

Java

Занятие 8. Вложенные классы, лямбды, Stream.

План

Вложенные классы в Java

- локальные классы
- статические и нестатические классы
- затенение
- анонимные классы

Вложенные классы в Java



Статические и нестатические вложенные классы

```
class OuterClass {  
...  
    static class StaticNestedClass {  
        ...  
    }  
    class InnerClass {  
        ...  
    }  
}
```

Вложенные классы в Java. Зачем?



```
public class Bicycle {  
  
    private String model;  
    private int weight;  
  
    public Bicycle(String model, int weight) {  
        this.model = model;  
        this.weight = weight;  
    }  
  
    public void start() {  
        System.out.println("Поехали!");  
    }  
  
    public class SteeringWheel {  
  
        public void right() {  
            System.out.println("Руль вправо!");  
        }  
  
        public void left() {  
  
            System.out.println("Руль влево!");  
        }  
    }  
  
    public class Seat {  
  
        public void up() {  
  
            System.out.println("сиденье поднято выше!");  
        }  
  
        public void down() {  
  
            System.out.println("сиденье опущено ниже!");  
        }  
    }  
}
```

Статические классы

```
public class Boeing737 {  
    private int manufactureYear;  
    private static int maxPassengersCount = 300;  
  
    public Boeing737(int manufactureYear) {  
        this.manufactureYear = manufactureYear;  
    }  
  
    public int getManufactureYear() {  
        return manufactureYear;  
    }  
  
    public static class Drawing {  
        public static int getMaxPassengersCount() {  
            return maxPassengersCount;  
        }  
    }  
}
```


Локальные классы

```
public class PhoneNumberValidator {  
    public void validatePhoneNumber(String number) {  
        class PhoneNumber {  
            private String phoneNumber;  
  
            public PhoneNumber() {  
                this.phoneNumber = number;  
            }  
  
            public String getPhoneNumber() {  
                return phoneNumber;  
            }  
  
            public void setPhoneNumber(String phoneNumber) {  
                this.phoneNumber = phoneNumber;  
            }  
        }  
        //...код валидации номера  
    }  
}
```

АНОНИМНЫЕ КЛАССЫ

```
Thread thread = new Thread(new Runnable() {  
    @Override  
    public void run() {  
        for (int i = 0; i < 10; i ++){  
            try {  
                Thread.sleep(500);  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
            System.out.println("i am anonymous");  
        }  
    }  
});
```

Функциональные интерфейсы

@FunctionalInterface

```
public interface Comparator<T> {  
    int compare(T var1, T var2);  
  
    ...  
}
```

Функциональные интерфейсы

```
TreeSet<Integer> set = new TreeSet<>(new Comparator<Integer>() {  
    @Override  
    public int compare(Integer var0, Integer var1) {  
        return var0 < var1 ? -1 : (var0.equals(var1) ? 0 : 1);  
    }  
});
```

Функциональные интерфейсы

```
TreeSet<Integer> set = new TreeSet<>(new Comparator<Integer>() {  
    @Override  
    public int compare(Integer var0, Integer var1) {  
        return var0 < var1 ? -1 : (var0.equals(var1) ? 0 : 1);  
    }  
});
```

Функциональные интерфейсы

- `Comparator<T>`
- `Consumer<T>`
- `Function<T,R>`
- `Predicate<T>`
- `Supplier<T>`
- Би – формы
- [Package java.util.function](#)

Predicate<T>

Функциональный интерфейс Predicate<T> проверяет соблюдение некоторого условия. Если оно соблюдается, то возвращается значение true. В качестве параметра принимает объект типа T:

```
@FunctionalInterface
public interface Predicate<T> {
    boolean test(T var1);

    ...
}
```

Function<T,R>

Функциональный интерфейс Function<T,R> представляет функцию перехода от объекта типа T к объекту типа R:

```
@FunctionalInterface
public interface Function<T, R> {
    R apply(T var1);
    ...
}
```


Consumer<T>

Consumer<T> выполняет некоторое действие над объектом типа T, при этом ничего не возвращая:

```
@FunctionalInterface
public interface Consumer<T> {
    void accept(T var1);
    ...
}
```

Supplier<T>

Supplier<T> не принимает никаких аргументов, но должен возвращать объект типа T:

```
@FunctionalInterface  
public interface Supplier<T> {  
    T get();  
}
```

А что если лямбда?



А что если лямбда?

Java 7:

```
TreeSet<Integer> set = new TreeSet<>(new Comparator<Integer>() {  
    @Override  
    public int compare(Integer var0, Integer var1) {  
        return var0 < var1 ? -1 : (var0.equals(var1) ? 0 : 1);  
    }  
});
```

Java 8:

```
TreeSet<Integer> set = new TreeSet<>((var0, var1) -> var0 < var1 ? -1 :  
    (var0.equals(var1) ? 0 : 1));
```

Как это работает?

Список аргументов	Значок стрелки	тело
<code>(int x, int y)</code>	<code>-></code>	<code>x + y</code>

Как это работает?

```
public TreeSet(Comparator<? super E> var1) {  
    ...  
}
```

```
TreeSet<Integer> set = new TreeSet<>((var0, var1) -> var0 < var1 ? -1 : (var0.equals(var1) ? 0 : 1));
```

Как это работает?

- No arguments: `() -> System.out.println("Hello")`
- One argument: `s -> System.out.println(s)`
- Two arguments: `(x, y) -> x + y`
- With explicit argument types:
`(Integer x, Integer y) -> x + y`
- Multiple statements:
`(x, y) -> {
 System.out.println(x);
 System.out.println(y);
 return (x + y);
}`

Как это работает?

Method reference

1

`HashFunc::compute1`

`(x) -> HashFunc::compute1(x)`

2

`HashFunc::compute2`

`(x, y) -> HashFunc::compute2(x, y)`

```
public class HashFunction {  
    public static int compute1(Object obj) { /*...*/ }  
    public static int compute2(int a, int b) { /*...*/ }  
}
```


Пример

Java 7:

```
Thread thread = new Thread(new Runnable() {  
    @Override  
    public void run() {  
        for (int i = 0; i < 10; i ++){  
            try {  
                Thread.sleep(500);  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
            System.out.println("i am  
anonymous");  
        }  
    }  
});
```

Java 8:

```
Thread thread = new Thread(() -> {  
    for (int i = 0; i < 10; i ++){  
        try {  
            Thread.sleep(500);  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
        System.out.println("i am anonymous");  
    }  
});
```

Практикум

Задание 1:

Создать класс University, в котором будут:

Два вложенных класса Student и Teacher (для простоты поля просто Имя и Фамилия, у каждого должно быть по конструктору)

Один статический вложенный класс GradingSystem (одно поле - Map<Integer,String>)

Поля: сет студентов и сет преподавателей, система оценок

Конструктор, принимающий два списка и система оценок

Метод main, в котором создаются два списка, мапа оценок и все это передается в тело конструктора

Практикум

Задание 2:

Дополнить класс University:

Добавить в Student и Teacher поле int id (уникальное)

В методе main создавать не просто Set, а TreeSet с компаратором (сравнивать по id)

Написать метод toString, который будет выводить сеты и мапу с использованием .forEach()

Дополнительно: добавить Predicate, который будет проверять, что имя и фамилия не пустые (например, внутри конструктора в цикле)

