

Специализация Java

Иван Игоревич Овчинников

2022-08-15 (00:54)

Оглавление

1	Java Core	2
1.1	Платформа: история и окружение	2
1.2	Управление проектом: сборщики проектов	6
1.3	Специализация: данные и функции	6
1.4	Специализация: ООП	7
1.5	Специализация: Тонкости работы	7
2	Java Development Kit	8
2.1	Исключения	8
2.2	Интерфейсы	8
2.3	Обобщённое программирование	8
2.4	Коллекции	8
2.5	Многопоточность	8
2.6	Графический интерфейс пользователя	8
3	Java Junior	9
3.1	JDBC	9
3.2	Сетевое программирование	9
3.3	Введение в архитектуры приложений на Java	9
3.4	Сериализация, StreamAPI, ReflectionAPI	9
3.5	Введение в Java EE	9
3.6	Введение в Spring framework	9
4	Java Junior+	10

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 нацеленности на сетевые приложения.

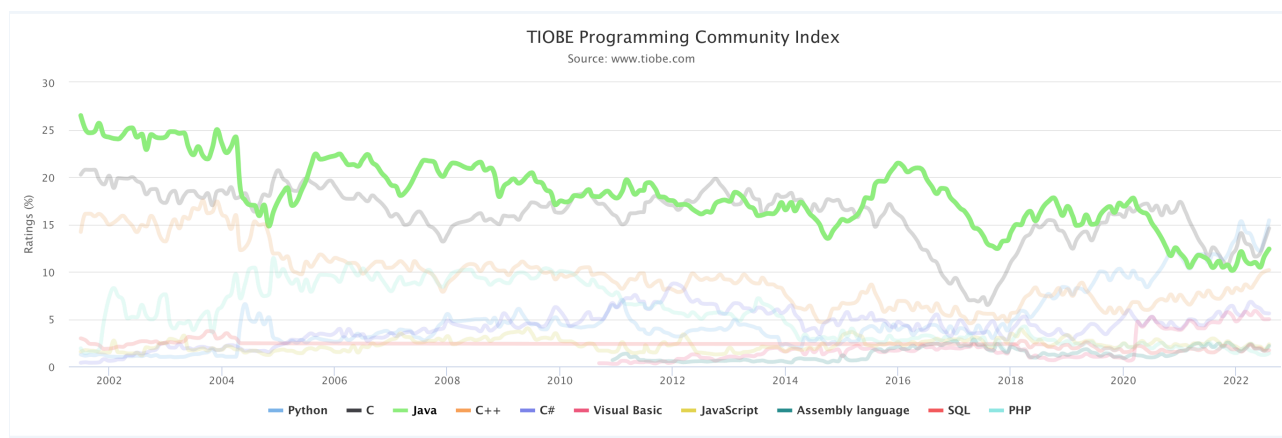


Рис. 1.1: График популярности языков программирования TIOBE

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, 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 JIT¹-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

¹JIT, 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.

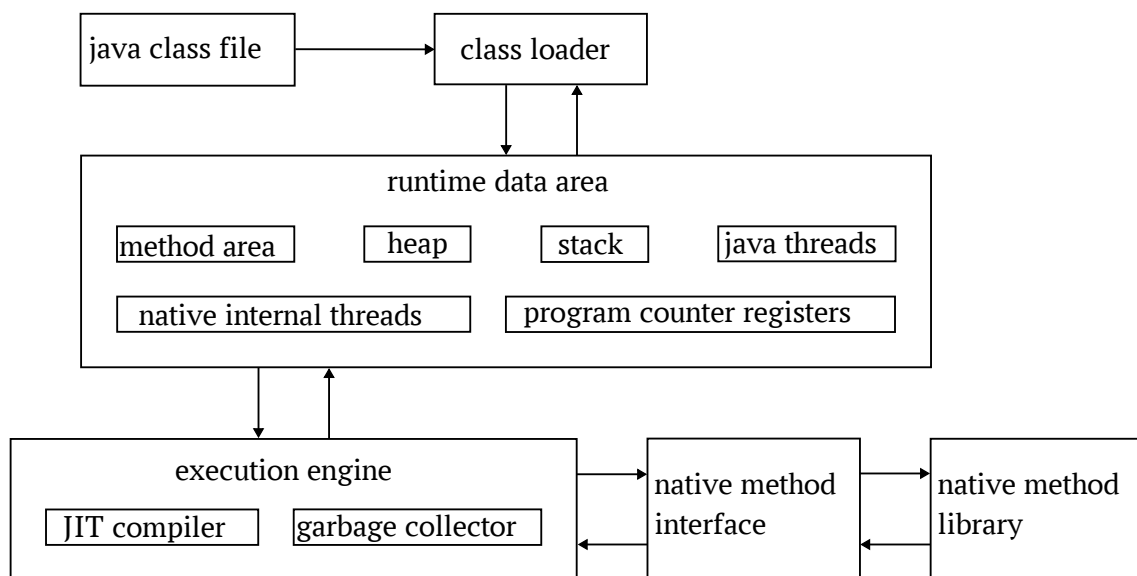


Рис. 1.2: принцип работы JVM

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 состоит из файла исходного кода, который возможно скомпилировать и запустить как самостоятельный объект.

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 }
```

А скомпилировать и запустить его возможно без использования тяжеловесных сред разработки, введя в командной строке ОС две команды.

```
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!
```

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

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

Документирование (Javadoc)

Задания к семинару

- Скомпилировать проект из трёх классов, находящихся в двух пакетах, а также создать для этого проекта стандартную веб-страницу с документацией

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;

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+

