Ответы на вопросы экзамена по курсу «Языки программирования» 09.01.2016

В ответах курсивом выделены необязательные пояснения, которые можно опустить (особенно на экзамене)

Вариант 1

Задача 1-1

Объясните, что означает понятие «виртуальное множественное наследование» в языке C++. Объясните, какие проблемы возникают при реализации этого понятия (не перечисляйте общих для множественного наследования проблем, а ограничьтесь только специфическими проблемами виртуального наследования)?

Ответ

Единичное виртуальное наследование — это наследование виртуальной базы, которое происходит с помощью ключевого слова virtual:

```
class VBase { … };
class Der : public virtual VBase {...};
```

Множественное виртуальное наследование – это наследование нескольких виртуальных баз. Особенностью виртуальной базы является то, что при виртуальном множественном наследовании одинаковые виртуальные базы объединяются а один экземпляр базы. Тем самым реализуется так называемое ромбовидное (diamond) наследование.

Здесь должен быть рисунок ромба ABCD…

На лекциях мы разбирали одну специфичную для виртуального наследования проблему, а именно, организацию доступа к виртуальной базе при виртуальном множественном наследовании

Проблема состоит в том, что функции наследника виртуальной базы Der должны транслироваться СПЕЦИАЛЬНЫМ образом при единичном виртуальном наследовании. *Множественного наследования еще в помине нет, а его уже нужно учитывать…*

Специфика проявляется при доступе функций класса Der к унаследованным членам-данным из виртуальной базы. Проблема в том, что при трансляции функций-членов класса Der реальное размещение (и, следовательно, смещения) членов-данных виртуальной базы `VBase` неизвестны.

Так при размещении переменной

Der x;

виртуальная база `VBase` может находиться непосредственно перед частью от Der. А при наследовании:

```
class Der1 : public virtual VBase {...};
class W : public Der1, public Der {...};
```

при размещении переменной

W y;

получается такая ситуация: пусть виртуальная база размещеается сразу перед частью от Der1, тогда между виртуальной базой и частью от Der в переменной y находится промежуток (то, что унаследовано от Der1). А функции из класса Der должны быть оттранслированы так (без всякого знания о деталях будущего множественного наследования), чтобы работать с виртуальной базой и для this=&x и для this=&y.

Хотя относительные смещения от виртуальной базы до части Der в этих случаях совершенно разные. И это, конечно, проблема, которую надо решать.

На этом можно и закончить — ведь проблема объяснена. Но можно (хотя и необязательно) добавить и про возможные решения проблемы. Стандартный подход состоит в том, что при единичном виртуальном наследовании добавляется еще одно скрытое поле — указатель (ссылка) на виртуальную базу. Это указатель определяется при размещении конкретной переменной (инициализируется, конечно, системной частью конструктора). Смещение этого поля от указателя this всегда фиксировано и не меняется при множественном наследовании. Функции производного класса никогда не работают с виртуальной базой напрямую, а только косвенно — через эту ссылку. Конечно, это требует некоторых дополнительных накладных расходов.

Задача 1-2

Объясните, что означают термины «семантика возобновления» и «семантика завершения». Приведите примеры языков, в которых реализована семантика завершения. Опишите пример моделирования семантики возобновления через семантику завершения.

Ответ

Семантика возобновления: после обработки исключения управление может вернуться непосредственно в точку, где возникло исключение (варианты: на следующий оператор или на любой оператор из того же блока).
Семантика завершения: после возникновения исключения блок, в котором оно возникло, обязательно завершается. Обработка исключения происходит в блоках, вызвавших блок с исключением.
Примеры языков*:* C++, Ада, Java, C#, Python, JavaScript.
в некоторых случаях семантика возобновления может быть смоделирована, например, в случае выделения возобновляемого ресурса (типа динамической памяти):

```
Resource GetResource() {
  for (;;)
    try {
      Resource r = … // попытка получить ресурс, например
                     // выделить память
      if (success) return r;
      throw NoResourceException();
    } catch (NoResourceException) {
      // попытка найти дополнительные ресурсы (например,
      // динамически собрать мусор)
      if (!success) throw;
    }
}
```

Задача 1-3

Дайте определение языковой конструкции «интерфейс». Что означает термин «интеграция интерфейсов в язык программирования»? Приведите два различных примера интерфейсов, интегрированных с языком Java.

Ответ

Интерфейс можно рассматривать как абстрактный класс, доведенный до «абсолюта». Интерфейс состоит только из публичных абстрактных методов. Поскольку реализации методов нет (равно как нет и невиртуальных методов), то интерфейс не имеет нестатических членов (статические члены, например, константы допустимы). В объявлении интерфейса присутствуют только сигнатуры методов (а также свойств в языках, где есть это понятие), и, возможно, статических членов и вложенных интерфейсов. Интерфейс представляет собой «чистый» контракт. Производный класс, наследуя интерфейс, «подписывается» под контрактом. Производный класс, наследующий интерфейс и замещающий все его методы, называют реализацией интерфейса.
Интеграция интерфейсов в язык означает, что язык (компилятор) поддерживает ряд стандартных интерфейсов, позволяющих интегрировать семантику языковых конструкций и пользовательские классы, реализующие эти интерфейсы. Например, в языке Java стандартный интерфейс `Iterable` используется в цикле `for-each` для прохода по коллекциям. Если пользовательский класс реализует этот интерфейс, то он тоже может использоваться в цикле `for-each`, как это можно делать со стандартными массивами и коллекциями.
Другой пример — интерфейс `Cloneable`. Если класс реализует интерфейс-маркер `Cloneable`, то это означает, что класс будет реализовывать открытый метод `clone()` создания своей копии:

```
class CloneAttack implements Cloneable
{
          public Object clone()
          {
               // возвращает свою копию
          }
}
```

Тут следует отметить, что любой класс уже содержит версию метода `clone()`, унаследованную от класса `Object`. Этот метод возвращает поверхностную копию объекта. Однако проблема в том, что этот метод - защищенный, поэтому может использоваться только из производных классов или из классов своего пакета. Для того, чтобы позволить копирование себя внешним классам и используется интерфейс `Cloneable`.

Задача 1-4

Дайте определение «пространства имен» в языке C# и пакета в языке Java. В чем состоит основное сходство и основное отличие этих понятий?

Ответ

Задача 1-5

Что будет напечатано в результате работы следующей программы на языке Python?

```
s= range(1,5)
```

```
sqr = True
processFunc = sqr and (lambda s: s*s) or (lambda s: s)
print([processFunc(i) for i in s])
```

Что будет напечатано, если `sqr = True` заменить на `sqr = False`?

Ответ

Будет напечатан список квадратов чисел от 1 до 4: [1, 4, 9, 16]. Если `sqr = True` заменить на `sqr = False`, то список самих чисел [1, 2, 3, 4]. *Вспомним, что функция range работает с полуоткрытым диапазоном и не включает правую границу в диапазон. Особенность выполнения логических операций в Python состоит в том, что результат логической операции — это всегда значение одного из операндов. В случае, если значения всех операндов — логические, то, конечно, результат тоже будет логическим (True или False). Если операнд не логический, то его значения `0`, `''`, `[]`, `()`, `{}` и `None` интерпретируются как ложь, все остальные значения — как истина. При этом результатом выражения является не интерпретация значения соотвествующего операнда, а его значение. Поэтому результатом операции «`and`» в операторе выше будет либо False (в случае, если первый операнд - False), либо значение ее второго операнда БЕЗ ПРЕОБРАЗОВАНИЯ ТИПА (Python вообще не любит преобразования) — то есть лямбда-функция возведения в квадрат. Аналогично результатом операции «`or`» в операторе выше будет либо лямбда-функция возведения в квадрат (если `sqr` есть True), либо (если значение `sqr - False`) значение второго операнда также без преобразования типа, то есть тождественная лямбда-функция. Далее эта функция применяется для генерации списка квадратов (sqr – True), либо списка самих чисел (sqr – False).*

Задача 1-6

Переписать программу из задачи №5 на языке C# с использованием механизма лямбда-функций и обобщенных делегатов так, чтобы она выдавала в стандартный вывод те же самые значения.

Ответ

Здесь требуется написать эквивалентный фрагмент программы, который делает то же самое. Особенность этого искусственного примера в том, что он демонстрирует технику, которая позволяет экономить на проверке условия sqr – True или False для каждого элемента списка (в C# аналогом питоньего списка будет, конечно, массив). То есть мы должны один раз сконфигурировать функцию (значение делегатского типа в C#) в зависимости от значения sqr, а затем вызывать эту функцию для каждого элемента массива без лишней проверки sqr. В случае Питона — это просто (можно было бы еще проще, но здесь это опустим), однако в случае C# (статически типизированного языка), лямбда-выражение само не является делегатом, а должно рассматриваться как «генератор» делегатов (функций) в зависимости от настройки типа аргументов лямбда-выражения. Поэтому очевидное, на первый взгляд, присваивание

var procFunc = x=>x;*

выдает ошибку компиляции: левая часть требует, чтобы ее тип определялся из правой части, а правая часть — лямбда-выражение — само требует информации из левой части о типе для настройки на конкретную функцию-делегат.

Так что надо вместо var объявить нужный делегатский тип. Для этого-то очень удобны обобщенные делегаты: нам нужен, очевидно, делегат — функция от int, возвращающая int. В пространстве имен System уже любезно заготовлен нужный обобщенный делегат -

Func<TRes, Targ>. Нужная нам конкретизация – Func<int,int>. Поэтому имеем (опуская антураж полной программы, а также скобки [и]):

```
int [] s = {1,2,3,4};
var sqr = true;
Func<int,int> procFunc;
if (sqr)
    procFunc = x=>x*x;
else
    procFunc = x=>x;
foreach (var i in s)
    System.Console.Write(procFunc(i).ToString() + ", ");
```

Обязательно ли использовать обобщенный делегат? Конечно, нет, точно также, как необязательно использовать лямбды — можно описать две функции (тут уже антураж программы не опустишь), далее вместо лямбда-выражений использовать имена этих функций. Точно также, вместо обобщенного делегата можно использовать конкретный делегат Func, описав его как delegate int Func(int x); там же, где описывались функции,ио есть в районе метода main в главном классе программы и т. д. Поэтому в задании требуется использовать лямбда-выражения и обобщенные делегаты, хотя я не наказывал за использование конкретного делегата. Лямбда-функции и выражения в процедурном программировании — это вообще вопрос удобства, можно и без них, но с ними — компактнее и нагляднее во многих случаях.

Задача 1-7

Рассмотрим файл на языке Java, содержащий определение класса C:

```
package test;
class C { public void f() { g(); } }
```

При трансляции этого файла выдается ошибка. Объясните, в чем она состоит, и что нужно добавить в программу, не меняя ничего в классе C, чтобы данный файл транслировался без ошибок? Можно добавлять в программу новые файлы, а также новые строки в заданный файл, но нельзя менять определение класса C.

Ответ

Ошибка состоит в вызове неопределенной функции `g();`
Значит, надо сделать так, чтобы эта функция была определена и видима (в Java – управление видимостью!) в классе C. Не меняя объявление класса C это можно сделать двумя способами: включить класс C в качестве внутренноего класса в какой-нибудь объемлющий класс:

```
package test;

class Outer{

      void g() { ; }

      class C { public void f() { g(); } }

...

}
```

либо использовать статический импорт *(именно это и предполагалось автором)*:

```
package test2;
class Extern
{
  public void g() { ; }
}
package test;
```

```
import static test2.Extern.g; // либо import static test2.Extern.*;
class C { public void f() { g(); } }
```

Можно и еще проще — вообще без дополнительного пакета.

Задача 1-8

Что будет напечатано в результате работы следующей программы на Java? Считаем, что каждый пакет находится в своем файле.

```
package P2;
class CC extends P1.AC {
      public void f1() { print("CC.f1");}
      public void f2() { print("CC.f2");}
      public void f3() { print("CC.f3");}
      public void f4() { print("CC.f4");}
}
public class TT {
      public static void main(String [] args) { new CC().show(); }
}
package P1;
public abstract class AC {
      public static final void print(String s) { System.out.println(s);}
      public void f1() { print("AC.f1");}
      void f2() { print("AC.f2");}
      protected void f3() { print("AC.f3");}
      private void f4() { print("AC.f4");}
      public final void show() { f1();f2();f3();f4();}
}
```

Ответ

CC.f1
AC.f2
CC.f3
AC.f4

Напомним, что в Java – управление видимостью, а не доступом. Это означает, в частности, что замещаться могут только ВИДИМЫЕ методы. Приватные методы, поэтому замещаться не могут ни в каком производном классе (поэтому AC.f4), а в другом пакете невидимы (и не могут замещаться методы с пакетным доступом, поэтому AC.f2). В другом варианте наследование происходит в том же пакете, поэтому метод f2 видим для замещения, и поэтому там будет выдано CC.f2