

# Специализация Java

Иван Игоревич Овчинников

2022-08-16 (22:44)

# Оглавление

<b>1</b>	<b>Java Core</b>	<b>2</b>
1.1	Платформа: история и окружение . . . . .	2
1.2	Управление проектом: сборщики проектов . . . . .	9
1.3	Специализация: данные и функции . . . . .	9
1.4	Специализация: ООП . . . . .	11
1.5	Специализация: Тонкости работы . . . . .	11
<b>2</b>	<b>Java Development Kit</b>	<b>12</b>
2.1	Исключения . . . . .	12
2.2	Интерфейсы . . . . .	12
2.3	Обобщённое программирование . . . . .	12
2.4	Коллекции . . . . .	12
2.5	Многопоточность . . . . .	12
2.6	Графический интерфейс пользователя . . . . .	12
<b>3</b>	<b>Java Junior</b>	<b>13</b>
3.1	JDBC . . . . .	13
3.2	Сетевое программирование . . . . .	13
3.3	Введение в архитектуры приложений на Java . . . . .	13
3.4	Сериализация, StreamAPI, ReflectionAPI . . . . .	13
3.5	Введение в Java EE . . . . .	13
3.6	Введение в Spring framework . . . . .	13
<b>4</b>	<b>Java Junior+</b>	<b>14</b>
	<b>Приложения</b>	<b>15</b>
A	Термины, определения и сокращения . . . . .	15

# Глава 1

## Java Core

### 1.1. Платформа: история и окружение

#### Конспект лекции

##### Краткая история (причины возникновения)

- Язык создавали для разработки встраиваемых систем, сетевых приложений и прикладного ПО;
- Популярен из-за скорости исполнения и полного абстрагирования от исполнителя кода;
- Часто используется для программирования бэк-энда веб-приложений из-за изначальной направленности на сетевые приложения.

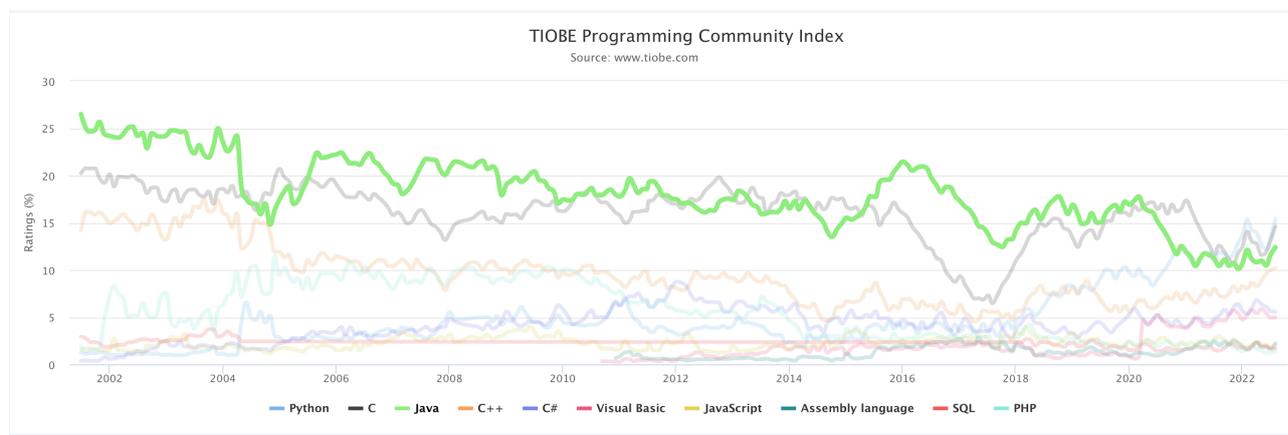


Рис. 1.1: График популярности языков программирования TIOBE

##### Базовый инструментарий, который понадобится (выбор IDE)

- NetBeans - хороший, добротный инструмент с лёгким ностальгическим оттенком;
- Eclipse - для поклонников Eclipse Foundation и швейцарских ножей с полусотней лезвий;
- IntelliJ IDEA - стандарт де-факто, используется на курсе и в большинстве современных компаний;
- Android Studio - если заниматься мобильной разработкой.

##### Что нужно скачать, откуда (как выбрать вендора, выбор версии)

Для разработки понадобится среда разработки (IDE) и инструментарий разработчика (JDK). JDK выпускается несколькими поставщиками, большинство из них бесплатны и полнофункциональны, то есть поддерживают весь функционал языка и платформы.

В последнее время, с развитием контейнеризации приложений, часто устанавливают инструментарий в Docker-контейнер и ведут разработку прямо в контейнере, это позволяет не захламлять компьютер разработчика разными версиями инструментария и быстро разворачивать свои приложения в CI или на целевом сервере.

В общем случае, для разработки на любом языке программирования нужны так называемые SDK (Software Development Kit, англ. - инструментарий разработчика приложений или инструментарий для разработки приложений). Частный случай такого SDK - инструментарий разработчика на языке Java - Java Development Kit.

На курсе будет использоваться BellSoft Liberica JDK 11, но возможно использовать и других производителей, например, самую распространённую Oracle JDK. Производителя следует выбирать из требований по лицензированию, так, например, Oracle JDK можно использовать бесплатно только в личных целях, за коммерческую разработку с использованием этого инструментария придётся заплатить.

Также возможно использовать и другие версии, но не старше 1.8. Это обосновано тем, что основные разработки на данный момент только начинают обновлять инструментарий до более новых версий (часто 11 или 13) или вовсе переходят на другие JVM-языки, такие как Scala, Groovy или Kotlin.

Для решения некоторых несложных задач курса мы будем писать простые приложения, не содержащие ООП, сложных взаимосвязей и проверок, в этом случае нам понадобится Jupyter notebook с установленным ядром (kernel) Java.

### Из чего всё состоит (JDK, JRE, JVM и их друзья)

TL;DR:

- JDK = JRE + инструменты разработчика;
- JRE = JVM + библиотеки классов;
- JVM = Native API + механизм исполнения + управление памятью.

Как именно всё работает? Если коротко, то слой за слоем накладывая абстракции. Программы на любом языке программирования исполняются на компьютере, то есть, так или иначе, задействуют процессор, оперативную память и прочие аппаратные компоненты. Эти аппаратные компоненты предоставляют для доступа к себе низкоуровневые интерфейсы, которые задействует операционная система, предоставляя в свою очередь интерфейс чуть проще программам, взаимодействующим с ней. Этот интерфейс взаимодействия с ОС мы для простоты будем называть Native API.

С ОС взаимодействует JVM (Wikipedia: Список виртуальных машин Java), то есть, используя Native API, нам становится всё равно, какая именно ОС установлена на компьютере, главное уметь выполняться на JVM. Это открывает простор для создания целой группы языков, они носят общее бытовое название JVM-языки, к ним относят Scala, Groovy, Kotlin и другие. Внутри JVM осуществляется управление памятью, существует механизм исполнения программ, специальный JIT<sup>1</sup>-компилятор, генерирующий платформенно-зависимый код.

JVM для своей работы запрашивает у ОС некоторый сегмент оперативной памяти, в котором хранит данные программы. Это хранение происходит «слоями»:

1. Eden Space (heap) – в этой области выделяется память под все создаваемые из программы объекты. Большая часть объектов живёт недолго (итераторы, временные объекты, используемые внутри методов и т.п.), и удаляются при выполнении сборки мусора этой области памяти, не перемещаются в другие области памяти. Когда данная область заполняется (т.е. количество выделенной памяти в этой области превышает некоторый заданный процент), сборщик мусора

<sup>1</sup>JIT, just-in-time - англ. вóвремя, прямо сейчас



выполняет быструю (minor collection) сборку. По сравнению с полной сборкой, она занимает мало времени, и затрагивает только эту область памяти, а именно, очищает от устаревших объектов Eden Space и перемещает выжившие объекты в следующую область.

2. Survivor Space (heap) – сюда перемещаются объекты из предыдущей области после того, как они пережили хотя бы одну сборку мусора. Время от времени долгоживущие объекты из этой области перемещаются в Tenured Space.
3. Tenured (Old) Generation (heap) — Здесь скапливаются долгоживущие объекты (крупные высокоуровневые объекты, синглтоны, менеджеры ресурсов и прочие). Когда заполняется эта область, выполняется полная сборка мусора (full, major collection), которая обрабатывает все созданные JVM объекты.
4. Permanent Generation (non-heap) – Здесь хранится метаданная, используемая JVM (используемые классы, методы и т.п.).
5. Code Cache (non-heap) — эта область используется JVM, когда включена JIT-компиляция, в ней кешируется скомпилированный платформенно-зависимый код.

JVM самостоятельно осуществляет сборку так называемого мусора, что значительно облегчает работу программиста по отслеживанию утечек памяти, но важно помнить, что в Java утечки памяти всё равно существуют, особенно при программировании многопоточных приложений.

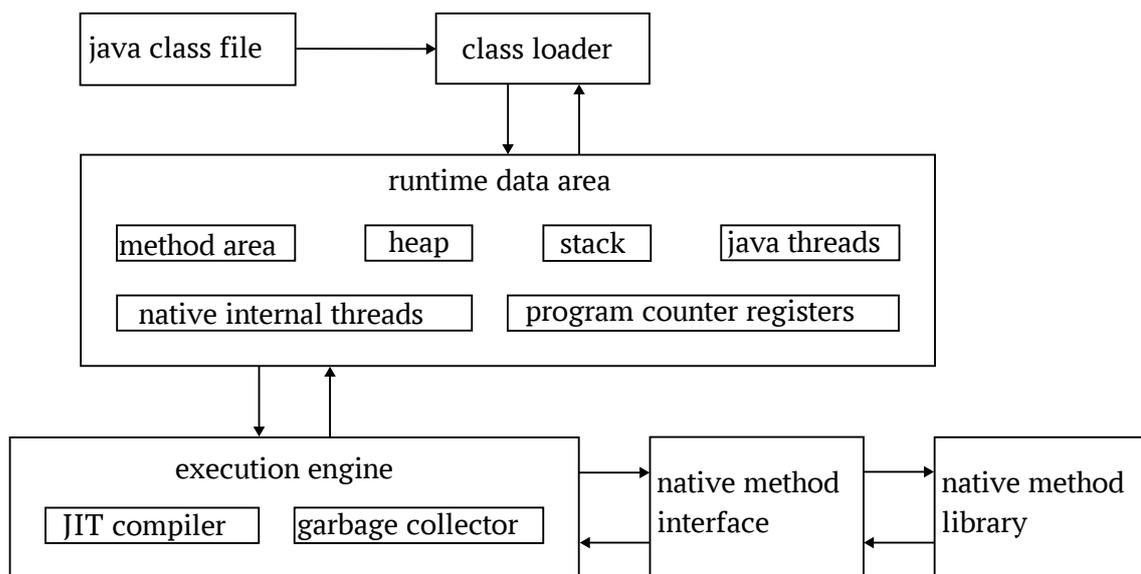


Рис. 1.2: принцип работы JVM

На пользовательском уровне важно не только исполнять базовые инструкции программы, но чтобы эти базовые инструкции умели как-то взаимодействовать со внешним миром, в том числе другими программами, поэтому JVM интегрирована в JRE - Java Runtime Environment. JRE - это набор из классов и интерфейсов, реализующих

- возможности сетевого взаимодействия;
- рисование графики и графический пользовательский интерфейс;
- мультимедиа;
- математический аппарат;
- наследование и полиморфизм;
- рефлексию;
- ... многое другое.

Java Development Kit является изрядно дополненным специальными Java приложениями SDK. JDK дополняет JRE не только утилитами для компиляции, но и утилитами для создания документации, отладки, развёртывания приложений и многими другими. В таблице 1.1 на странице 5, приведена примерная структура и состав JDK и JRE, а также указаны их основные и наиболее часто

используемые компоненты из состава Java Standard Edition. Помимо стандартной редакции существует и Enterprise Edition, содержащий компоненты для создания веб-приложений, но JEE активно вытесняется фреймворками Spring и Spring Boot.

Language										
tools + tools api	javac	java	javadoc	javap	jar	JPDA				
	JConsole	Java VisualVM	JMC	JFR	Java DB	Int'l	JVM TI			
deployment	IDL	Troubleshoot	Security	RMI	Scripting	Web services	Deploy			
	Java Web	Applet/Java plug-in								
UI toolkit	Swing	Java 2D		AWT		Accessibility				
	Drag'n'Drop	Input Methods		Image I/O		Print Service		Sound		
Integration libraries	IDL	JDBC	JNDI	RMI	RMI-IIOP		Scripting			
	Override Mechanism	Intl Support		Input/Output		JMX				
Other base libraries	XML JAXP	Math		Networking		Beans				
	Security	Serialization		Extension Mechanism		JNI				
Java lang and util base libs	JAR	Lang and util	Ref Objects	Preference API		Reflection				
	Zip	Management	Instrumentation	Stream API		Collections				
JVM	Logging	Regular Expressions	Concurrency Utilities	Datetime		Versioning				
	Java Hot Spot VM (JIT)									
Java Standard Edition										
Java Runtime Environment										
Java Development Kit										

Таблица 1.1: Общее представление состава JDK

**Структура проекта (пакеты, классы, метод main, комментарии)**

Проекты могут быть любой сложности. Часто структуру проекта задаёт сборщик проекта, предписывая в каких папках будут храниться исходные коды, исполняемые файлы, ресурсы и докумен-



тация. Без их использования необходимо задать структуру самостоятельно.

**Простейший проект** чаще всего состоит из одного файла исходного кода, который возможно скомпилировать и запустить как самостоятельный объект. Отличительная особенность в том, что чаще всего это один или несколько статических методов в одном классе.

Файл `Main.java` в этом случае может иметь следующий, минималистичный вид

```
1 public class Main {
2     public static void main(String[] args) {
3         System.out.println("Hello, world!");
4     }
5 }
```

**Скриптовый проект** это достаточно новый тип проектов, он получил развитие благодаря растущей популярности Jupyter Notebook. Скриптовые проекты удобны, когда нужно отработать какую-то небольшую функциональность или пошагово пояснить работу какого-то алгоритма.

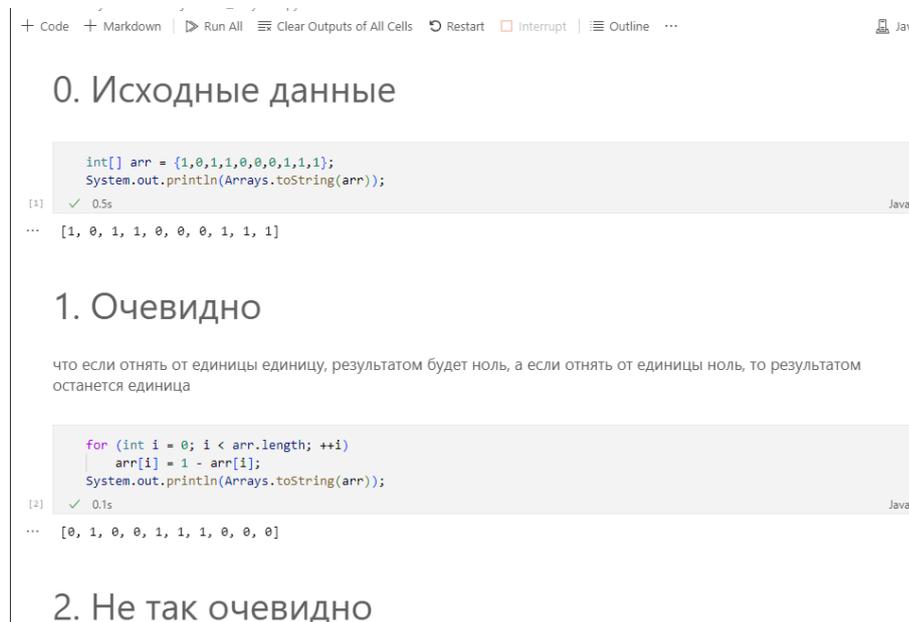


Рис. 1.3: Пример простого Java проекта в Jupyter Notebook

**Обычный проект** состоит из пакетов, которые содержат классы, которые в свою очередь как-то связаны между собой и содержат код, который исполняется.

- **Пакеты.** Пакеты объединяют классы по смыслу. Классы, находящиеся в одном пакете доступны друг другу даже если находятся в разных проектах. У пакетов есть правила именования: обычно это обратное доменное имя (например, для `gb.ru` это будет `ru.gb`), название проекта, и далее уже внутренняя структура. Пакеты именуют строчными латинскими буквами. Чтобы явно отнести класс к пакету, нужно прописать в классе название пакета после оператора `package`.
- **Классы.** Основная единица исходного кода программы. Одному файлу следует сопоставлять один класс. Название класса - это имя существительное в именительном падеже, написанное с заглавной буквы. Если требуется назвать класс в несколько слов, применяют `UpperCamelCase`.
- `public static void main(String[] args).` Метод, который является точкой входа в программу. Должен находиться в публичном классе. При создании этого метода важно полностью повторить его сигнатуру и обязательно написать его с названием со строчной буквы.
- **Комментарии.** Это часть кода, которую игнорирует компилятор при преобразовании исходного кода. Комментарии бывают:
  - `// comment` - до конца строки. Самый простой и самый часто используемый комментарий.

- `/* comment */` - внутристрочный или многострочный. Никогда не используйте его внутри строк, несмотря на то, что это возможно.
- `/** comment */` - комментарий-документация. Многострочный. Из него утилитой Javadoc создаётся веб-страница с комментарием.

Для примера был создан проект, содержащий два класса, находящихся в разных пакетах. Дерево проекта представлено на рис. 1.1, где папка с выходными (скомпилированными) бинарными файлами пуста, а файл README.md создан для лучшей демонстрации корня проекта.

### Sample

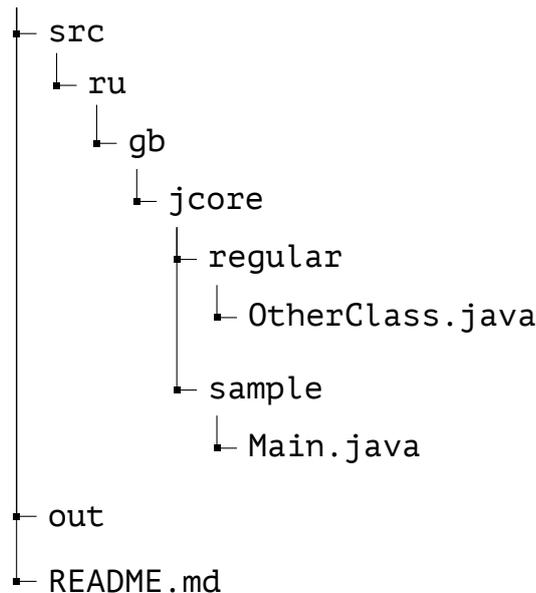


Рис. 1.4: Структура простого проекта

Содержимое файлов исходного кода представлено ниже.

```
1 package ru.gb.jcore.sample;
2
3 import ru.gb.jcore.regular.OtherClass;
4
5 public class Main {
6     public static void main(String[] args) {
7         System.out.println("Hello, world!"); // greetings
8         int result = OtherClass.sum(2, 2); // using a class from other package
9         System.out.println(OtherClass.decorate(result));
10    }
11 }
```

```
1 package ru.gb.jcore.regular;
2
3 public class OtherClass {
4     public static int sum(int a, int b) {
5         return a + b; // return without overflow check
6     }
7
8     public static String decorate(int a) {
9         return String.format("Here is your number: %d.", a);
10    }
11 }
```

### Отложим мышки в сторону (CLI: сборка, пакеты, запуск)

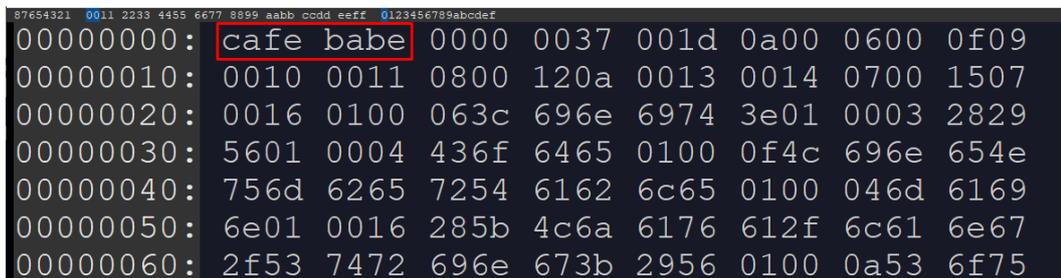
Простейший проект возможно скомпилировать и запустить без использования тяжеловесных сред разработки, введя в командной строке ОС две команды:



- `javac <Name.java>` скомпилирует файл исходников и создаст в этой же папке файл с байт-кодом;
- `java Name` запустит скомпилированный класс (из файла с расширением `.class`).

```
1 ivan-igorevich@gb sources % ls
2 Main.java
3 ivan-igorevich@gb sources % javac Main.java
4 ivan-igorevich@gb sources % ls
5 Main.class Main.java
6 ivan-igorevich@gb sources % java Main
7 Hello, world!
```

Скомпилированные классы всегда содержат одинаковые первые четыре байта, которые в шестнадцатиричном представлении формируют надпись «кофе, крошка».



```
87654321 0011 2233 4455 6677 8899 aabb codd eeff 0123456789abcdef
00000000: cafe babe 0000 0037 001d 0a00 0600 0f09
00000010: 0010 0011 0800 120a 0013 0014 0700 1507
00000020: 0016 0100 063c 696e 6974 3e01 0003 2829
00000030: 5601 0004 436f 6465 0100 0f4c 696e 654e
00000040: 756d 6265 7254 6162 6c65 0100 046d 6169
00000050: 6e01 0016 285b 4c6a 6176 612f 6c61 6e67
00000060: 2f53 7472 696e 673b 2956 0100 0a53 6f75
```

Для компиляции более сложных проектов, необходимо указать компилятору, откуда забирать файлы исходников и куда складывать готовые файлы классов, а интерпретатору, откуда забирать файлы скомпилированных классов. Для этого существуют следующие ключи:

- `javac`:
  - `-d` выходная папка (директория) назначения;
  - `-sourcepath` папка с исходниками проекта;
- `java`:
  - `-classpath` папка с классами проекта;

Классы проекта компилируются в выходную папку с сохранением иерархии пакетов.

```
1 ivan-igorevich@gb Sample % javac -sourcepath ./src -d out src/ru/gb/jcore/sample/Main.java
2 ivan-igorevich@gb Sample % java -classpath ./out ru.gb.jcore.sample.Main
3 Hello, world!
4 Here is your number: 4.
```

## Документирование (Javadoc)

Документирование конкретных методов и классов всегда ложится на плечи программиста, потому что никто не знает программу и алгоритмы в ней лучше, чем программист. Утилита Javadoc избавляет программиста от необходимости осваивать инструменты создания веб-страниц и записывать туда свою документацию. Достаточно писать хорошо отформатированные комментарии, а остальное Javadoc возьмёт на себя.

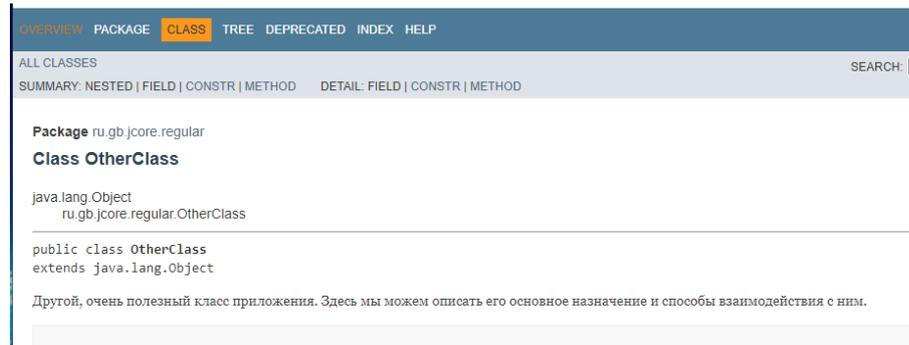


Рис. 1.5: Часть страницы автосгенерированной документации

Чтобы просто создать документацию надо вызвать утилиту `javadoc` с набором ключей.

- `ru` пакет, для которого нужно создать документацию;
- `-d` папка (или директория) назначения;
- `-sourcerath` папка с исходниками проекта;
- `-cp` путь до скомпилированных классов;
- `-subpackages` нужно ли заглядывать в пакеты-с-пакетами;
- `-locale ru_RU` язык документации (для правильной расстановки переносов и разделяющих знаков);
- `-encoding` кодировка исходных текстов программы;
- `-docencoding` кодировка конечной сгенерированной документации.

Чаще всего в комментариях используются следующие ключевые слова:

- `@param` описание входящих параметров
- `@throws` выбрасываемые исключения
- `@return` описание возвращаемого значения
- `@see` где ещё можно почитать по теме
- `@since` с какой версии продукта доступен метод
- `{@code "public"}` вставка кода в описание

## Задания к семинару

- Создать проект из трёх классов (основной с точкой входа и два класса в другом пакете), которые вместе должны составлять одну программу, позволяющую производить четыре основных математических действия и осуществлять форматированный вывод результатов пользователю.
- Скомпилировать проект, а также создать для этого проекта стандартную веб-страницу с документацией ко всем пакетам.

## Сценарий лекции

### 1.2. Управление проектом: сборщики проектов

Управление проектом: Jar-файлы; Gradle/Maven: зависимости, выгрузка артефакта, публичные репозитории, свойства проекта, приватные репозитории (хостинг); Bazel

## Задания к семинару

- Создать загружаемый артефакт с функцией суммирования двух чисел и загрузить его в локальный кэш артефактов;



## 1.3. Специализация: данные и функции

Базовые функции языка: математические операторы, условия, циклы, бинарные операторы; Данные: типы, преобразование типов, константы и переменные (примитивные, ссылочные), бинарное представление, массивы (ссылочная природа массивов, индексация, манипуляция данными); Функции: параметры, возвращаемые значения, перегрузка функций;

### 1.3.1. Данные

Хранение данных в Java осуществляется привычным для программиста образом: в переменных и константах. Языки программирования бывают типизированными и нетипизированными (бестиповыми).

Отсутствие типизации в основном присуще старым и низкоуровневым языкам программирования, например, Forth, некоторые ассемблеры. Все данные в таких языках считаются цепочками бит произвольной длины и, как следует из названия, не делятся на типы. Работа с ними часто труднее, и при чтении кода не всегда ясно, о каком типе переменной идет речь. При этом часто безтиповые языки работают быстрее типизированных, но описывать с их помощью большие проекты со сложными взаимосвязями довольно утомительно.

Java является языком со строгой (сильной) явной статической типизацией.

Что это значит?

- Статическая - у каждой переменной должен быть тип и мы этот тип поменять не можем. Этому свойству противопоставляется динамическая типизация;
- Явная - при создании переменной мы должны ей обязательно присвоить какой-то тип, явно написав это в коде. Бывают языки с неявной типизацией, например, Python;
- Строгая(сильная) - невозможно смешивать разнотипные данные. С другой стороны, существует JavaScript, в котором запись `2 + true` выдаст результат 3.

Все данные в Java делятся на две основные категории: примитивные и ссылочные.

Данные: типы, преобразование типов, константы и переменные (примитивные, ссылочные), бинарное представление, массивы (ссылочная природа массивов, индексация, манипуляция данными);

Тип	Пояснение	Диапазон
byte	Самый маленький из адресуемых типов, 8 бит, знаковый	[-128, +127]
short	Тип короткого целого числа, 16 бит, знаковый	[-32 768, +32 767]
char	Целочисленный тип для хранения символов в кодировке UTF-8, 16 бит, беззнаковый	[0, +65 535]
int	Основной тип целого числа, 32 бита, знаковый	[-2 147 483 648, +2 147 483 647]
long	Тип длинного целого числа, 64 бита, знаковый	[-9 223 372 036 854 775 808, +9 223 372 036 854 775 807]
float	Тип вещественного числа с плавающей запятой (одинарной точности, 32 бита)	
double	Тип вещественного числа с плавающей запятой (двойной точности, 64 бита)	
boolean	Логический тип данных	true, false

Рис. 1.6: Основные типы данных в языке C



Базовые функции языка: математические операторы, условия, циклы, бинарные операторы;  
Функции: параметры, возвращаемые значения, перегрузка функций;

### Антипаттерн ”магические числа”

В прошлом примере мы использовали антипаттерн - плохой стиль для написания кода. Число 18 используется в коде без пояснений. Такой антипаттерн называется ”магическое число”. Рекомендуется помещать числа в константы, которые хранятся в начале файла. `ADULT = 18` `age = float(input('Ваш возраст: '))` `how_old = age - ADULT` `print(how_old, "лет назад ты стал совершеннолетним")`

Плюсом такого подхода является возможность легко корректировать большие проекты. Представьте, что в вашем коде несколько тысяч строк, а число 18 использовалось несколько десятков раз. □ При развертывании проекта в стране, где совершеннолетием считается 21 год вы будете перечитывать весь код в поисках магических ”18” и править их на ”21”. В случае с константой изменить число нужно в одном месте. □ Дополнительной сложности могут возникнуть, если в коде будет 18 как возраст совершеннолетия и 18 как коэффициент для расчёта чего-либо. Теперь править код ещё сложнее, ведь возраст изменился, а коэффициент -нет. В случае с сохранением значений в константы мы снова меняем число в одном месте.

### Задания к семинару

- Написать как можно больше вариантов функции инвертирования массива единиц и нулей за 15 минут (без ветвлений любого рода);
- Сравнить без условий две даты, представленные в виде трёх чисел гггг-мм-дд;

## 1.4. Специализация: ООП

Инкапсуляция: Классы и объекты (внутренние классы, вложенные классы, `static`, `private/public`, `final`, интерфейс взаимодействия с объектом), перечисления (создание, конструкторы перечислений, объекты перечислений, дополнительные свойства); Наследование: `extends`, `Object` (глобальное наследование), `protected`, преобразование типов, `final`; Полиморфизм: `override`, `abstract`, `final`;

## 1.5. Специализация: Тонкости работы

Файловая система и представление данных; Пакеты `java.io`, `java.nio`, `String`, `StringBuilder`, `string pool`, ?JSON/XML?

## Глава 2

# Java Development Kit

### 2.1. Исключения

Механизм и понятие, введение в **многопоточность**, throw; Наследование и полиморфизм в исключениях; Способы обработки исключений (try/catch, throws, finally); try-with-resources, штатные и нештатные ситуации

### 2.2. Интерфейсы

Понятие и принцип работы, implements; Наследование и множественное наследование интерфейсов; Значения по-умолчанию

### 2.3. Обобщённое программирование

### 2.4. Коллекции

List, Set; Хэш-код, интерфейс Comparable; Map, Object (Использование методов, Переопределение методов); Итераторы, **Многопоточные** коллекции

### 2.5. Многопоточность

Понятие, Принцип (реальная и псевдопараллельность); Runnable, Thread (Свойства, Особенности создания); Deadlock, Состояние гонки, Object (Ожидание/уведомление); Синхронизация (Синхронизация по монитору, Частичная, по классу)

### 2.6. Графический интерфейс пользователя

GUI (Swing): Окна и свойства окон, **Многопоточность** и абстрагирование асинхронных вызовов; менеджеры размещений и проведение параллелей с веб-фреймворками, разделение на фронт-энд и бэк-энд; JPanel и рисование, Обработка действий пользователя

## Глава 3

# Java Junior

### 3.1. JDBC

### 3.2. Сетевое программирование

Socket, ServerSocket, Многопоточность, абстрагирование сетевого взаимодействия, интерфейсы

### 3.3. Введение в архитектуры приложений на Java

клиент-серверы, веб-приложения, сервлеты, толстые и тонкие клиенты, выделение бизнес-логики и хранения, создание общих пространств и модульность проекта

### 3.4. Сериализация, StreamAPI, ReflectionAPI

### 3.5. Введение в Java EE

### 3.6. Введение в Spring framework

## Глава 4

### Java Junior+



# Приложения

## А. Термины, определения и сокращения

- **UTF-8** (от англ. Unicode Transformation Format, 8-bit — «формат преобразования Юникода, 8-бит») — распространённый стандарт кодирования символов, позволяющий более компактно хранить и передавать символы Юникода, используя переменное количество байт (от 1 до 4), и обеспечивающий полную обратную совместимость с 7-битной кодировкой ASCII. Кодировка UTF-8 сейчас является доминирующей в веб-пространстве. Она также нашла широкое применение в UNIX-подобных операционных системах.
- Операционная система, сокр. **ОС** (англ. operating system, OS) — комплекс управляющих и обрабатывающих программ, которые, с одной стороны, выступают как интерфейс между устройствами вычислительной системы и прикладными программами, а с другой стороны — предназначены для управления устройствами, управления вычислительными процессами, эффективного распределения вычислительных ресурсов между вычислительными процессами и организации надёжных вычислений.
- **IDE** (Integrated Development Environment) – это интегрированная, единая среда разработки, которая используется разработчиками для создания различного программного обеспечения. IDE представляет собой комплекс из нескольких инструментов, а именно: текстового редактора, компилятора или интерпретатора, средств автоматизации сборки и отладчика.