Стремительное развитие Java: сравниваем на практике

СГУ им. Чернышевского
факультет КНиИТ
Черноусова Юлия Андреевна
Черноусова Елена Михайловна

Java 7 → 2011 год

Java 8 → 2014 год

Java 9 → 2017 год

Java 10 → 2018 год

Java 8

- лямбда-выражения
- ссылки на методы и конструкторы
- функциональные интерфейсы
- Stream API

Java 8. Лямбда-выражения

Лямбда-выражения – это сокращенная форма анонимных функций. Лямбда-выражения позволяют приблизить Java к функциональным языкам программирования.

Также Java 8 позволяет передавать **ссылки на методы и конструкторы**. Для этого нужно использовать конструкцию «::».

Примеры

С использованием анонимных классов:

```
Collections.sort(list, new Comparator<MyClass>() {
    public int compare(MyClass myClass1, MyClass myClass2) {
        return myClass1.getSomeProperty().compareTo(myClass2.getSomeProperty());
    }
});
```

С использованием лямбда-выражений:

```
Collections.sort(list, (myClass1, myClass2) ->
    myClass1.getSomeProperty().compareTo(myClass2.getSomeProperty()));
```

С использованием ссылки на метод:

```
Collections.sort(list, Comparator.comparing(MyClass::getSomeProperty));
```

Java 8. Функциональные интерфейсы

Каждому лямбда-выражению соответствует тип, представленный интерфейсом. Такой **функциональный интерфейс** должен содержать ровно один абстрактный метод. Каждое лямбда-выражение этого типа будет сопоставлено объявленному методу. Для того чтобы гарантировать, что интерфейс содержит ровно один абстрактный метод, используется аннотация *@FunctionalInterface*.

Java 8. Функциональные интерфейсы

```
@FunctionalInterface
interface Converter<F, T> {
    T convert(F from);
}

public static void main(String[] args) {
    Converter<String, Double> converter = (from) ->
        Double.valueOf(from);

    System.out.println(converter.convert("10.5")); //10.5
}
```

Java 8. Stream API

Технология **Stream API** играет большую роль в обеспечении функционального программирования в Java.

Stream API - это новый способ работать со структурами данных в функциональном стиле. Стримы являются *монадами*, то есть структурами, которые представляют вычисления в виде последовательности шагов.

Java 8. Stream API

```
int[] numbers = {-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5};
long count = 0;

//Стандартный способ
for (int i : numbers) {
    if (i > 0) count++;
}
System.out.println(count); //5

//С использованием Stream API
long countStream = IntStream.of(numbers)
    .filter(num -> num > 0).count();
System.out.println(countStream); //5
```

Java 8. Stream API. Методы

.filter() - фильтрует записи, соответствующие условию

.skip() - позволяет пропустить определенные элементы

.map() - преобразует каждый элемент

.sorted() - позволяет сортировать значения

...

Java 8. Parallel Stream API

Последовательные стримы:

```
long countElem = values.stream().sorted().count();
```

Параллельные стримы:

```
long countElem  = values.parallelStream().sorted().count();
```

Количество элементов списка	Время выполнения последовательной сортировки, мс	Время выполнения параллельной сортировки, мс
10.000	31	20
100.000	117	90
1.000.000	843	471

Java 9

- модульная система
- инструмент JShell
- неизменяемые объекты коллекций
- private методы в интерфейсах
- введение реактивного программирования
- изменения в try-with-resources

Java 9. Модульная система

До Java 9 использовались JAR-файлы для разработки приложений, базирующихся на Java. Однако эта архитектура имеет несколько ограничений и недостатков. Для их устранения внедрили модульную систему.

Это нововведение является частью проекта **Jigsaw**, который разрабатывался еще с 2008 года. Этот принцип организации позволяет разбивать программы на независимые и межпрограммные модули.

Java 9. JShell

JShell является REPL-системой. REPL (англ. read-eval-print loop) — система для интерактивного программирования в консоли. Это позволяет пользователю вводить строку кода и сразу видеть результат её выполнения.

```
C:\Program Files\Java\jdk-9.0.4\bin>jshell
|  Welcome to JShell -- Version 9.0.4

jshell> int a = 10
a ==> 10

jshell> int div (int a, int b){
   ...> return a / b; }
|  created method div(int,int)

jshell> div(10,4)
$3 ==> 2
```

Java 9. Неизменяемые объекты коллекций

В Java 9 были представлены несколько удобных методов для создания неизменяемых ***List, Set, Map*** и ***Map.Entry*** объектов без использования дополнительных классов. Интерфейсы теперь имеют методы of().

Пример создания Map с ключами *1, 2, 3* и элементами *one, two, three* соответственно:

```
Map immutableMap = Map.of(1, "one", 2, "two", 3, "three");
```

Java 9. Private методы в интерфейсах

Еще одним немаловажным нововведением является возможность создавать ***private*** методы в интерфейсах. Это позволяет избавиться от нагромождения и переизбытка кода. В Java 8 можно было обеспечивать реализацию метода в интерфейсах, используя default и static методы, что было не очень удобно.

Java 9. Reactive Stream API

Reactive Stream API позволяет реализовывать асинхронные, масштабируемые и параллельные приложения. Для реактивного программирования (программирование с асинхронными потоками данных) были добавлены следующие классы:

- java.util.concurrent.Flow.Publisher
- java.util.concurrent.Flow.Subscriber
- java.util.concurrent.Flow.Processor

Java 9. Try-with-resources

Еще в Java 7 появилась новая конструкция обработки исключений ***Try-With-Resources*** для автоматического управления ресурсами. В Java 9 были введены изменения в эту конструкцию, чтобы повысить читаемость кода.

```
//до Java 9
BufferedReader reader1 = new BufferedReader(new FileReader("input.txt"));
try (BufferedReader reader2 = reader1) {
    ...
}

//с Java 9
BufferedReader reader1 = new BufferedReader(new FileReader("input.txt"));
try (reader1) {
    ...
}
```

Java 10

- новый зарезервированный тип данных var
- совместное использование классов приложений
- параллельная сборка мусора

Java 10. Новый тип данных var

Новый тип данных *var* избавляет от необходимости указывать тип локальной переменной явно. Такой тип может использоваться при объявлении индекса в цикле, а также для любых проинициализированных локальных переменных.

```
var list = new ArrayList<String>(); //тип ArrayList<String>
var stream = list.stream(); //тип Stream<String>
```

Тип *var* нужно применять с осторожностью, так как существует много ситуаций, когда произойдет ошибка, например, нельзя присваивать переменной типа var значение null, лямбда-выражение, а также значения должны иметь однозначный тип.

Java 10. Совместное использование классов

В Java 10 было принято обновление под названием **Application Class-Data Sharing**, с помощью которого улучшается загрузка и отслеживание элементов, что позволяет классам приложений размещаться в общем архиве.

Теперь JVM не нужно загружать одни и те же классы, входящие в стандартную библиотеку. Это уменьшает время загрузки программ и используемую память.

Java 10. Параллельная сборка мусора

Теперь сборщик мусора **G1** (Garbage-First), который начиная с Java 9 работает по умолчанию, сможет производить сборку мусора сразу в нескольких потоках. Ранее это происходило только в одном.

Кроме того, с помощью параметра *XX:ParallelGCThreads* можно устанавливать количество потоков.

Заключение

Развитие информационных технологий не стоит на месте, и язык программирования Java меняется и развивается с каждой новой версией. Поэтому нужно всегда быть в курсе новых технологий, которые могут помочь при разработке.

Важно следить за изменениями в различных языках программирования не только при разработке, но и в образовательном процессе, чтобы знания, преподносящиеся обучающимся, были актуальными.

Спасибо за внимание!