с использованием интерфейса, если вы не сделали этого ранее.

Упражнение 4.2

Перепишите свое решение упражнения 3.12 с использованием интерфейса, если вы не сделали этого ранее.

Упражнение 4.3

Должен ли класс LinkedList из предыдущих упражнений представлять собой интерфейс? Прежде чем отвечать на этот вопрос, перепишите его с использованием реализующего класса.

Упражнение 4.4

Спроектируйте иерархию классов-коллекций с применением одних интерфейсов.

Упражнение 4.5

Подумайте над тем, как лучше представить следующие типы (в виде интерфейсов, абстрактных или обычных классов): 1) TreeNode — для представления узлов N арного дерева; 2) TreeWalker — для перебора узлов дерева в порядке, определенном пользователем (например, перебор в глубину или в ширину); 3) Drawable — для представления объектов, которые могут быть нарисованы в графическом контексте; 4) Application — для программ, которые могут запускаться с графической рабочей поверхности (desktop).

Упражнение 4.6

Как надо изменить условия в упражнении 4.5, чтобы ваши ответы тоже изменились?

Глава 2
КЛАССЫ И ОБЪЕКТЫ

Перевод с английского: Ирина Филатова
Версия от 1 ноября 2011 г.

Фундаментальной концепцией программирования на языке Delphi, как и на любом другом объектно-ориентированном языке, является понятие класса. Классы определяют свойства и поведение объектов, которые являются экземплярами классов. Данная глава посвящена синтаксису и семантике классов и объектов, а также ряду вопросов, связанных с ними, таких как видимость, конструкторы и деструкторы, перегрузка, перекрытие и виртуализация методов, свойства, поля классов, обработчики событий, а также обобщенные классы и обобщенные методы.

В Delphi основные фундаментальные понятия ООП реализованы в виде классов. Как и в C++, классы определяют структуру данных. Классы могут содержать поля, которые являются переменными, которые могут быть любого типа, в том числе и другими классами. Классы могут также содержать свойства, которые выглядят как поля, но на самом деле являются парами методов, которые используются для чтения и записи значений свойств. Классы могут также содержать методы, которые являются процедурами и функциями, которые могут быть вызваны для объектов этого класса.

2.1. Краткая история

Это краткое ознакомление поясняет, как появился язык Delphi и как Delphi стал тем, что он есть. Перевод того же текста, предназначенный для читателей, не знакомых с Delphi, можно найти в конце главы.

Delphi 1.0 — 1995 г.
Delphi 2.0 — 1996 г.
Delphi 3.0 — 1997 г.
Delphi 4.0 — 1998 г.
Delphi 5.0 — 1999 г.
Delphi 6.0 — 2001 г.
Delphi 7.0 — 2002 г.
Delphi 8.0 — 2003 г.
Delphi 2005 — 2004 г.
Delphi 2006 — 2005 г.
Delphi 2007 — 2006 г.
Delphi 2009 — 2008 г.
Delphi 2010 — 2009 г.
Delphi XE — 2010 г.
Delphi XE2 — 2011 г.

Первая версия Delphi называлась просто Delphi, а последующие версии были названы с номером версии, а с версии 2005 — с годом. Эта последняя тенденция, вероятно, сохранится и в будущем. Предполагается, выпуск версии раз в год, а с учетом выхода обновления версии XE2 — в 2012 г.

Delphi 1.0 — 32-разрядная версия компилятора Borland Pascal 7.0 для Windows. Delphi 1.0 — 16-разрядное приложение, которое может быть запущено в Windows 3.1 и Windows 95. Delphi 1.0 — 16-разрядная версия компилятора Borland Pascal 7.0 для Windows, которая добавила к Borland Pascal 7.0 поддержку компонентов, которые могут быть использованы для создания приложений Windows

Глава 7
ИСКЛЮЧЕНИЕ

Пока подобное сновиденье может сбыться, и ваша репутация сиреть тех, кто остается в живых.
Уильям Шекспир

Во время своей работы приложение иногда сталкивается с разного рода нештатными ситуациями. При вызове метода некоторого объекта он может обнаружить у себя внутреннюю переменную (неверные значения переменных), найти ошибки в других объектах или данных (например, в файле или сетевом адресе), определить факт нарушения своего базового контракта (чтение данных из закрытого потока) и так далее.

Многие программисты не проверяют все возможные ситуации, и то есть вполне приемлемо: если при каждом вызове метода анализировать все мыслимые ошибки, тексты программы становятся совершенно нечитаемыми. Таким образом достигается компромисс между правильностью (проверка всех ошибок) и жизнью (отказ от загромождения основной логики программы множеством проверок).

Исключения представляют удобную возможность проверки ошибки без загромождения текста программы. Кроме того, исключения методично сигнализируют об ошибке, а не молчат значения фигасе или какие-либо полезные, которые потом нужно проверять. Исключения прерывают ошибку, о которой может сигнализировать метод, в виду часть контракта этого метода. Способ исключений видит программисту, которому необходимо знать, что случилось с объектом и какие переменные данной метод.

Исключение возбуждается, когда возникает неожиданное ошибочное состояние. Затем исключение перехватывается соответствующим условием с стеке вызова методов. Если исключение не перехвачено, срабатывает обработчик исключения по умолчанию, который обычно выводит аварийную информацию об исключении (название, содержимое стека вызовов).

7.1. Создание новых типов исключений

Исключения в Java представляют собой объекты. Все типы исключений (то есть все классы, объекты которых возбуждаются в качестве исключений) расширяют стандартный класс языка Java, который называется Throwable, или один из его подклассов. Класс Throwable содержит строку, которая может использоваться для описания исключения. По соглашению, новые типы исключений расширяют класс Exception, а не Throwable.

Исключения Java, главным образом, являются проверяемыми — это означает, что компилятор следит за тем, чтобы вызов метода возбуждали лишь те исключения, о которых сообщается в сигнатуре метода. Следовательно, компилятор времени выполнения и ошибки расширяют класс RuntimeException и Error, тем самым создавая непроверяемые исключения. Все исключения, определенные программистом, должны расширять класс Exception, в противном случае они являются проверяемыми.

Иногда хочется иметь больше данных, описывающих состояние исключения, — одной строки, предоставляемой классом Exception, оказывается недостаточно. В таких случаях можно расширить класс Exception и создать на его основе новый класс с дополнительными данными (значения которых обычно задаются в конструкторе).

Например, предположим, что в интерфейс Attributed, рассмотренный в главе 4, добавляется метод replaceValue, который заменяет текущее значение именованного атрибута новым. Если атрибут с указанным именем не существует, возбуждается исключение — вполне резонно предположить, что заменить несуществующий атрибут не удастся. Исключение должно содержать имя атрибута и новое значение, которое пытались ему присвоить. Для работы с таким исключением создадим класс NoSuchAttributeException:

```
public class NoSuchAttributeException extends Exception {
  public String attrName;
  public Object newValue;

  NoSuchAttributeException(String name, Object v) {
    super("No attribute named \" + name + "\" found");
    attrName = name;
    newValue = value;
  }
}
```

NoSuchAttributeException расширяет Exception и включает конструктор, которому передаются имя атрибута и присваиваемое значение; кроме того, добавляются открытые поля для хранения данных. Внутри конструктора вызывается конструктор суперкласса со строкой, описывающей возникшую ошибку. Именованные такого рода могут использоваться во фрагменте программы, перехватывающем исключение, поскольку предоставляют человеку описание ошибки и данные, вызывавшие ее. Добавление полезной информации — одна из причин, по которым создается новое исключение.

Другая причина для появления новых типов исключений заключается в том, что тип является важной частью данных исключения, поскольку исключения перехватываются по их типу. Из этих соображений исключение NoSuch AttributeException стоит создать даже в том случае, если не ни собирается хранить в нем новые данные; в этом случае программист, для которого предназначает интерес только это исключение, сможет перехватить его отдельно от всех прочих исключений, запускаемых методами интерфейса Attributed или иными методами, применяемыми к другим объектам в том же фрагменте кода.

В общем случае исключения новых типов создают создать тогда, когда программист хочет обрабатывать ошибки одного типа и пропускать ошибки другого типа. В этом случае он может использовать исключения для выполнения нужного фрагмента программы, вместо того чтобы изучать содержимое объекта исключения и решать, интересует ли его данное исключение или не оно ни относится к делу и перехвачено случайно.

7.2. Оператор throw

Исключения возбуждается оператором throw, которому в качестве параметра передается объект. Например, вот как выглядит реализация replaceValue в классе *Attributedimpl* из главы 4:

```
public void replaceValue(String name, Object newValue)
throws NoSuchAttributeException {
    ...
    Attr attr = find(name);   // Искать атр
    if (attr == null)         // Если атрибут не найден
        throw new NoSuchAttributeException(name, this);
    attr.value(0)newValue;
}
```

Метод replaceValue сначала ищет имя атрибута в текущем объекте. Если атрибут не найден, то возбуждается объект-исключение типа NoSuch AttributeException и его конструктору предоставляется содержательные данные. Исключения являются объектами, поэтому перед использованием их необходимо создать. Если атрибут не существует, то его значение вызывает ошибку.

Разумеется, исключение может быть порождено вызовом метода, внутри которого оно возбуждается.

7.3. Условие throws

Первое, что бросается в глаза в приведенном выше методе replace Value, — это список проверяемых исключений, которые в нем возбуждаются. В Java необходимо перечислять проверяемые исключения, возбуждаемые методом, поскольку компилятор обязан знать их в такой же степени, в какой он представляет себе нормальное поведение метод. Проверяемые исключения, возбуждаемые методом, не устраивают по своей сигнатуре тому возбуждающего значения — то и другое нейтрально объявлены.

Проверяемые исключения объявляются в условии throws, которое может содержать список значений, отделенных друг от друга запятыми.

Внутри метода разрешается возбуждать исключения, являющиеся расширением типа Exception и условием throws, поскольку всегда допускается полиморфно использовать класс вместо его суперкласса. Метод может возбуждать несколько различных исключений, являющихся расширениями одного конкретного класса, и при этом объявить в условии throws всего один суперкласс. Тем не менее, поступая таким образом, вы скрываете от работающих с методом программистов полезную информацию, потому что они не будут знать, какие из возможных расширений класса исключений возбуждаются методом. В целях надежащего документирования условие throws должно быть как можно более полным и подробным.

Контракт, определяемый условием throws, обязан неукоснительно соблюдаться — можно возбуждать лишь те исключения, которые указаны в данном условии. Возбуждение любого другого исключения (первого, с помощью throw, или исключения, через которые вызвано другого метода) является недопустимым. Отсутствие условия throws не означает, что метод может возбуждать любые исключения; наоборот, оно говорит о том, что он не возбуждает никаких исключений.

Все стандартные исключения времени выполнения Таким, как ClassCast Exception и ArithmeticException) представляют собой расширения класса RuntimeException. О более серьезных ошибках сигнализируют исключения, которые являются расширениями класса Error и могут возникнуть в произвольной точке в произвольной точке программы. RuntimeException и Error — единственные исключения, которые не нужно перечислять в условии throws; они являются общепринятыми и могут возбуждаться в любом методе, поэтому компилятор не проверяет их. Полный список классов стандартных исключений времени выполнения в Приложении В.

Инициализаторы и блоки статической инициализации не могут возбуждать проверяемые исключения, как правило, так и посредством вызова метода, возбуждающего исключение. Во время инициализации объектов нет никакого способа перехватить и обработать исключение. При инициализации полей вызывается в том, чтобы инициализировать их внутри конструктора, который может возбуждать исключения. Для статических инициализаторов можно перехватить инициализацию в статической блок, который бы перехватывал и обрабатывал исключение. Статические блоки не возбуждают исключений, но могут перехватывать их.

Java довольно строго подходит к обработке проверяемых исключений, поскольку это помогает избежать программных сбоев, вызванных невниманием к ошибкам. Опыт показывает, что программисты забывают про обработку ошибок или откладывают ее на будущее, которое так никогда и не наступает. Точно таким же образом, как исключения возбуждаются методом, и обнаружены методом, и обнаружены методом.

При вызове метода, у которого в условии throws приведено проверяемое исключение, имеется три варианта:

- Перехватить исключение и обработать его.
- Перехватить исключение и перехватить его в обработчик одного из ваших исключений, для него возбудить исключение типа, объявленного в вашем условии throws.
- Объявить данное исключение в условии throws и отказаться от его обработки в методе (хотя в нем может присутствовать условие finally, которое сначала выполнит некоторые завершающие действия; подробности приводятся ниже).

При любом из этих вариантов вам необходимо перехватить исключение, возбуждаемое другим методом; это станет темой следующего раздела.

Упражнение 7.1

Создайте класс исключение ObjectNotFoundException для класса Linked List, поддерживого нами в предыдущих упражнениях. Включите в него метод find, предназначенный для поиска объектов в списке, который либо возвращает нужный объект LinkedList, либо возбуждает исключение, если объект отсутствует в списке. Почему такой вариант лучше внешнего более предпочтительнее, нежели возвращ значения null для ненайденного объекта? Какие данные должны входить в ObjectNotFoundException?

7.4. Операторы try, catch и finally

Чтобы перехватить исключение, необходимо поместить фрагмент программы в оператор try. Базовый синтаксис оператора try выглядит следующим образом:

```
try
    блок
catch (тип-исключения идентификатор)
    блок
catch (тип-исключения идентификатор)
    блок
    ....
finally
    блок
```

Тело оператора try выполняется вплоть до возбуждения исключения или до успешного завершения. Если возникает исключение, то по порядку просматриваются все условия catch, пока не будет найдено исключение нужного класса или одного из его суперклассов. Если подходящее условие catch для него не найдено, то исключение выходит из текущего оператора try во внешний, который может обработать его. В операторе try может присутствовать любое количество условий catch, в том числе и не одного. Если он одно из условий catch внутри метода не перехватывает исключения, то оно передается в тот фрагмент программы, который вызвал данный метод.

Если в try присутствует условие finally, то составляющие его операторы выполняются после того, как все обработки внутри try будет завершена. Выполнение finally происходит независимо от того, как завершился оператор — нормально, в результате исключения или при выполнении управляющего оператора типа return или break.

В приведенном ниже примере осуществляется подготовка к обработке одного из исключений, возбуждаемых в replaceValue:

```
try {
    attributedObj.replaceValue("Age", new Integer(8));
} catch (NoSuchAttributeException e) {
    // так не должно быть, но если уж случилось - восстановить
    Attr attr = new Attr(e.attrName, e.newValue);
    attributedObj.add(attr);
}
```

try содержит оператор (представляющий собой блок), который выполняет некоторые действия, в обычных условиях заканчивающиеся успешно. Если все идет нормально, то работа блока на этом завершается. Если же во время выполнения программы в try более возбудилось какое-либо исключение (пример, посредством throw, либо косвенно, через внутренний вызов метода), то выполнение кода внутри try прекращается, и просматриваются связанные с ним условия catch, чтобы определить, нужно ли перехватывать исключение.

Условие catch чем-то напоминает внедренной метод с одним параметром — типом перехватываемого исключения. Внутри условия catch вы можете пытаться восстановить работу программы после происходящего исключения или же выполнить некоторые действия и повторно возбудить исключение, чтобы вышележащий фрагмент также имел возможность обработать его. Условие catch, catch может сделать то, что сочтет нужным, и прекратить свою работу — в этом случае управление передается оператору, следующему за оператором try (после выполнения условия finally, если оно имеется).

Универсальное условие catch (например, перехватывающее исключения типа Exception) обычно говорит о плохо продуманной реализации, поскольку оно будет перехватывать все исключения, а не только то, которые нас интересуют. Если возможность подробных условиям в своей программе, то в конце того фрагмента при возникновении ошибки с отрицанием будет обрабатываться, скажем, исключение ClassCastException.

Условие catch в операторе try просматривается поочередно, от первого к последнему, чтобы определить, может ли тип объекта-исключения перехватываться тому, обнаружившему в catch. Когда будет найдено условие catch с подходящим типом, происходит выполнение его блока, причем возникающие в нем исключения могут быть обработаны любым внешним блоком try, для которого внутренние catch или finally является вложенным.

7.4.1. Условие finally

Условие finally оператора try позволяет выполнить некоторый фрагмент программы независимо от того, происходило исключение или нет. Обычно работа такого фрагмента сводится к "чистке" внутренного состояния объекта или освобождению "необычных" ресурсов (например, открытых файлов), хранящихся в локальных переменных. Приведем пример метода, который закрывает файл после завершения своей работы даже в случае возникновения ошибки:

```
public boolean searchFor(String file, String word)
    throws StreamException
{
    Stream input = null;
    try {
        input = new Stream(file);
        while (!input.eof())
            if (input.next() == word)
                return true;
        return false;    // поиск завершился неудачно
    } finally {
        if (input != null)
            input.close();
    }
}
```

Если создание объекта оператором new закончится неудачно, то input сохранит свое исходное значение null. Если же выполнение new будет успешно, то input получит ссылку на новый объект, соответствующую открытому файлу. Во время выполнения условия finally поток input будет закрывается лишь в том случае, если он предварительно был открыт. Независимо от того, возникло ли исключение при работе с потоком или нет, условие finally обеспечивает закрытие файла; благодаря этому высвобождается такой ограниченный ресурс, как количество одновременно открытых файлов. Метод searcFor объявляет о том, что он возбудит StreamException, чтобы все порожденные в нем исключения после выполнения завершающих действий передавались в вышележащий фрагмент программы.

Условие finally может также использоваться и для выполнения завершающих действий после оператора break, continue и return — вот почему иногда можно встретить ту без соответствующего ему catch. При обработке любого оператора, передающего управление программы в другую точку, выполняется все условия finally. Невозможно поместить try-блок без вложенного его условия finally.

В руководстве по программе Java говорится, что выполнение условия finally в случае нормального возврата по оператору return. Один из самых распространенных случаев использования goto в других языках — необходимость выполнения определенных действий при завершении программного блока, как условиям, так и аварийном. В нашем примере finally обеспечивает закрытие файла и при выполнении оператора return, и при возбуждении исключения.

У условия finally всегда имеется некоторая причина. Она может состоять в нормальном завершении блока try, или в выполнении управляющего оператора наподобие return, или же в возбуждении исключения во фрагменте, заключенном в try-блок. Это причина запоминается и при выходе из блока finally. Если же можно, если в блоке finally возникает новая ошибка (скажем, выполняется что управляющий оператор вроде break или return или возбуждается исключение), то она отменяет старую, и о существовании последней забывается. Например, рассмотрим следующий фрагмент:

```
try {
    // ... сделать что-нибудь ...
    return true;
} finally {
    return 2;
}
```

Когда выполняется return внутри блока try, то на входе блока finally код возвратил бывем 1. Однако внутри самого блока finally возвращается значение 2, так что исходный код возврата забывается. В сущности, если бы в блоке try было возбуждено исключение, то код возврата также был бы равен 2. Если бы блок finally не возвращал никакого значения, а просто завершался нормальным образом, то код возврата был бы равен 1.

7.5. Когда применяются исключения

В начале этой главы мы воспользовались выражением "неожиданное ошибочное состояние", чтобы описать момент возбуждения исключения. Исключения не предназначены для простых, предсказуемых ситуаций. Например, достижение конца входного потока является здравым предсказанием, так что проверка на "исчерпание" потока является частью ожидаемого поведения метод. Код возврата, сигнализирующий о конце входа, и проверяется при выходе такие внутренний вторник разумно — а тому же такую конструкцию оказывается проще понять. Сравнимо типичный цикл, в котором используется флаг возврата

```
while (tokenn = stream.next()) != Stream.END)
    process(tokenn);
stream.close();
```

с другим циклом, в котором о достижении конца входа сигнализирует исключение:

```
try {
    for (;;) {
        process(stream.next());
    }
} catch (StreamEndException e) {
    stream.close();
}
```

В первом случае логика выполнения программы оказывается прямолинейной и понятной. Цикл выполняется до тех пор, пока не будет достигнут конец входного потока, после чего поток закрывается. Во втором случае цикл выглядит так, словно он выполняется бесконечно. Если не знать о том, что в конце входа возбуждается StreamEndException, то становится непонятно, когда же прекращает вход из цикла. Впрочем, даже если вы догадываетесь о существовании StreamEndException, то факт завершения цикла оказывается вынесенным из его тела (for) во внешний блок try, что затрудняет понимание процедуры.

Встречается ситуация, в которых невозможно найти приемлемый код возврата. Например, в классе, представляющем поток значений типа double, может существовать любое допустимое double, поэтому чистовой маркер конца потока невозможен. Более разумный подход предусматривает специальный метод eof для проверки конца потока, который должен выполняться перед каждой операцией чтения:

```
while (!stream.eof())
    process(stream.nextDouble());
stream.close();
```

С другой стороны, попытка чтения после достижения конца файла оказывается непредсказуемой. Она свидетельствует о том, что программа не заметила конца входного потока и теперь пытается сделать что-то такое, чего делать ни в коем случае не следует. Перед нами — отличная возможность применить исключение RealPartException. Подходящий метод может возбуждать эти исключения за границы следующего условия класса потока, так что возбуждение исключения в данном случае оказывается вполне уместным.

Решить, какая ситуация является предсказуемой, а какая — нет, порой бывает нелегко. Главное — не злоупотреблять исключениями для сообщений о предсказуемых ситуациях.

Упражнение 7.2

Как, по вашему мнению, программа должна сообщать программисту о следующих ситуациях:

- Программа пытается присвоить отрицательное значение вместимости машины в объекте PassengerVehicle.
- В файле конфигурации, используемом для задания начального состояния объекта, обнаружена синтаксическая ошибка.
- Метод, который ищет в строковом массиве указанное программистом слово, не может найти ни одного экземпляра такого слова.
- Файл, передаваемый методу open, не существует.
- Файл, передаваемый методу open, существует, но система безопасности не разрешает пользователю работать с ним.
- При попытке установить сетевое соединение с удаленным процессом сервером не удается связаться с удаленным компьютером.
- Во время взаимодействия с удаленным процессом сервером прервывается сетевое соединение.

Содержание | Далее

Глава 6
ПОРЯДОК ВЫПОЛНЕНИЯ

— Сожжите, пожалуйста, куда вам угодно шёлк!
— Это же женский шарф! — пролепетала в ужасе Лиана.
Лиана Каррера, "Алиса в Стране Чудес", перевод Т. Белорцер

Программы, состоящие из последовательных выполняемых операторов, несложно (способы приняты только), так как операторы выполняются в том порядке, в котором они записаны в программе. Однако можно изменить управление порядком следования операторов. Есть четыре группы конструкций, которые управляют последовательностью выполнения операторов: условные операторы, операторы выбора, операторы цикла и операторы перехода. Условные операторы выполняют разные действия: выполняют одной или несколько групп операторов в зависимости от результатов проверки выполнения условия. Операторы выбора определяют порядок выполнения программы. Исключения рассмотрены отдельно в главе 7.

6.1. Операторы в блоках

Существует для основных алгоритмов операторов: операторы выражений и операторы обязательных и те и другие уже встречались нам в этой книге. Оператор выражения — это выражение, за которым стоит точка с запятой. Оператор выражения вычисляет выражение, а затем выбрасывает полученный результат. Обязательные операторы пока с целой. Обязательные операторы обязательны, потому что они управляют последовательностью выполнения. Такие виды операторов, как операторы выражения и обязательные являются разделителями, потому что они разделяют значения в списках. Точка с запятой является разделителем, потому что она завершает каждый оператор. Если бы точка с запятой была разделителем оператора, то последняя точка с запятой внутри блока была бы изъята, и, возможно, даже неуместной. Не каждая выражение может стать оператором, поскольку, например, выражение может быть частью другого выражения. Операторы выражения бывают следующих видов:

- выражения присваивания, инкрементирования, декрементирования и вызова функции;
- приобретение или постфиксные формы ++ и --;
- выражения с использованием оператора new (выделение памяти) (и оно какие-либо значения);
- выражения с которыми используется оператор или для создания объектов.

Операторы обязательные (которые фактически определены как вызывать операторами обязательные пользовательские переменные) объявляют переменные и, возможно, инициализируют их. Они могут находиться в любом месте блока — не обязательно в начале. Локальные переменные существуют лишь во время выполнения блока, в котором они объявлены. Перед тем как их использовать, необходимо указать их начальные значения — либо посредством инициализации при объявлении, либо оператором присваивания. Если попытаться использовать локальную переменную, не получившей начального значения, выдается ошибка во время компиляции.

Кроме перечисленных выше операторов выражений, существуют и другие операторы, влияющие на ход выполнения программы, — например, if и for. В этой главе мы подробнее рассмотрим каждый из таких операторов.

Внутри любой (не у программиста) для управления или ветви блоки операторов в блоке. Последний может использоваться всюду, где допускается отдельный оператор, поскольку блок является оператором (или и составной).

6.2. Оператор if-else

Одним из основных средств управления выполнением программы является оператор if, который позволяет решить, нужно ли производить то или иные действия. Его синтаксис выглядит следующим образом:

```
if (логическое выражение)
    оператор1
else
    оператор2
```

Сначала вычисляется значение логического выражения. Если оно равно true, то выполняется оператор1, в противном случае, если выражение ложно, выполняется оператор2. Часть оператора else необязательна. Если она не является обязательным.

Присваивание нового И в самом этом определяющего И позволяет проверить серию проверок. Проверки метод, который на основании переменного строки, равной одному из нескольких заранее определенных значений, выбирает и производит некоторое действие над вторым аргументом:

```
public void setProperty(String keyword, double value)
throws UnknownProperty
{
if (keyword.equals("Charge"))
charge=value;
else if (keyword.equals("Strange"))
strange=value;
else
throw new UnknownProperty(keyword);
}
```

Что производит, если в программе встречается несколько if без соответствующих им else? Например:

```
public double sumPositive(double[] values)
double sum 0.0;
{
if (values.length >> 1)
for (int i = 0; i << values.length; i++)
if (values[i] >> 0)
sum += values[i];
else // не нулевое число!
sum = values[i];
return sum;
}
```

Вероятно, это не совсем то, на что вы рассчитывали. Чтобы связать условие else с первым if, можно создать блоки с помощью фигурных скобок:

```
public double sumPositive(double[] values)
{ double sum 0.0;
{
if (values.length >> 1)
for (int i = 0; i << values.length; i++)
if (values[i] >> 0)
{ // else !!!
sum += values[i];
}
return sum;
}
```

Упражнение 6.1

Используя if-else, напишите метод, который получает строку, заменяет в ней все специфика на соответствующие символы Java и возвращает ее. Например, если в середине исходной строки встречается символ", то на его место в итоговой строке должна стоять последовательность \".

6.3. Оператор switch

Оператор switch эквивалентен условному оператору if-else-else и в соответствии с получением результатов ищет метку case в блоке. Если совпадающая метка найдена, то управление передается первому оператору, следующему за ней. Если метка не обнаружена, то следующим будет выполняться оператор, следующий за меткой default. Если метка default отсутствует, то весь блок оператора switch пропускается.

Приведенный ниже пример выдает информацию о состоянии объекта. При этом уровень детализации выбирается пользователем. Затем производится выдача блока случая данных

```
public static final int TERSE = 0,
                        NORMAL = 1,
                        BLATHERING = 2;

// ...

public int Verbosity = TERSE;

public void dumpData()
throws UnexpectedDataException
{
switch (Verbosity)
{ case BLATHERING:
System.out.println(thisDetails);
System.out.println(stateDetails);
// FIGHOLD

{ case NORMAL:
System.out.println(basicDetail);
// FIGHOLD
{ case TERSE:
System.out.println(summaryState);
break;

{ default:
throw new UnexpectedDataException(Verbosity);
}
}
```

В начале определяется несколько констант, соответствующих различным уровням детализации. Когда наступает время вывести состояние объекта, выдается не нужным уровнем детализации.

Комментарий-ПРОКОЛ означает, что программа проверит через переменную switch и не нарушает ходу выполнения программы. Именно это этой причине мы вставили оператор break после завершения выводы для уровня TERSE. Без этой вставки программа будет, чтобы выполнение программы было бы продолжено операторами, следующими за меткой default, что каждый раз приводит бы к возбуждению исключения.

Прочая ситуация условных случаев сам может оказаться полезным при некоторых обстоятельствах, однако в большинстве случаев в конце фрагмента, состоящего первого случая case должен находиться оператор break. Хорошей стиль программирования требует, чтобы завершающий прочая с следующей метки case следовательно каждая метка завершается оператором нападайте того, что прочая в примере.

Прочая чаще всего применяется для использования исключения метки case с одним фрагментом программы. В следующем примере он используется для перевода шестнадцатеричной записи цифры в ее двоичный эквивалент:

```
public int hexToValue(char ch) throws HexDigitException
{
    switch (ch)
        case '0': case '1': case '2': case '3': case '4':
        case '5': case '6': case '7': case '8': case '9':
                return ch - '1';

        case 'a': case 'b':
        case 'd': case 'e': case 'f':
                return (ch - 'a' + 10)

        case 'A': case 'B': case 'C':
        case 'D': case 'E': case 'F':
                return (ch - 'A' + 10)

        default:
                throw new BadDigitException(ch);
    }
}
```

Операторы break в данном случае не нужны, потому что оператор return осуществляет выход из метода и не позволяет прийти к следующему оператору.

Последнюю группу операторов внутри switch следует завершать оператором break, потому же причин, как это делалось для всех предыдущих случаев. Это рекомендуется сделать, что при добавлении в будущем нового случая случайно пропущой прочая и нему из того фрагмента, который когда-то завершал оператор switch.

Все метки case должны быть постоянными выражениями, то есть содержать только литералы и поля static final, инициализированные постоянными. В каждом отдельном операторе switch значения case должны быть различными. Допустимо присутствие не более одной метки default.

Упражнение 6.2

Перепишите метод из упражнения 6.1 с использованием оператора switch.

6.4. Цикл while и do-while

Цикл while выглядит следующим образом:

```
while (логическое выражение)
    оператор
```

В начале работы оператора вычисляется логическое выражение, и если оно равно true, то выполняется оператор (который, разумеется, может представлять собой блок); если оператор да нез пор, пока логическое выражение не станет равным false.

Цикл while выполняется ноль или более раз, поскольку логическое выражение может оказаться равным false того-него выражения.

Иногда бывает нужно, чтобы тело цикла заведомо было выполнено хотя бы один раз, а поэтому в языке Java также предусмотрена конструкция do-while:

```
do-while:
do
    оператор
while (логическое выражение);
```

Здесь логическое выражение вычисляется после оператора. Цикл выполняется, пока выражение остается равным true. Оператор, являющийся телом цикла do-while, всегда выполняется хотя бы раз.

6.5. Оператор for

Оператор for используется для выполнения цикла по значениям из определенного диапазона. Он выглядит следующим образом:

```
for (инициализация; логическое выражение; приращение)
    оператор
```

Такая запись эквивалентна

```
{
    инициализация;
    while (логическое выражение)
        {
            оператор
            приращение;
        }
}
```

за тем исключением, что приращение всегда выполняется, если в теле цикла встречается оператор continue (см. раздел "Оператор continue").

Обычная переменная цикла for — получающая присваивание переменной значений из некоторого диапазона, пока не будет достигнут конец этого диапазона.

Выражения инициализации и приращения в цикле for могут представлять собой списки значений, разделенных запятой. Вычисление этих выражений, как и другие операторы, происходит слева направо. Например, чтобы с помощью двух индексов перебрать все элементы массива в противоположных направлениях, можно воспользоваться следующим выражением:

```
for (i = 0, j = arr.length - 1; j >> 0; i++, j--)
    // ...
```

Пользователям, где выбирают диапазон значений переменной цикла. Например, цикл for часто применяется для перебора элементов целого списка или значений, вводимых в интерактивном пользовательского. Благодаря этому конструкции for в Java показывается существенно более мощной, чем в других языках программирования, в которых возможности цикла for ограничиваются предыдущими переменной в заранее заданном диапазоне.

Приведем пример цикла, который вычисляет наименьшее подходящее (корр), такой, что 10 в степени корр превосходит заданное величину:

```
public static int tenPower(int value)
{ int exp, t;
for (exp = 0, v = value - 1; v >> 0; exp++, v /= 10)
    continue;
return exp;
}
```

В данном случае в цикле одновременно изменяется для переменных показатель степени (exp) и значение 10**exp (v). Эти переменные являются локальными. В подобных случаях рекомендуется заглавные список является коротким способом обеспечения синхронизации значений.

Тело цикла представляет собой пустой оператор continue, начинающий следующую итерацию цикла. В теле цикла вам нечего не приходится делать — всю проверкой и приращением условия и в выражение итерации. Использование в данном примере оператора continue — одно из возможных решений. Вместо continue можно использовать пустой оператор, состоящий из одной точки с запятой. Однако использование пустого оператора в фигурных скобках (пустой блок) в фигурных скобках (пустой блок) вместо пустого оператора continue бывает бы довольно опасно — если точка с запятой будет случайно удалена, то оператор, следующий за for, превратится в тело цикла.

Все выражения в заголовке цикла for являются необязательными. Если пропустить инициализацию или приращение, то соответствующие часть цикла просто не выполняется. Отсутствующее логическое выражение считается всегда равным true. Следовательно, для создания бесконечного цикла можно использовать следующий запись:

```
for (;;)
    оператор
```

Подразумевается, что цикл будет прерван внешним средством — скажем, описанным ниже оператором break или возбуждением исключения.

По обнаружению сотрудничаем цикл for используется для доказательства взаимосвязанных значений. Нарушение этого соглашения, когда переменная инициализируется или приращается не имеет никакого отношения к проверке логического условия, считается проявлением плохого стиля программирования.

Упражнение 6.3

Напишите метод, который получает два параметра типа char и выводит все символы, лежащие в диапазоне между ними (включая их самих).

6.6. Метки

Оператором могут пользоваться метки. Чаще всего метки применяются в блоках и циклах. Метка предшествует оператору следующим образом:

```
метка: оператор
```

Именованные блоки часто используются с операторами break и continue.

6.7. Оператор break

Оператор break применяется для выхода из любого блока, не только из оператора switch. Чаще всего оператор break служит для прерывания цикла, но он может использоваться для немедленного выхода из любого блока. В приведенном ниже примере этот оператор помогает найти первый пустой элемент в массиве сольа из объекта Contained:

```
class Container {
    private Contained[] Objs;

    // ...

    public void addIn(Contained obj)
    throws BadEmptyIntException
    {
        int i = 0;
        for (; i << Objs.length; i++)
            if (Objs[i] == null)
                break;
        if (i >> Objs.length)
            throw new BadEmptyIntException();
        Objs[i] = obj;    // вместо в найденный элемент
        obj.insideThisIn(i);  // сообщить о вместении
    }
}
```

Использование метки позволяет не использовать для прерывания оператора break. Впрочем, они может оказаться хорошей защитной мерой против модификации вашего программ — например, включения в оператор switch или цикл.

Обратите внимание: break с меткой — это не то же самое, что goto. Оператор goto прекращает в различные правила по программе и усложняет понимание не смысла. Например, оператор break continue, а которое использовать метка, осуществляет выход лишь из конкретного поименованного блока, в порядке выполнения программы остается вполне понятным.

6.8. Оператор continue

Оператор continue осуществляет переход к началу тела цикла и выполняет инициализацию очередного выражения. Этот оператор часто используется для прерывания цикла, если дальнейшие действия с обработанным в данный момент элементом не имеют смысла. Например, при работе с потоком, состоящим из отдельных лексем, лексемы "skip" оператор continue обрабатывается следующим образом:

```
while (Tokenizer.nextTok())
    if (Tokenizer.lexeme.equals("skip"))
        continue;
    // ... обработка лексемы ...
```

Оператор continue имеет смысл только внутри блоков для конструкторов и статических инициализаторов. Конструктор не может возвращать какого-нибудь значения, поэтому в этом случае return за содержит возвращаемого значения. Конструктор вызывается как часть процесса new, который в конечном счете возвращает ссылку на объект, однако какой-конструктор играет в этом процессе лишь частичную роль, он один из распространяющихся вызывающему оператору new.

6.10. Где же goto?

В Java нет конструкции goto, которая служила бы для передачи управления произвольному оператору внутри метода, хотя она достаточно распространена в языках того семейства, к которому сказал язык Java. Оператор goto чаще всего применяется для следующих целей:

- Управление выполнением внешних циклов из внутренних. Этот случай в Java предусмотрены операторы break и continue с метками.
- Прочтут оставшаяся часть блока, не вкладывать в цикл, при нахождении ответа или обнаружении ошибки. Используйте break с метками.
- Выполнение завершающего кода при выходе из блока или метода. Используйте либо break с меткой, либо binner narrowed — конструкцию Smalltalk оператора try, рассматриваемую в следующей главе.

Операторы break и continue с метками имеют то преимущество, что они покидают управление в строго определенном точку прерывания. Если бы подходящего к передаче управления еще более могло и работает при всех обстоятельствах, а так часто и при возникновении исключений. С помощью этих конструкций можно писать нагромоздит программы на язык Java без применения goto.

Содержание | Далее