

ИНТЕРФЕЙСЫ И ЛЯМБДА-ВЫРАЖЕНИЯ

Функциональные интерфейсы

■ Интерфейс с единственным методом

```
public interface Comparator <T>{  
    int compare(T first, T second);  
}
```

```
public interface Runnable {  
    void run();  
}
```

```
public interface ActionListener {  
    void actionPerformed (ActionEvent e);  
}
```

Функциональные интерфейсы

- Аннотация

`@FunctionalInterface`

Чаще всего для реализации интерфейса используют анонимные классы

Без использования лямбда-выражения

```
String [ ] words;  
.  
.  
.  
Arrays.sort (words); // стандартное сравнение  
  
Arrays.sort(words, new Comparator<String>(){  
    public int compare(String first, String second){  
        return first.length() - second.length();  
    }  
});
```

Лямбда-выражение

Java 8 внесено расширение на уровне языка. Можно использовать там, где ожидается объект класса, реализующего функциональный интерфейс.

```
String [ ] words;
```

```
. . .
```

```
Arrays.sort (words,  
             (first,second) -> first.length() - second.length());
```

```
Comparator<String> comp =  
    (first,second) -> first.length() - second.length();
```

```
Arrays.sort (words, comp );
```

Лямбда-выражение

- Лямбда-выражение можно присваивать переменной, типом которой является функциональный интерфейс
- Лямбда-выражение можно использовать там, где требуется объект класса, реализующего функциональный интерфейс.
- Java компилирует lambda-выражения с преобразованием их в private-методы класса.

Формат записи

```
Runnable runn = () -> System.out.println("!!!!!");
```

```
ActionListener lsnr =
```

```
    event -> System.out.println("BUTTON");
```

```
Runnable mult = () -> {
```

```
    System.out.println("!!!!!");
```

```
    for ( ; ;) System.out.println(".");
```

```
}
```

```
BinaryOperator <Long> add = (x,y) -> { return x+y; } // (x,y) -> x+y;
```

Использование значений

Изнутри лямбда-выражения можно не только обращаться ко всем «видимым» переменным, но и вызывать те методы, к которым есть доступ

При реализации анонимным классом захват контекста – финальные переменные

```
final String name = getUsername(); // требование
```

```
button.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        System.out.println("Hi "+name);  
    }  
});
```

Использование значений

В лямбда-выражении – эффективно финальные переменные (становятся такими, если попадают в лямбда-выражение)

```
String name = getUsername();
```

```
button.addActionListener( event ->  
    System.out.println("Hi "+name);  
);
```

```
//не допустимо name = getOtherName();
```

Функциональные интерфейсы

■ Пакет

`java.util.function,*`

Интерфейс	Метод
<code>Predicate <T></code>	<code>boolean test (T t)</code>
<code>BinaryOperator<T></code>	<code>T apply (T f, T s)</code>
<code>Consumer <T></code>	<code>void accept (T t)</code>
<code>Supplier <T></code>	<code>T get()</code>

Ссылки на методы

- В лямбда-выражении можно использовать имеющиеся методы классов

```
Arrays.sort(words, (x, y) -> x.compareToIgnoreCase(y));
```

- Для упрощения можно использовать ссылку на метод класса String

```
Arrays.sort(words, String::compareToIgnoreCase);
```

Ссылки на методы

```
List <String> ws =Arrays.asList(words);  
.  
.  
.  
// forEach – параметр Consumer<? super T>  
  
ws.forEach(x->System.out.println(x));  
ws.forEach(System.out::println);  
  
// replaceAll –параметр UnaryOperator<E>  
ws.replaceAll(x->x.toUpperCase());  
ws.replaceAll(String::toUpperCase);  
  
// removeIf – параметр Predicat <?super E>  
ws.removeIf(x->x.isNull());  
ws.removeIf(Objects::isNull);
```

Собственные функциональные интерфейсы

```
@FunctionalInterface
```

```
public interface WorkerInterface {  
    public void doSomeWork();  
}
```

Собственные функциональные интерфейсы

```
public class WorkerInterfaceTest {
    public static void execute(WorkerInterface worker) { worker.doSomeWork();
    }
    public static void main(String [] args) {
        execute(new WorkerInterface() {
            public void doSomeWork() {
                System.out.println("Worker вызван через анонимный класс");
            }
        });
        execute( () -> System.out.println("Worker вызван через Lambda") );
        WorkerInterface ex = ()-> System.out.println(" Lambda для Worker ");
        execute(ex);
    } // main()
}
```

Пример с предикатом

```
public static void filter(List<Integer> list,
                        Predicate<Integer> predicate) {
    for(Integer n: list) {
        if(predicate.test(n)) {
            System.out.print(n + " ");
        }
    }
    System.out.println();
}
```

Пример с предикатом

```
List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7);
System.out.println("Выводит все числа: ");
filter(list, (n)->true);
System.out.println("Не выводит ни одного числа: ");
filter(list, (n)->false);
System.out.println("Вывод четных чисел: ");
filter(list, (n)-> n%2 == 0 );
System.out.println("Вывод чисел больше 5: ");
filter(list, (n)-> n > 5 );
```