Введение в программирование на Java

Лекция 15. Начала функционального программирования.

Виталий Олегович Афанасьев

19 мая 2025

Императивное программирование

```
1 public static int fib(int n) {
2     if (n <= 0) {
3         return 0;
4     }
5     if (n <= 2) {
6         return 1;
7     }
8     return fib(n - 1) + fib(n - 2);
9 }
```

Императивное программирование

```java
public static int fib(int n) {
    if (n <= 0) {
        return 0;
    }
    if (n <= 2) {
        return 1;
    }
    return fib(n - 1) + fib(n - 2);
}
```

Императивный стиль:

1. Если `n` меньше, либо равно 0, вернуть 0.
2. Иначе, если `n` меньше, либо равно 2, вернуть 1.
3. Иначе:
4. Вызвать `fib` со значением `n - 1`.
5. Вызвать `fib` со значением `n - 2`.
6. Сложить результаты вызовов.
7. Вернуть результат сложения.

Декларативное программирование

```
1 public static int fib(int n) {
2     if (n <= 0) {
3         return 0;
4     }
5     if (n <= 2) {
6         return 1;
7     }
8     return fib(n - 1) + fib(n - 2);
9 }
```

Декларативный стиль:

fib — это такая функция, что

- $fib(n) = 0$, при $n \leq 0$;
- $fib(n) = 1$, при $1 \leq n \leq 2$;
- $fib(n) = fib(n-1) + fib(n-2)$, при $3 \leq n$;

Императивное и декларативное программирование

Императивная парадигма — **как** решить задачу (при помощи какой последовательности шагов).

Декларативная парадигма — **что** является результатом (но не способ его получения).

Императивное и декларативное программирование

Императивная парадигма — **как** решить задачу (при помощи какой последовательности шагов).

Декларативная парадигма — **что** является результатом (но не способ его получения).

Хороший пример чистого декларативного языка: SQL.

```
SELECT employee_name
    FROM employees
    WHERE salary > 100
    ORDER BY employee_name;
```

Функциональное программирование

Функциональное программирование рассматривает функции как **объекты первого класса (first-class citizens)**.

Функции можно сохранять в переменные, передавать в другие функции как аргументы и возвращать из функций.

- Haskell
- Lisp
- Scala
- Erlang
- F#
- **Java** (с некоторыми особенностями)
- ...

Сортировка с компаратором (1)

```
1 public interface Comparator {
2     int compare(int first, int second);
3 }
```

```
1 public int[] myCoolSort(int[] values, Comparator comparator) {
2     ...
3     boolean ordered = comparator.compare(a, b) <= 0;
4     ...
5 }
```

Сортировка с компаратором (2)

Чтобы использовать такую сортировку, необходимо определить целый класс:

```
1 public class NonAscending implements Comparator {
2     @Override
3     public int compare(int first, int second) {
4         return second - first;
5     }
6 }
```

```
1 int[] values = ...;
2 int[] sortedValues = myCoolSort(values, new NonAscending());
```

Анонимные классы

```
1 public interface Comparator {
2     int compare(int first, int second);
3 }
```

Можно создать реализацию интерфейса (или абстрактного класса) по месту (без создания отдельного класса).

Такая реализация не имеет имени, поэтому называется **анонимным классом**.

```
1 int[] values = ...;
2 int[] sortedValues = myCoolSort(values, new Comparator() {
3     @Override
4     public int compare(int first, int second) {
5         return second - first;
6     }
7 });
```

Лямбда-выражения (1)

```
1 int[] values = ...;
2 int[] sortedValues = myCoolSort(values, new Comparator() {
3     @Override
4     public int compare(int first, int second) {
5         return second - first;
6     }
7 });
```

Лямбда-выражения (1)

```
1 int[] values = ...;
2 int[] sortedValues = myCoolSort(values, new Comparator() {
3     @Override
4     public int compare(int first, int second) {
5         return second - first;
6     }
7 });
```

Т.к. интерфейс состоит всего из одного абстрактного метода, запись можно сократить ещё больше.

Создадим **анонимную функцию** (в Java их называют **лямбда-выражениями**):

```
1 int[] values = ...;
2 int[] sortedValues = myCoolSort(
3     values,
4     (int first, int second) -> {
5         return second - first;
6     }
7 );
```

Лямбда-выражения (2)

В большинстве случаев компилятор автоматически выводит типы параметров, поэтому их можно опускать:

```
1 int[] values = ...;
2 int[] sortedValues = myCoolSort(
3     values,
4     (first, second) -> {
5         return second - first;
6     }
7 );
```

Лямбда-выражения (3)

Если тело лямбды состоит всего из одного `return` (т.е. результатом является единственное выражение), то записать можно ещё короче:

```
1 int[] values = ...;
2 int[] sortedValues = myCoolSort(
3     values,
4     (first, second) -> second - first
5 );
```

Функциональные интерфейсы (1)

Лямбда-выражения можно передавать как аргументы, только если в качестве типа параметра указан **функциональный интерфейс**.

Функциональный интерфейс — интерфейс с единственным абстрактным методом.

Такие интерфейсы лучше помечать аннотацией `@FunctionalInterface` — если интерфейс не функциональный, то код не скомпилируется.

```
1 @FunctionalInterface
2 public interface Comparator {
3     int compare(int first, int second);
4 }
```

Функциональные интерфейсы (2)

В пакете `java.util.function` определено множество стандартных функциональных интерфейсов:

- `Function<T, R>` — функция с параметром типа `T` и возвращаемым типом `R`.
- `Consumer<T>` — функция с параметром типа `T` и без возвращаемого значения.
- `Supplier<T>` — функция без параметров и возвращаемым типом `T`.
- `Runnable` — функция без параметров и без возвращаемого значения.
- `BiFunction<T, U, R>` — функция с параметрами типов `T` и `U` и возвращаемым типом `R`.
- `Predicate<T>` — функция с параметром типа `T` и возвращаемым типом `boolean`.
- `UnaryOperator<T>` — то же, что и `Function<T, T>`.
- ...

Функциональные интерфейсы: Пример со списками (1)

```
1 public interface List<T> {
2     boolean removeIf(Predicate<? super T> filter);
3     void forEach(Consumer<? super T> action);
4     void replaceAll(UnaryOperator<T> operator);
5     ...
6 }
```

Note: это упрощённый пример; некоторые из методов находятся в интерфейсе `Collection`, а не `List`.

Функциональные интерфейсы: Пример со списками (2)

```
1  List<String> strings = Arrays.asList("Hello", "World", "Hi", "abc");
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

Функциональные интерфейсы: Пример со списками (2)

```
1  List<String> strings = Arrays.asList("Hello", "World", "Hi", "abc");
2  strings.forEach(
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

Функциональные интерфейсы: Пример со списками (2)

```
1  List<String> strings = Arrays.asList("Hello", "World", "Hi", "abc");
2  strings.forEach(
3      (item) -> {
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

Функциональные интерфейсы: Пример со списками (2)

```
List<String> strings = Arrays.asList("Hello", "World", "Hi", "abc");
strings.forEach(
    (item) -> {
        System.out.println(item); // Hello World Hi abc
```

Функциональные интерфейсы: Пример со списками (2)

```
List<String> strings = Arrays.asList("Hello", "World", "Hi", "abc");
strings.forEach(
    (item) -> {
        System.out.println(item); // Hello World Hi abc
    }
```

Функциональные интерфейсы: Пример со списками (2)

```
List<String> strings = Arrays.asList("Hello", "World", "Hi", "abc");
strings.forEach(
    (item) -> {
        System.out.println(item); // Hello World Hi abc
    }
);
```

Функциональные интерфейсы: Пример со списками (2)

```
List<String> strings = Arrays.asList("Hello", "World", "Hi", "abc");
strings.forEach(
    (item) -> {
        System.out.println(item); // Hello World Hi abc
    }
);

strings.removeIf((item) -> item.length() <= 3);
```

Функциональные интерфейсы: Пример со списками (2)

```
1  List<String> strings = Arrays.asList("Hello", "World", "Hi", "abc");
2  strings.forEach(
3      (item) -> {
4          System.out.println(item); // Hello World Hi abc
5      }
6  );
7
8  strings.removeIf((item) -> item.length() <= 3);
9  strings.forEach(
10
11
12
13
14
15
16
17
18
19
20
```

Функциональные интерфейсы: Пример со списками (2)

```
1  List<String> strings = Arrays.asList("Hello", "World", "Hi", "abc");
2  strings.forEach(
3      (item) -> {
4          System.out.println(item); // Hello World Hi abc
5      }
6  );
7
8  strings.removeIf((item) -> item.length() <= 3);
9  strings.forEach(
10     (item) -> {
11
12
13
14
15
16
17
18
19
20
```

Функциональные интерфейсы: Пример со списками (2)

```
1  List<String> strings = Arrays.asList("Hello", "World", "Hi", "abc");
2  strings.forEach(
3      (item) -> {
4          System.out.println(item); // Hello World Hi abc
5      }
6  );
7
8  strings.removeIf((item) -> item.length() <= 3);
9  strings.forEach(
10     (item) -> {
11         System.out.println(item); // Hello World
12
13
14
15
16
17
18
19
20
```

Функциональные интерфейсы: Пример со списками (2)

```
1  List<String> strings = Arrays.asList("Hello", "World", "Hi", "abc");
2  strings.forEach(
3      (item) -> {
4          System.out.println(item); // Hello World Hi abc
5      }
6  );
7
8  strings.removeIf((item) -> item.length() <= 3);
9  strings.forEach(
10     (item) -> {
11         System.out.println(item); // Hello World
12     }
13
14
15
16
17
18
19
20
```

Функциональные интерфейсы: Пример со списками (2)

```
1  List<String> strings = Arrays.asList("Hello", "World", "Hi", "abc");
2  strings.forEach(
3      (item) -> {
4          System.out.println(item); // Hello World Hi abc
5      }
6  );
7
8  strings.removeIf((item) -> item.length() <= 3);
9  strings.forEach(
10     (item) -> {
11         System.out.println(item); // Hello World
12     }
13 );
14
15
16
17
18
19
20
```

Функциональные интерфейсы: Пример со списками (2)

```
1  List<String> strings = Arrays.asList("Hello", "World", "Hi", "abc");
2  strings.forEach(
3      (item) -> {
4          System.out.println(item); // Hello World Hi abc
5      }
6  );
7
8  strings.removeIf((item) -> item.length() <= 3);
9  strings.forEach(
10     (item) -> {
11         System.out.println(item); // Hello World
12     }
13 );
14
15 strings.replaceAll((item) -> item.toLowerCase());
16
17
18
19
20
```

Функциональные интерфейсы: Пример со списками (2)

```
1  List<String> strings = Arrays.asList("Hello", "World", "Hi", "abc");
2  strings.forEach(
3      (item) -> {
4          System.out.println(item); // Hello World Hi abc
5      }
6  );
7
8  strings.removeIf((item) -> item.length() <= 3);
9  strings.forEach(
10     (item) -> {
11         System.out.println(item); // Hello World
12     }
13 );
14
15 strings.replaceAll((item) -> item.toLowerCase());
16 strings.forEach(
17
18
19
20
```

Функциональные интерфейсы: Пример со списками (2)

```
1  List<String> strings = Arrays.asList("Hello", "World", "Hi", "abc");
2  strings.forEach(
3      (item) -> {
4          System.out.println(item); // Hello World Hi abc
5      }
6  );
7
8  strings.removeIf((item) -> item.length() <= 3);
9  strings.forEach(
10     (item) -> {
11         System.out.println(item); // Hello World
12     }
13 );
14
15 strings.replaceAll((item) -> item.toLowerCase());
16 strings.forEach(
17     (item) -> {
18
19
20
```

Функциональные интерфейсы: Пример со списками (2)

```
1  List<String> strings = Arrays.asList("Hello", "World", "Hi", "abc");
2  strings.forEach(
3      (item) -> {
4          System.out.println(item); // Hello World Hi abc
5      }
6  );
7
8  strings.removeIf((item) -> item.length() <= 3);
9  strings.forEach(
10     (item) -> {
11         System.out.println(item); // Hello World
12     }
13 );
14
15 strings.replaceAll((item) -> item.toLowerCase());
16 strings.forEach(
17     (item) -> {
18         System.out.println(item); // hello world
19
20
```

Функциональные интерфейсы: Пример со списками (2)

```
1  List<String> strings = Arrays.asList("Hello", "World", "Hi", "abc");
2  strings.forEach(
3      (item) -> {
4          System.out.println(item); // Hello World Hi abc
5      }
6  );
7
8  strings.removeIf((item) -> item.length() <= 3);
9  strings.forEach(
10     (item) -> {
11         System.out.println(item); // Hello World
12     }
13 );
14
15 strings.replaceAll((item) -> item.toLowerCase());
16 strings.forEach(
17     (item) -> {
18         System.out.println(item); // hello world
19     }
20
```

Функциональные интерфейсы: Пример со списками (2)

```
1  List<String> strings = Arrays.asList("Hello", "World", "Hi", "abc");
2  strings.forEach(
3      (item) -> {
4          System.out.println(item); // Hello World Hi abc
5      }
6  );
7
8  strings.removeIf((item) -> item.length() <= 3);
9  strings.forEach(
10     (item) -> {
11         System.out.println(item); // Hello World
12     }
13 );
14
15 strings.replaceAll((item) -> item.toLowerCase());
16 strings.forEach(
17     (item) -> {
18         System.out.println(item); // hello world
19     }
20 );
```

Ссылки на методы (1)

Зачастую в качестве тела лямбды используется вызов всего одного метода:

```
1  public class Main {
2      public static void main(String[] args) {
3          ...
4          strings.removeIf(item -> isTooShort(item));
5          ...
6      }
7
8      private static boolean isTooShort(String s) {
9          return s.length() <= 3;
10     }
11 }
```

Ссылки на методы (2)

Такой код может быть записан ещё короче, посредством передачи ссылки на метод:

```
1  public class Main {
2      public static void main(String[] args) {
3          ...
4          strings.removeIf(Main::isTooShort);
5          ...
6      }
7
8      private static boolean isTooShort(String s) {
9          return s.length() <= 3;
10     }
11 }
```

Ссылки на методы (3)

Ссылки на функцию могут быть указаны тремя способами:

1. `Класс::статическаяФункция`
   - `private static boolean isTooShort(String s)`
   - `strings.removeIf(item -> Main.isTooShort(item))`
   - `strings.removeIf(Main::isTooShort)`
2. `Объект::неСтатическаяФункция`
   - `public void println(String s)`
   - `strings.forEach(item -> System.out.println(item))`
   - `strings.forEach(System.out::println)`
3. `Класс::неСтатическаяФункция`
   - `public void toLowerCase()`
   - `strings.replaceAll(item -> item.toLowerCase())`
   - `strings.replaceAll(String::toLowerCase)`

Ссылки на методы (4)

Ещё примеры:

- `x -> separator.equals(x)`
- `separator::equals`

- `(x, y) -> x.equals(y)`
- `String::equals`

- `str -> Integer.parseInt(str)`
- `Integer::parseInt`

- `name -> new Employee(name)`
- `Employee::new`

- `length -> new String[length]`
- `String[]::new`

Захват переменных (1)

Лямбда-выражения и анонимные классы могут **захватывать** переменные из внешней области:

```
1  public static void example() {
2      List<String> strings = Arrays.asList("Hello", "World");
3
4      String prefix = "> ";
5      strings.forEach(
6          (item) -> {
7              System.out.println(prefix + item);
8          }
9      );
10     // > Hello
11     // > World
12 }
```

Объекты, которые захватывают внешние значения, ещё называют **замыканиями (closures)**.

Захват переменных (2)

Но для захваченных переменных существует одно ограничение:

- Переменным должно присваиваться значение лишь единожды (т.е. быть **effectively final**).

Если же это не так — компилятор не позволит захватить такую переменную:

```
1  public static void example() {
2      List<String> strings = Arrays.asList("Hello", "World");
3
4      String prefix = "> ";
5      prefix = "--- ";
6      strings.forEach(
7          (item) -> {
8              // ERROR: prefix присваивается несколько раз
9              System.out.println(prefix + item);
10         }
11     );
12 }
```