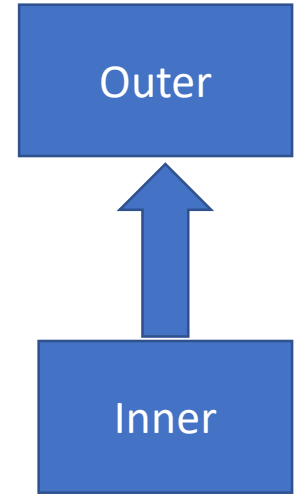
Внутренние классы в Java

```
class Outer {
        int _number;
        public void f() { ..... }
        public class Inner {this
                void foo(int i) {
                // можно ссылаться на члены внешнего класса
                _number = i;
                // общий синтаксис: this._number и Outer.this._number
                }
        }
}
```



как создавать? Нужна ссылка на Outer-объект

Outer o; .... Outer.Inner inn = o.new Inner();

Внутри Outer: this.new Inner()

Inner-объекты имеют ссылку на объект объемлющего класса, а Outer-объекты - нет

Подпрограммные типы данных

Внутренние (+ локальные + анонимные) классы ОЧЕНЬ близко подходят к концепции замыкания

Замыкание - это функция, которая использует в своем теле ссылки на свободные переменные (то есть переменные из контекста).

Нет своб. переменных => замыкание - просто функция.

Замыкание имеет смысл, только если в ЯП есть ФТД (то есть функции - это значения) - иначе говоря - функции первого порядка, а также локальные функции, которые могут использовать контекст.

Связанные термины - функции первого порядка, функции высшего порядка.

Замыкание = функция + "захваченный контекст"

Важный частный случай для локальных функций и замыканий - анонимные функции (лямбда-функции).

ФТД и замыкания в ОИЯП (Java)

Java - главная проблема - отсутствие ФТД (в C++, C# уже были)

ЗАТО: идея замыкания уже реализована

Java 2014 (Java 8):

как реализовать понятие ФТД в языке, где нет понятия функции, а есть только методы?

"Language Hack"

Начиная с Java 2005: есть обобщенные классы и обобщенные интерфейсы

$\Rightarrow$в Java 8:

ФТД === интерфейс с единственным методом

Значения - анонимные функции и ссылки на методы класса

ФТД и замыкания в ОИЯП (Java)

Java - интерфейсы

```
interface ISomething {
        void foo();
        int bar(int i);
}
class CoClass implements ISomething {
        public void foo() {..}
        public int bar(int i) {..}
}

CoClass c = new CoClass();
ISomething f = c;
// интерфейсы наследуемы множественным образом,
// так как наследуется таблица виртуальных методов
f.foo(); f.bar(-1);
```

ФТД и замыкания в ОИЯП (Java)

Java - функциональный интерфейс == интерфейс, в котором есть только одна функция

// @FunctionalInterface - необязательно, но если добавить, то компилятор будет ругаться при попытке добавить второй метод в такой интерфейс

```
@FunctionalInterface
interface IFoo {
        int bar(int i);
}
```

А это и есть ФТД!!! А где его значения?

ФТД и замыкания в ОИЯП (Java)

```
@FunctionalInterface
interface IFoo {
        int bar(int i);
}
```

А это и есть ФТД!!! А где его значения?

- Ссылка на метод класса
- Анонимная функция

```
IFoo f = ClsName::method_name
IFoo g = (int x)->{return x+1;};
f.bar(0); f.bar(-1);
```

ФТД и замыкания в ОИЯП (Java)

Java - большой набор стандартных обобщенных интерфейсов:

```
interface Function<T, R> {
		R apply(T x);
}
interface BiFunction<T1, T2, R> {
		R apply(T1 x, T2 y);
}

interface UnaryOperator<T> {
		T apply(T x);
}
interface BinaryOperator<T> {
		T apply(Tx, Ty);
}
// в Java интерфейсы не перегружаются!
interface Predicate<T> {
		bool test(T x);
}

interface Consumer<T> {
		void accept(T x);
}
interface Supplier<T> {
		T get();
}

interface Compare<T> {
		bool compare(T x, Ty);
}
```

ФТД и замыкания в ОИЯП (Java)

Можно и не писать свои интерфейсы...

```
Function<T, R> f = (T x)->{return r;// r типа R};
Function<Integer, String> f1 =
    new Function<Integer, String> {
            String apply(Integer i) { return String.valueOf(i); }
    };
// анонимный внутренний класс
// можно записать и в виде лямбды, будет короче
Function<Integer, String> f2 = (Integer i)->{return String.valueOf(i);};
// можно и методы
Function<Integer, String> f3 = String::valueOf;
f3.apply(2);
```

ФТД и замыкания в ОИЯП (Java)

Java. Ссылки на методы

```
// ссылки на статические методы
Function<T, R> f1 = Class::method;
// method должен удовлетворять сигнатуре f.apply
// ссылки на нестатические методы
BiFunction<Class, T, R> f2 = Class::method; // method - нестатическая функция T->R
Class c = new Class();
R r = f2.apply(c, t);
// ссылка первого рода (самим надо указывать this)
Function<T, R> f3 = c::method; // c - объект класса, даст ссылку на this, метод нестатический
R r = f3.apply(t);
// ссылка на конструктор
Class::new; // может удовлетворять практически любому интерфейсу.
        // если конструктор умолчания, то удовлетворяет Supplier
```

ФТД и замыкания в ОИЯП (Java)

```
Locale lRu = f.apply("ru", "RU");
Locale lEn = f.apply("en", "UK");
// варианты f
// 1
BiFunction<String, String, Locale> f = new BiFunction<String, String, Locale> {
        Locale apply(String lang, String country) {
                return new Locale(lang, country);
        }
}; // обертка вокруг конструктора
// 2
BiFunction<String, String, Locale> f = (lang, country)->new Locale(lang, country);
// 3
BiFunction<String, String, Locale> f = Locale::new;
```

ФТД и замыкания в ОИЯП (Java)

Java Stream API

Контейнер => поток

List<Integer> list = new List<Integer>();

list.stream() - это поток - "конвейер" с методами, аргументы которых - функции обработки элементов

- терминальные - max(), min(), count()

  list.stream().max((i,j)->i > j)

  list.stream().count((i)->i >=0)

  list.stream().sorted()

- промежуточные (конвейерные) - преобразуют последовательность ленивым образом

  list.stream(). filter(x->x>=0). .....

  list.stream().map(x->x * x). ....

  list.stream() .filter(x->x>0).map(x->sqrt(x)).peek(System.out::println)

ФТД и замыкания в ОИЯП (Java)

Потоки языка Java (не путать с потоками ввода-вывода и потоками управления) - Streams - Threads

Частный случай терминальных методов - редуцирующие:

list.stream().sum()

list.stream().reduce((x,y)->x * y)

Что они возвращают для Stream<T>?

Optional<T>

или list.stream().reduce(0, (x,y)->x * y)

T reduce(T identity, BinaryOperator<T> accum)

Тут уже без вариантов - T

ФТД и замыкания в ОИЯП (Java)

Потоки языка Java (не путать с потоками ввода-вывода и потоками управления) - Streams - Threads

```
list.stream() .filter(x->x>=0).map(x->sqrt(x)).peek(System.out::println)
```

Альтернатива (старый стиль)

```
for (int x: list) {
    if (x >=0) {
        x = sqrt(x);
        System.printl(x);
    }
}
```

Но:

```
int r = list.stream()
    .parallel().filter(x->x>=0).map(x->sqrt(x))
    .sequential().reduce(0, (x,y)->x + y);
```

ФТД и замыкания в ОИЯП (C#)

C# - ФТД уже присутствовали в языке

В .Net есть классы (System.Delegate, System.MulticastDelegate), которые НЕЛЬЗЯ использовать прямо в коде, но которые используются компилятором для реализации т.н. "делегатов"

Общее определение делегата (как концепции): класс (объект класса), который содержит внутри себя ссылку на объект другого класса, и реализует свои методы (операции) путем вызова ("делегирования") метода этого объекта.

То есть переадресация операций делегата к операциям другого класса.

В ООП: делегирование - альтернативный путь реализации наследования.

ФТД и замыкания в ОИЯП (C#)

C# - ФТД уже присутствовали в языке (только так не назывались....)

```
class Y {
        public void g() {..}
        public static void gg() {..}
}
class X {
        delegate void PVF(); // фактически - объявление ФТД
        void foo() {..}
        static void bar() {..}
        PFV cd; // выглядит как объявление переменной типа PVF
        void xxx() {
                cb = new pvf(foo);  // можно и так: cb = new pvf(this.foo);
                Y yy;
                cb += new pvf(bar);  // можно и так: cb += new pvf(X.bar);
                cb += new pvf(yy.gg);
                cb += new pvf(Y.g);
                cb(); // все зарегистрированные делегаты будут вызваны
        }
}
```

ФТД и замыкания в ОИЯП (C#)

Можно и забыть про new (но помнить надо....)

```
class X {
        delegate void PVF(); // фактически - объявление ФТД
        PFV cd; // выглядит как объявление переменной типа PVF
        void xxx() {
                cb = foo;  // можно и так: cb = this.foo;
                Y yy;
                cb += bar;  // можно и так: cb += X.bar;
                cb += yy.gg;
                cb += Y.g;
                cb(); // все зарегистрированные делегаты будут вызваны
        }
}
```

ФТД и замыкания в ОИЯП (C#)

delegate int IntFi (int);

Фактически на делегаты C# можно рассматривать как референциальные ТД

- значения - МЕТОДЫ с заданной сигнатурой (int (int)) - this НЕ ВХОДИТ в сигнатуру
- операции (a - переменная-делегат) :
    - a = b (b - значение или переменная-делегат)
    - d1 != d2, d1 == d2
    - a += b
    - a -= b
    - a(5) // из-за этого все и затевалось
    - А также все операции из класса System.MulticastDelegate...

Все предыдущие операции - это "СинтСахар" над этими операциями

https://docs.microsoft.com/en-us/dotnet/api/system.multicastdelegate?view=net-5.0

ФТД и замыкания в ОИЯП (C#)

C#:

Операции (a - переменная-делегат) :

a = b (b - значение или переменная-делегат)

a += b

a -= b

a(5) // из-за этого все и затевалось

Реализация модели "подписка-рассылка":

a += b // подписка

a -= b // отказ от подписки

a(5) // рассылка - все "подписанты" вызываются

Независимость от того, статический метод, или нет!

ФТД и замыкания в ОИЯП (C#)

C#: реализация модели "подписка-рассылка":

a += b // подписка

a -= b // отказ от подписки

a(msg) // рассылка - все "подписанты" вызываются (то есть оповещаются о приходе сообщения

В чем проблема?

ФТД и замыкания в ОИЯП (C#)

C#: реализация модели "подписка-рассылка":

a += b // подписка

a -= b // отказ от подписки

a(msg) // рассылка - все "подписанты" вызываются (то есть оповещаются о приходе сообщения

В чем проблема?

Кто может "рассылать" сообщения?

Любой (.... подписант).

Классические ТД: если объект доступен, то доступны все публичные операции (см. выше).

В модели "подписка-рассылка" операции a += b , a -= b доступны всем, а рассылка - a(arg_list) - только "владельцу".

Кто "владелец"? (альтернативы - ????)

ФТД и замыкания в ОИЯП (C#)

C#: реализация модели "подписка-рассылка":

a += b // подписка

a -= b // отказ от подписки

a(msg) // рассылка - все "подписанты" вызываются (то есть оповещаются о приходе сообщения

В чем проблема?

Кто может "рассылать" сообщения?

Любой (.... подписант).

Классические ТД: если объект доступен, то доступны все публичные операции (см. выше).

В модели "подписка-рассылка" операции a += b , a -= b доступны всем, а рассылка - a(arg_list) - только "владельцу".

"Владелец" - класс, в котором описана переменная a (не тип!)

ФТД и замыкания в ОИЯП (C#)

C#: реализация модели "подписка-рассылка":
События:
Где-то определен и доступен тип делегата PVF....

```
class EventSource {
        public event System.Action myEvent;
        private void FireSomeEvent () { ... myEvent(); ..... }
        .....
}
// к экземпляру myEv : из функций класса EventSource применимы //все операции (из список см. выше),
извне (из любого другого класса) только += и -= (подписка-отписка)
class X {
        static void Panic() { System.Console.WriteLine("Караул!!!");}
        void NoProblem() {  SaveMyInfo(this);  }
}
X x = new X();
EventSource es = new EventSource();
es.myEvent += Panic;
es.myEvent += x.NoProblem;
es.myEvent(); // ошибка
```

ФТД и замыкания в ОИЯП (C#)

C#: реализация модели "подписка-рассылка":
События:
Где-то определен и доступен тип делегата PVF....

```
class EventSource {
        public event Action myEvent;
        private void FireSomeEvent () { ... myEvent(); ..... }
        .....
}
```

// к экземпляру myEv : из функций класса EventSource применимы //все операции (из список см. выше), извне (из любого другого класса) только += и -= (подписка-отписка)

```
class X {
        static void Panic() { System.Console.WriteLine("Караул!!!");}
        void NoProblem() {  SaveMyInfo(this);  }
}
X x = new X();
EventSource es = new EventSource();
es.myEvent += Panic; // // это уже не Combine(....)
es.myEvent += x.NoProblem; // это уже не Combine(....)
```

Изнутри:
для каждого события генерируются (или могут быть указаны явно) два метода:

```
public event Action myEvent {
        add() { .... value имеет тип PVF .....} // вызывается для +=
        remove() { ... value имеет тип PVF....} // вызывается для -=
}
```

На что похоже? - на свойства (get/set).
Поэтому - свойства и события - могут быть членами интерфейсов (а переменные - данные "обычных" классов - нет)

ФТД и замыкания в ОИЯП (C#)

C# 1.0 (1999): ФТД - уже есть (не совсем в стиле C/C++)
C# 2.0 (2005):
- обобщенные делегаты (уже не нужно писать свои объявления типа PVF, IntFi...)
- анонимные делегаты

System.Func<T,R> (int f(double x))
System.Func<T1,T2,R> (double foo(int x, bool b))
.....
System.Action (уже был - аналог PROC в Обероне, M-2....)
System.Action<T>
System.Action<T1,T2>
.....

ФТД и замыкания в ОИЯП (C#)

C# 2.0 (2002):

- обобщенные делегаты (уже не нужно писать свои объявления типа PVF, IntFi...)

System.Func<T,R> (int f(double x))

System.Func<T1,T2,R> (double foo(int x, bool b))

.....

System.Action (уже был - аналог PROC в Обероне, М-2....)

System.Action<T>

System.Action<T1,T2>

.....

System.EventHandler<T> (T - наследник типа System.EventArg)

ФТД и замыкания в ОИЯП (C#)

C# 2.0 (2005):

- анонимные делегаты

```
static int freeze1(Func<int, int> f, int arg) { return f(arg); }
static int add1(int x) { return x + 1; }
freeze1 (add1, 10);
```

либо:

```
freeze1 (delegate(int x) { return x + 1;} , 10);
```

Является ли это все функциями высших порядков?

Еще нет....

ФТД и замыкания в ОИЯП (C#)

C# 2.0 (2005):

- анонимные делегаты

```
static Func<int>  Freeze1(Func<int, int> f, int arg) {
        return delegate () { return f(arg);}
}
```

А это уже и есть замыкание....

Про подписки и рассылки уже забыли....

ФТД и замыкания в ОИЯП (C#)

C# 3.0 (2008):

лямбда-выражения и лямбда-операторы:

x => x+1

(x,y) => x + y

(x, y) => { return x + y;} // лямбда-оператор

```
 Func<int,int> f = (x) => x + 1;
Func<Func<int, int>, int, Func<int>> ff = (funct,arg) => ()=>funct(arg);
```

ФТД и замыкания в ОИЯП (C#)

```
Func<int,int> f = (x) => x + 1;
Func<int, double> foo = (x) => x + 1;
```

ФТД и замыкания в ОИЯП (C#)

```
Func<int,int> f = (x) => x + 1;
Func<int, double> foo = (x) => x + 1;
Func<Func<int, int>, int, Func<int>> ff = (funct,arg) => ()=>funct(arg);
```

ФТД и замыкания в ОИЯП (C#)

Func<int,int> f = (x) => x + 1;

Func<int, double> foo = (x) => x + 1;

Func<*Func<int, int>, int, Func<int>*> ff = что это?

ФТД и замыкания в ОИЯП (C#)

Func<int,int> f = (x) => x + 1;

Func<int, double> foo = (x) => x + 1;

Func<*Func<int, int>, int, Func<int>*> ff = что это?

Что-то вроде:

delegate(Some arg1, int arg2) { return delegate() { return arg1(arg2);}

Где Some - это Func<int,int>

ФТД и замыкания в ОИЯП (C#)

Func<int,int> f = (x) => x + 1;

Func<int, double> foo = (x) => x + 1;

Func<*Func<int, int>, int, Func<int>*> ff = что это?

Что-то вроде:

delegate(Some arg1, int arg2) { return delegate() { return arg1(arg2);}

Где Some - это Func<int,int>

```
 Func<Func<int, int>, int, Func<int>> ff =
        (funct,arg) => ()=>funct(arg);
```

ФТД и замыкания в ОИЯП (C#)

Func<int,int> f = (x) => x + 1;

Func<int, double> foo = (x) => x + 1;

Func<*Func<int, int>, int, Func<int>*> ff = что это?

Что-то вроде:

delegate(Some arg1, int arg2) { return delegate() { return arg1(arg2);}

Где Some - это Func<int,int>

```
 Func<Func<int, int>, int, Func<int>> ff =
        (funct,arg) => ()=>funct(arg);
```

ФТД и замыкания в ОИЯП (C#)

Func<int,int> f = (x) => x + 1;

Func<int, double> foo = (x) => x + 1;

Func<*Func<int, int>, int, Func<int>*> ff = что это?

Что-то вроде:

delegate(Some arg1, int arg2) { return delegate() { return arg1(arg2);}

Где Some - это Func<int,int>

```
 Func<Func<int, int>, int, Func<int>> ff =
        (funct,arg) => ()=>funct(arg);
```

ФТД и замыкания в ОИЯП (C#)

Аналоги потоков (Stream API) (достигается за счет применения так называемых "расширяющих методов" к объектам классов, реализующих интерфейсы IEnumerable и IQueryable):

List<int> list = new List<int>(); ....

list.Where(x=>x>=0).Select(x=>sqrt(x)).OrderBy((x,y)=>x > y)

list.Aggregate(0, (total,next)=>total * next) // сравните с Java-reduce

list.Where(x=>x%2==0).Max()

Что возвращает? Nullable<int>