Язык программирования Java

Мультимедийный курс

автор: Васильев А.Н.

www.vasilev.kiev.ua

Киев 2017

Функциональные интерфейсы, лямбда-выражения, ссылки на методы и анонимные классы

- Анонимные классы
- Синтаксис лямбда-выражения
- Функциональные интерфейсы
- Использование лямбда-выражений
- Ссылка на метод объекта
- Ссылка на нестатический метод класса
- Ссылка на статический метод
- Ссылка на конструктор

Анонимные классы

Достаточно часто возникает необходимость в создании класса, который используется в программе один-единственный раз. В таких случаях нередко используют анонимные классы - классы, у которых нет "собственного имени"

Анонимный класс можно создать путем наследования класса (обычно абстрактного) или реализации интерфейса

Анонимный класс на основе абстрактного суперкласса

```
Суперкласс переменная=new Суперкласс(аргументы){
    // Описание методов
};
```

Анонимный класс на основе интерфейса

```
Интерфейс переменная=new Интерфейс(){
    // Описание методов
};
```

Анонимный класс на основе абстрактного суперкласса

Анонимный класс на основе абстрактного суперкласса - 1

```
// Абстрактный суперкласс:
abstract class Base{
   // Текстовое поле:
   private String name;
   // Конструктор:
   Base(String txt){
      name=txt;
   }
   // Метод для отображения значения текстового поля:
   void show(){
      System.out.println("Имя объекта: "+name);
   }
   // Объявление абстрактного метода:
   abstract void hello();
} // Главный класс:
class AnonymousClassDemo{
   public static void main(String[] args){
      // Создание объекта анонимного класса:
      Base obj=new Base("Красный"){
         // Описание абстрактного метода из суперкласса:
         void hello(){
            System.out.println("Объект анонимного класса");
         }
      }; // Завершение инструкции создания объекта анонимного класса
         // Продолжение на следующем слайде!!!
```

Анонимный класс на основе абстрактного суперкласса

Анонимный класс на основе абстрактного суперкласса - 2

```
        // Вызов методов из объекта, созданного на основе
        // анонимного класса:
        obj.show();
        obj.hello();
        // Создание анонимного объекта анонимного класса
        // и вызов их этого объекта метода showAll(),
        // описанного в анонимном классе:
        new Base("Зеленый"){
            // Описание абстрактного метода из суперкласса:
            void hello(){
                System.out.println("Анонимный объект");
            }
            // Описание нового метода:
            void showAll(){
                hello();
                show();
            }
        }.showAll(); // Вызов метода
    }
}
```

Результат

```
Имя объекта: Красный
Объект анонимного класса
Анонимный объект
Имя объекта: Зеленый
```

Анонимный класс на основе интерфейса

Анонимный класс на основе интерфейса - 1

```
// Интерфейс:
interface Base{
   // Метод с кодом по умолчанию:
   default void show(){
      System.out.println("Интерфейс Base");
   }
   // Объявление метода:
   void hello();
}
// Главный класс:
class MoreAnonymousClassDemo{
   public static void main(String[] args){
      // Создание объекта анонимного класса:
      Base obj=new Base(){
         // Описание метода из интерфейса:
         public void hello(){
            System.out.println("Объект анонимного класса");
         }
      }; // Завершение инструкции создания объекта
         // анонимного класса
      // Вызов методов из объекта, созданного на основе
      // анонимного класса:
      obj.show();
      obj.hello();
      // Продолжение на следующем слайде!!!
```

Анонимный класс на основе интерфейса

Анонимный класс на основе интерфейса - 2

```
        // Создание анонимного объекта анонимного класса
        // и вызов их этого объекта метода showAll(),
        // описанного в анонимном классе:
        new Base(){
            // Описание метода из интерфейса:
            public void hello(){
                System.out.println("Анонимный объект");
            }
            // Описание нового метода:
            void showAll(){
                hello();
                show();
            }
        }.showAll(); // Вызов метода
    }
}
```

Результат

```
Интерфейс Base
Объект анонимного класса
Анонимный объект
Интерфейс Base
```

Синтаксис лямбда-выражения

- Лямбда-выражение можно рассматривать как некоторый блок кода, определяющий действие
- Также можно рассматривать лямбда-выражение как определение анонимного метода - то есть метода, у которого нет имени (аналогия не совсем точная, то во многих случаях помогает понять логику происходящего)
- Используются лямбда-выражения в основном как альтернатива к созданию анонимных классов

Шаблон описания лямбда-выражения

```
(аргументы)->{команды}
```

Пример описания лямбда-выражения

```
(int x,int y)->{int z=x+y; System.out.println(z);}
```

Пример описания лямбда-выражения

```
(int x,int y)->{
    int z=x+y;
    System.out.println(z);
}
```

Упрощение синтаксиса лямбда-выражений

Есть правила, которые позволяют упрощать синтаксис описания лямбда-выражений:

- Тип аргумента можно не указывать, если он быть идентифицируется исходя из контекста команды использования лямбда-выражения

- Если в лямбда-выражении один аргумент, тип которого не указан, то круглые скобки можно не использовать

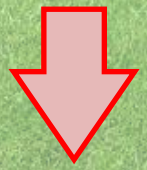
- Если в лямбда-выражении нет аргументов, то используются пустые круглые скобки

- Если в теле лямбда-выражения всего одна команда, то фигурные скобки можно не использовать. В случае, когда единственная команда в теле лямбда-оператора состоит из `return`-инструкции, ключевое слово `return` можно не использовать

Функциональный интерфейс и лямбда-выражения

Функциональный интерфейс - это интерфейс с одним и только одним абстрактным методам

Лямбда-выражение может быть присвоено значением переменной интерфейсного типа - но при условии, что интерфейс функциональный, а сигнатура абстрактного метода в функциональном интерфейсе соответствует параметрам лямбда-выражения (количество и тип параметров)



Создается объект на основе класса (анонимного), реализующего данный интерфейс, а в качестве кода для определения абстрактного метода используется код лямбда-выражения

Использование лямбда-выражения

Программа: использование лямбда-выражения - 1

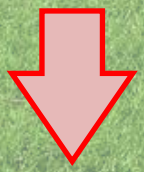
```
// Функциональный интерфейс:
interface MyNums{
   // Метод с кодом по умолчанию:
   default void show(int n){
      System.out.println("Аргумент: "+n);
      System.out.println("Результат: "+get(n));
   }
   // Абстрактный метод:
   int get(int n);
}
// Главный класс:
class UsingLambdaDemo{
   // Главный метод:
   public static void main(String[] args){
      // Присваивание лямбда-выражения значением
      // переменной интерфейсного типа:
      MyNums A=(int n)->{
         // Локальные переменные:
         int k,s=0;
         // Вычисление суммы натуральных чисел:
         for(k=1;k<=n;k++){
            s+=k;
         }
         // Результат:
         return s;
      }; // Продолжение на следующем слайде!!!
```

Использование лямбда-выражения

Программа: использование лямбда-выражения - 2

```
        System.out.println("Используем лямбда-выражение:");
        // Вызов методов из интерфейсной переменной:
        A.show(10);
        System.out.println("Проверка: "+A.get(10));
        // Альтернативная ссылка на объект:
        MyNums B=A;
        System.out.println("Новое лямбда-выражение:");
        // Интерфейсной переменой значением присваивается
        // новое лямбда-выражение:
        A=n->n*n;
        System.out.println("Вызов метода show() из A:");
        A.show(10);
        System.out.println("Вызов метода show() из B:");
        B.show(10);
    }
}
```

Альтернативный способ создания объектов



Результат

```
Используем лямбда-выражение:
Аргумент: 10
Результат: 55
Проверка: 55
Новое лямбда-выражение:
Вызов метода show() из A:
Аргумент: 10
Результат: 100
Вызов метода show() из B:
Аргумент: 10
Результат: 55
```

Использование анонимных классов

Программа: без лямбда-выражений - 1

```
interface MyNums{                          // Интерфейс
   default void show(int n){
      System.out.println("Аргумент: "+n);
      System.out.println("Результат: "+get(n));
   }
   int get(int n);
}
class UsingAnonymousClassDemo{
   public static void main(String[] args){
      // Присваивание интерфейсной переменной ссылки
      // на объект анонимного класса:
      MyNums A=new MyNums(){
         public int get(int n){     // Описание метода из интерфейса
            int k,s=0;
            for(k=1;k<=n;k++){
               s+=k;
            }
            return s;
         }
      };
      System.out.println("Используем анонимный класс:");
      A.show(10);
      System.out.println("Проверка: "+A.get(10));
      MyNums B=A;
      System.out.println("Новый анонимный класс:");
      // Продолжение на следующем слайде!!!
```

Использование анонимных классов

Программа: без лямбда-выражений - 2

```
        // Переменой значением присваивается ссылка
        // на объект нового анонимного класса:
        A=new MyNums(){
            // Описание метода из интерфейса:
            public int get(int n){
                return n*n;
            }
        };
        System.out.println("Вызов метода show() из A:");
        A.show(10);
        System.out.println("Вызов метода show() из B:");
        B.show(10);
    }
}
```

Результат

```
Используем анонимный класс:
Аргумент: 10
Результат: 55
Проверка: 55
Новый анонимный класс:
Вызов метода show() из A:
Аргумент: 10
Результат: 100
Вызов метода show() из B:
Аргумент: 10
Результат: 55
```

Использование лямбда-выражений

Программа: несколько интерфейсов

```
// Первый функциональный интерфейс:
interface Alpha{
   void showA();
} // Второй функциональный интерфейс:
interface Bravo{
   void showB();
} // Третий функциональный интерфейс:
interface Charlie{
   void showC();
} // Главный класс:
class LambdaAndInterfacesDemo{
   public static void main(String[] args){      // Главный метод
      // Значением интерфейсной переменной присваивается лямбда-выражение:
      Alpha A=()->System.out.println("Используем лямбда-выражение");
      // Вызов метода из интерфейсной переменой:
      A.showA();
      // Значением интерфейсной переменной присваивается лямбда-выражение:
      Bravo B=()->System.out.println("Используем лямбда-выражение");
      // Вызов метода из интерфейсной переменой:
      B.showB();
      // Значением интерфейсной переменной присваивается ссылка на метод:
      Charlie C=A::showA;
      // Вызов метода из интерфейсной переменной:
      C.showC();
   }
}
```

Результат

```
Используем лямбда-выражение
Используем лямбда-выражение
Используем лямбда-выражение
```

Ссылка на метод объекта

Ссылка на нестатический метод конкретного объекта выполняется так: указывается имя объекта, и через оператор `::` указывается название метода

Ссылка на метод объекта
`объект::имя_метода`

Если ссылка на объект выполнена в формате `объект::метод`, и при этом метод описан с некоторыми аргументами, то ссылке объект::метод соответствует лямбда-выражение вида

```
аргументы->объект.метод(аргументы)
```

Ссылка на метод объекта

Программа: ссылка на метод объекта - 1

```
// Класс:
class MyClass{
   // Закрытое целочисленное поле:
   private int number;
   // Конструктор:
   MyClass(int n){
      number=n;
   }
   // Метод для присваивания значения полю:
   void set(int n){
      number=n;
   }
   // Метод для считывания значения поля:
   int get(){
      return number;
   }
}
// Первый функциональный интерфейс:
interface MyGetter{
   int myget();
}
// Второй функциональный интерфейс:
interface MySetter{
   void myset(int n);
}
// Продолжение на следующем слайде!!!
```

Ссылка на метод объекта

Программа: ссылка на метод объекта - 2

```
class ObjMethReferenceDemo{                    // Главный класс
   public static void main(String[] args){     // Главный метод
      MyClass obj=new MyClass(100);            // Создание объекта
      System.out.println("Создан объект с полем 100");
      MyGetter A=obj::get;                     // Используем ссылки на методы
      MySetter B=obj::set;
      // Проверяем "значение поля" вызовом метода myget()
      // из интерфейсной переменной A:
      System.out.println("Переменная A: "+A.myget());
      // Проверяем значение поля объекта:
      System.out.println("Переменная obj: "+obj.get());
      // Полю объекта присваивается значение:
      obj.set(200);
      System.out.println("Полю присвоено значение 200");
      // Проверяем "значение поля" вызовом метода myget()
      // из интерфейсной переменной A:
      System.out.println("Переменная A: "+A.myget());
      // Присваиваем "значение полю" вызовом метода myset()
      // из интерфейсной переменной B:
      B.myset(300);
      System.out.println("Выполнена команда B.myset(300)");
      // Проверяем "значение поля" вызовом метода myget()
      // из интерфейсной переменной A:
      System.out.println("Переменная A: "+A.myget());
      // Продолжение на следующем слайде!!!
```

Ссылка на метод объекта

Программа: ссылка на метод объекта - 3

```
        // Проверяем значение поля объекта:
        System.out.println("Переменная obj: "+obj.get());
        // Создается новый объект:
        obj=new MyClass(400);
        System.out.println("Создан объект с полем 400");
        // Проверяем "значение поля" вызовом метода myget()
        // из интерфейсной переменной A:
        System.out.println("Переменная A: "+A.myget());
        // Проверяем значение поля объекта:
        System.out.println("Переменная obj: "+obj.get());
        // Присваиваем "значение полю" вызовом метода myset()
        // из интерфейсной переменной B:
        B.myset(500);
        System.out.println("Выполнена команда B.myset(500)");
        // Проверяем "значение поля" вызовом метода myget()
        // из интерфейсной переменной A:
        System.out.println("Переменная A: "+A.myget());
        // Проверяем значение поля объекта:
        System.out.println("Переменная obj: "+obj.get());
    }
}
```

Ссылка на метод объекта

Результат

```
Создан объект с полем 100
Переменная A: 100
Переменная obj: 100
Полю присвоено значение 200
Переменная A: 200
Выполнена команда B.myset(300)
Переменная A: 300
Переменная obj: 300
Создан объект с полем 400
Переменная A: 300
Переменная obj: 400
Выполнена команда B.myset(500)
Переменная A: 500
Переменная obj: 400
```

Ссылка на нестатический метод класса

Ссылка на нестатический метод класса выполняется в следующем виде: указывается имя класса, и через оператор `::` указывается название метода

Ссылка на нестатический метод класса

```
класс::имя_метода
```

Если используется ссылка вида `класс::метод`, и при этом метод нестатический и описан с некоторыми аргументами, то указанная ссылка эквивалента лямбда-выражению следующего вида:

```
(объект,аргументы)->объект.метод(аргумнты)
```

- Допустим, имеется класс `MyClass`, и в нем описан нестатический метод `method()` с аргументом `num` типа `int` и аргументом `symb` типа `char`
- Ссылка на метод `MyClass::method` соответствует лямбда-выражение вида `(MyClass obj,int num,char symb)->{код_метода_method()}`
- Если ссылку `MyClass::method` присвоить интерфейсной переменной, то в интерфейсе абстрактный метод должен быть объявлен с тремя аргументами: типа `MyClass`, `int` и `char` (тип результата метода должен согласовываться с типом результата метода `method()`)

Ссылка на нестатический метод класса

Программа: ссылка на нестатический метод класса - 1

```
class MyClass{ // Класс с полем и методами
   private int number;
   MyClass(int n){
      number=n;
   }
   void set(int n){
      number=n;
   }
   int get(){
      return number;
   }
} // Первый функциональный интерфейс:
interface MyGetter{
   int myget(MyClass obj);
} // Второй функциональный интерфейс:
interface MySetter{
   void myset(MyClass obj,int n);
} // Главный класс:
class MethReferenceDemo{
   public static void main(String[] args){
      // Создается объект:
      MyClass obj=new MyClass(100);
      System.out.println("Создан объект с полем 100");
      // Используем ссылки на методы:
      MyGetter A=MyClass::get;
      MySetter B=MyClass::set; // Продолжение на следующем слайде
```

Ссылка на нестатический метод класса

Программа: ссылка на нестатический метод класса - 2

```
        System.out.println("Переменная A: "+A.myget(obj));
        System.out.println("Переменная obj: "+obj.get());
        obj.set(200);
        System.out.println("Полю присвоено значение 200");
        System.out.println("Переменная A: "+A.myget(obj));
        B.myset(obj,300);
        System.out.println("Выполнена команда B.myset(obj,300)");
        System.out.println("Переменная A: "+A.myget(obj));
        System.out.println("Переменная obj: "+obj.get());
        // Создается новый объект:
        obj=new MyClass(400);
        System.out.println("Создан объект с полем 400");
        System.out.println("Переменная A: "+A.myget(obj));
        System.out.println("Переменная obj: "+obj.get());
        B.myset(obj,500);
        System.out.println("Выполнена команда B.myset(obj,500)");
        System.out.println("Переменная A: "+A.myget(obj));
        System.out.println("Переменная obj: "+obj.get());
    }
}
```

Ссылка на нестатический метод класса

Результат

```
Создан объект с полем 100
Переменная A: 100
Переменная obj: 100
Полю присвоено значение 200
Переменная A: 200
Выполнена команда
B.myset(obj,300)
Переменная A: 300
Переменная obj: 300
Создан объект с полем 400
Переменная A: 400
Переменная obj: 400
Выполнена команда
B.myset(obj,500)
Переменная A: 500
Переменная obj: 500
```

Ссылка на статический метод

Ссылка на статический метод класса выполняется в следующем виде: указывается имя класса, и через оператор `::` указывается название метода (то есть выполняется так же, как на нестатический метод класса)

Ссылка на статический метод класса
`класс::имя_метода`

Если используется ссылка на метод вида `класс::метод`, и при этом `метод` является статическим и описан с некоторыми `аргументами`, то указанная выше ссылка эквивалентна лямбда-выражению вида

```
(аргументы)->класс.метод(аргументы)
```

Например, ссылка `System.out::println` эквивалентна лямбда-выражению `t->System.out.println(t)`

Ссылка на статический метод

Программа: ссылка на статический метод класса - 1

```
// Класс со статическими методами:
class MyClass{
   // Методом отображается сообщение:
   static void show(){
      System.out.println("Метод класса MyClass");
   }
   // Методом вычисляется сумма чисел:
   static int sum(int n){
      int k,s=0;
      for(k=1;k<=n;k++){
         s+=k;
      }
      return s;
   }
} // Первый интерфейс:
interface MyShow{
   void myshow();
} // Второй интерфейс:
interface MySum{
   int mysum(int n);
} // Третий интерфейс:
interface MyPrinter{
   void myprint(Object t);
}
// Продолжение на следующем слайде!!!
```

Ссылка на статический метод

Программа: ссылка на статический метод класса - 2

```
// Главный класс:
class StatMethReferenceDemo{
   // Главный метод:
   public static void main(String[] args){
      // Использование ссылок на статические методы:
      MyShow A=MyClass::show;
      MySum B=MyClass::sum;
      MyPrinter P=System.out::println;
      // Вызов методов из интерфейсных переменных:
      A.myshow();
      P.myprint("Сумма чисел: "+B.mysum(10));
   }
}
```

Результат

```
Метод класса MyClass
Сумма чисел: 55
```

Ссылка на конструктор

Ссылку можно выполнять не только на метод, но и на конструктор. Синтаксис выполнения ссылки на конструктор такой: после имени класса указывается оператор `::`, после которого следует ключевое слово `new`

Ссылка на конструктор

```
класс::new
```

Если использована ссылка на конструктор вида `класс::new`, то ее эквивалентом является лямбда-выражение вида `(аргументы)->new класс(аргументы)`, где подразумевается, что при создании объекта `класса` конструктору передаются `аргументы`

Фактически речь идет не просто о конструкторе, а о команде создания нового объекта с помощью конструктора. Результатом выражения вида `new класс(аргументы)`, очевидно, является ссылка на созданный объект - то есть значение типа `класс`. Такой результат должен быть у абстрактного метода из функционального интерфейса, переменной которого значением присваивается ссылка на конструктор

Ссылка на конструктор

Программа: ссылка на конструктор

```
class MyClass{              // Класс
   private int number;      // Закрытое поле
   MyClass(int n){          // Конструктор
      number=n;
   } // Открытые методы:
   void show(){
      System.out.println("Значение поля: "+number);
   }
   void set(int n){
      number=n;
   }
} // Интерфейс:
interface MyInterface{
   MyClass create(int n);
} // Главный класс:
class ConstructorReferenceDemo{
   public static void main(String[] args){  // Главный метод
      MyInterface ref=MyClass::new;  // Использование ссылки на конструктор
      // Создание объекта вызовом метода из интерфейсной переменной:
      MyClass obj=ref.create(100);
      // Вызов методов объекта:
      obj.show();
      obj.set(200);
      obj.show();
   }
}
```

Результат

```
Значение поля: 100
Значение поля: 200
```

Ссылка на перегруженный метод

- Ссылка может выполняться на перегруженный метод (в классе описано несколько версий метода)
- По ссылке на метод нельзя однозначно определить, о какой версии метода идет речь
- Вывод о том, какую версию метода следует использовать, делается на основе интерфейсной переменной, которой в качестве значения присваивается ссылка на метод

Программа: ссылка на перегруженный метод - 1

```
// Класс:
class MyClass{
    // Поле:
    int number;
    // Перегруженный метод:
    void set(){
        number=0;
    }
    void set(int n){
        number=n;
    }
}
// Первый интерфейс:
interface Alpha{
    void none();
}
// Продолжение на следующем слайде!!!
```

Ссылка на перегруженный метод

Программа: ссылка на перегруженный метод - 2

```
// Второй интерфейс:
interface Bravo{
   void one(int n);
}
// Главный класс:
class OverloadedMethRefDemo{
   // Главный метод:
   public static void main(String[] args){
      // Создание объекта:
      MyClass obj=new MyClass();
      // Использование ссылки на перегруженный метод:
      Alpha A=obj::set;
      Bravo B=obj::set;
      // Вызов метода через интерфейсную переменную:
      B.one(100);
      // Проверка значения поля объекта:
      System.out.println("Значение поля: "+obj.number);
      // Вызов метода через интерфейсную переменную:
      A.none();
      // Проверка значения поля объекта:
      System.out.println("Значение поля: "+obj.number);
   }
}
```

Результат

```
Значение поля: 100
Значение поля: 0
```

Домашнее задание

- Проанализировать программный код примеров, представленных в презентации
- Набрать код и проверить его функциональность

Продолжение следует…

Дополнение: передача лямбда-выражения аргументом методу

Вычисление интеграла

Детали

Не вдаваясь в несущественные детали, будем под интегралом $\int_a^b f(x)dx$ подразумевать площадь под кривой $f(x)$, при условии, что значения аргумента x попадают в диапазон $a \le x \le b$. Для вычисления интеграла диапазон интегрирования $a \le x \le b$ разбивается на n внутренних интервалов равной длины $h = \frac{b-a}{n}$, границы этих внутренних интервалов определяются узловыми точками $x_k = a + kh$, где индекс $k = 0,1,2,\dots,n$ (и, таким образом, $x_0 \equiv a$ и $x_n \equiv b$). Вычисляемый интеграл представляется в виде сумма интегралов $\int_a^b f(x)dx = \sum_{k=0}^{n-1} \int_{x_k}^{x_{k+1}} f(x)dx$. Для каждого из интегралов в сумме используется приближенное выражение $\int_{x_k}^{x_{k+1}} f(x)dx \approx \frac{f(x_k)+f(x_{k+1})}{2} h$. В итоге получаем формулу $\int_a^b f(x)dx \approx \frac{f(a)+f(b)}{2} h + h\sum_{k=1}^{n-1} f(x_k)$, которая называется *формулой трапеций*. Именно этой формулой мы воспользуемся для вычисления интегралов от разных функций.

Дополнение: передача лямбда-выражения аргументом методу

Вычисление интеграла

Для вычисления интеграла необходимо задать:
- функцию $f(x)$, которая находится под знаком интеграла и называется подынтегральной функцией;
- границы a и b области интегрирования.

Решать задачу будем с помощью специального статического метода, аргументами которому передаются лямбда-выражение (определяет подынтегральную функцию $f(x)$) и два числовых значения (границы области интегрирования a и b).

Результатом метод возвращает значение (приближенное) для интеграла. При вычислении результата используется формула

$$\int_a^b f(x)dx \approx \frac{f(a)+f(b)}{2}h + h\sum_{k=1}^{n-1} f(x_k) = \frac{f(a)+f(b)}{2}h + h\sum_{k=1}^{n-1} f(a+kh).$$

Параметр h вычисляется как $h = \frac{b-a}{n}$, а целочисленное значение n определяется через локальную переменную.

Дополнение: передача лямбда-выражения аргументом методу

Вычисление интеграла

```
interface MyFunction{                    // Функциональный интерфейс
  // Метод с double-аргументом и double-результатом:
  double f(double x);
} // Главный класс:
class IntegralCalcDemo{ // Статический метод для вычисления интеграла:
  static double integrate(MyFunction obj,double a,double b){
    int n=1000;                          // Количество внутренних интервалов
    double h=(b-a)/n;                    // Длина внутреннего интервала
    double s=(obj.f(a)+obj.f(b))*h/2;// Переменная для записи интегральной суммы
    for(int k=1;k<=n-1;k++){             // Вычисление интегральной суммы
      s+=h*obj.f(a+k*h);
    } // Результат метода - значение интеграла:
    return s;
  } // Главный метод:
  public static void main(String[] args){ // Вычисление интеграла:
    System.out.print(integrate((double x)->{return x*(1-x);},0,1));
    System.out.println(" - точное значение "+1.0/6); // Значение для сравнения
    // Вычисление интеграла:
    System.out.print(integrate((double x)->{return 1/x;},1,2));
    // Значение для сравнения:
    System.out.println(" - точное значение "+Math.log(2));
    // Вычисление интеграла:
    System.out.print(integrate((double x)->{return Math.exp(-x);},0,10));
    // Значение для сравнения:
    System.out.println(" - точное значение "+(1-Math.exp(-10)));}}
```

Дополнение: передача лямбда-выражения аргументом методу

Вычисление интеграла - результат

```
0.16666650000000022 - точное значение 0.16666666666666666
0.6931472430599375 - точное значение 0.6931471805599453
0.9999629330113494 - точное значение 0.9999546000702375
```

Дополнение: лямбда-выражения и результат метода

- **Лямбда-выражение может использоваться при определении результата метода**
- **Формальным результатом в таком случае является значение интерфейсного типа, но собственно в коде метода результат может быть "оформлен" в виде лямбда-выражения**

Вычисление производной

Детали

Производной $f'(x)$ для функции $f(x)$ называется предел отношения приращения функции к приращению аргумента при стремлении последнего к нулю: $f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x)-f(x)}{\Delta x}$. Для нас в данном случае важно то обстоятельство, что при вычислении производной на основе одной исходной функции (дифференцируемая функция $f(x)$) по определенным правилам получают другую функцию (производная функция $f'(x)$).

При вычислении производной в числовом виде нередко используют приближенное выражение $f'(x) \approx \frac{f(x+\Delta x)-f(x)}{\Delta x}$, в котором параметр Δx, обозначающий приращение аргумента, мал (в принципе, чем меньше - тем лучше), но отличен от нуля. Именно такой подходы мы используем далее для вычисления на основе дифференцируемой функции $f(x)$ ее производной $f'(x)$.

Дополнение: лямбда-выражения и результат метода

Вычисление производной

```
interface MyFunction{// Функциональный интерфейс
   // Метод с double-аргументом возвращает результатом значение типа double:
   double f(double x);
} // Главный класс:
class DerivativeCalcDemo{// Статический метод для вычисления производной:
   static MyFunction Derivative(MyFunction ref){
      double dx=1e-5;     // Приращение по аргументу для вычисления производной
      // Результат метода - лямбда-выражения:
      return (double x)->{return (ref.f(x+dx)-ref.f(x))/dx;};
   } // Главный метод:
   public static void main(String[] args){
      // Производная для первой функции:
      MyFunction A=Derivative((double x)->{return x*(3-x);});
      // Производная для второй функции:
      MyFunction B=Derivative((double x)->{return x*Math.exp(-x);});
      // Проверка результатов вычислений:
      System.out.println("Производная для первой функции");
      System.out.println("Вычислено:\tТочно:");
      for(double t=0;t<=5;t++){
         System.out.printf("%8.5f\t%8.5f\n",A.f(t),(3-2*t));
      }
      System.out.println("Производная для второй функции");
      System.out.println("Вычислено:\tТочно:");
      for(double t=0;t<=5;t++){
         System.out.printf("%8.5f\t%8.5f\n",B.f(t),Math.exp(-t)*(1-t));
}}}
```

Дополнение: лямбда-выражения и результат метода

Вычисление производной - результат

```
Производная для первой функции
Вычислено:    Точно:
 2,99999       3,00000
 0,99999       1,00000
-1,00001      -1,00000
-3,00001      -3,00000
-5,00001      -5,00000
-7,00001      -7,00000
Производная для второй функции
Вычислено:    Точно:
 0,99999       1,00000
-0,00000       0,00000
-0,13534      -0,13534
-0,09957      -0,09957
-0,05495      -0,05495
-0,02695      -0,02695
```

Дополнение: лямбда-выражение и поле объекта

Лямбда-выражение и поле объекта - 1

```
// Функциональный интерфейс:
interface MyInterface{
    // Метод с целочисленным аргументом возвращает
    // целочисленный результат:
    int getNumber(int n);
}
// Класс с полем интерфейсного типа:
class MyClass{
    // Закрытое поле интерфейсного типа:
    private MyInterface ref;
    // Конструктор:
    MyClass(MyInterface mi){
        set(mi);
    }
    // Метод для присваивания значения полю:
    void set(MyInterface mi){
        ref=mi;
    }
    // Метод с целочисленным аргументом возвращает
    // результатом целое число:
    int get(int n){
        // Вызов метода из объекта, на которое ссылается
        // поле интерфейсного типа:
        return ref.getNumber(n);
    }
} // Продолжение на следующем слайде!!!
```

Дополнение: лямбда-выражение и поле объекта

Лямбда-выражение и поле объекта - 2

```
// Главный класс:
class LambdaAsFieldDemo{
   // Главный метод:
   public static void main(String[] args){
      // Создание объекта класса с передачей аргументом
      // конструктору лямбда-выражения:
      MyClass obj=new MyClass((int n)->{return n*n;});
      // Проверка результата:
      System.out.println("Аргумент:");
      for(int k=0;k<=5;k++){
         System.out.print(k+"\t");
      }
      System.out.println("\nАргумент в квадрате:");
      for(int k=0;k<=5;k++){
         System.out.print(obj.get(k)+"\t");
      }
      // Полю объекта присваивается новое значение:
      obj.set((int n)->{return n*n*n;});
      // Проверка результата:
      System.out.println("\nАргумент в кубе:");
      for(int k=0;k<=5;k++){
         System.out.print(obj.get(k)+"\t");
      }
      System.out.println("");
   }
}
```

```
Аргумент:
0    1    2    3    4    5
Аргумент в квадрате:
0    1    4    9    16   25
Аргумент в кубе:
0    1    8    27   64   125
```