

# 1 □ Конспекты курса: “Продвинутая Java Платформа”

## 1.1 Лектор: Александр Маторин

Кафедра БИТ, МФТИ

Ведётся студентом: [Андрей Бодакин](#)

□ [Скачать полный PDF со всеми лекциями](#) — генерируется автоматически через GitHub Actions при каждом обновлении.

---

## 1.2 □ Описание курса

Курс посвящён глубокому изучению языка **Java**, начиная с основ синтаксиса и заканчивая продвинутыми темами: многопоточность, JVM, коллекции, работа с памятью и многое другое.

Лектор: **Александр Маторин** — практик, эксперт в Java-экосистеме.

Цель репозитория — систематизировать знания, вести конспекты лекций, делиться материалами и примерами кода.

---

## 1.3 □ Оглавление

### 1.3.1 □ Лекции

- [Лекция 1](#) — Основы синтаксиса Java + Настройка окружения и работа в IntelliJ IDEA
  - [Лекция 2](#) — Прimitives типы, классы-обёртки, Пакеты Java, Object, equals/hashCode
  - [Лекция 3](#) — Структуры данных в Java
  - Следующие лекции будут добавляться по мере прохождения курса...
- 

## 1.4 □ Как использовать

- Все конспекты в формате **Markdown** — легко читать на GitHub.
  - Примеры кода — в папке `code/` (если есть).
  - Pull Request'ы и Issues приветствуются — если нашли ошибку или хотите дополнить материал.
- 

## 1.5 □ Сотрудничество

Если ты тоже учишься на курсе — присылай свои конспекты, дополнения, примеры кода! Открыт для совместного ведения и улучшения материалов.

---

## 1.6 □ Лицензия

Этот репозиторий распространяется под лицензией **MIT** — используйте свободно для обучения и распространения знаний.

---

□ “Знания растут, когда ими делишься.”

## 2 □ Лекция 1 — Основы синтаксиса Java + Настройка окружения и работа в IntelliJ IDEA

---

### 2.1 □ Основные темы

- Установка и настройка JDK
  - Компиляция и запуск через `javac` и `java`
  - Переменные среды: `PATH`, `CLASSPATH`
  - Выбор и настройка IDE (IntelliJ IDEA)
  - Горячие клавиши и рефакторинги в IntelliJ IDEA
  - Основы синтаксиса: классы, методы, переменные, управляющие конструкции
- 

### 2.2 □ Настройка Java-окружения

#### 2.2.1 Установка JDK

→ Скачать можно с: - [Oracle JDK](#) - [OpenJDK \(Adoptium / Temurin\)](#) ← **рекомендуется**  
Проверка установки:

```
\ExtensionTok{java} \AttributeTok{{-}version}  
\ExtensionTok{javac} \AttributeTok{{-}version}
```

→ Должны вывести версию Java и компилятора.

---

#### 2.2.2 Переменная окружения PATH

→ `PATH` — список директорий, где система ищет исполняемые файлы.

□ Добавить путь к `bin` JDK в `PATH`:

**Linux/macOS** (в `~/.bashrc` или `~/.zshrc`):

```
\BuiltInTok{export} \VariableTok{PATH}\OperatorTok{=}\StringTok{"/path/to/jdk/bin:}\VariableTok{$PATH}\StringTok{"}
```

**Windows:** - Панель управления → Система → Дополнительные параметры → Переменные среды → `PATH`

→ Добавить путь, например:

`C:\Program Files\Java\jdk-21\bin`

---

#### 2.2.3 CLASSPATH

→ Указывает JVM, где искать `.class`-файлы и библиотеки.

□ Обычно не нужно настраивать вручную при работе с IDE или Maven/Gradle.

→ Если компилируешь вручную:

```
\ExtensionTok{javac} \AttributeTok{{-}cp} \StringTok{".;lib/*"}\NormalTok{ MyClass.java}
\ExtensionTok{java} \AttributeTok{{-}cp} \StringTok{".;lib/*"}\NormalTok{ MyClass}
```

□ . — текущая директория.

□ ; — разделитель в Windows, : — в Linux/macOS.

## 2.3 □ Работа с IntelliJ IDEA

### 2.3.1 Почему IntelliJ IDEA?

- Самая популярная и мощная IDE для Java.
- Умное автодополнение, рефакторинги, отладка, интеграция с Maven/Gradle, Git.
- Community Edition — бесплатна и достаточно для обучения.

## 2.4 🖱️ Горячие клавиши IntelliJ IDEA (must-have)

Действие	Windows/Linux	macOS
Автодополнение	Ctrl + Space	Cmd + Space
Быстрое исправление / подсказки	Alt + Enter	Option + Enter
Запуск программы	Shift + F10	Ctrl + R
Отладка	Shift + F9	Ctrl + D
Поиск по проекту	Ctrl + Shift + F	Cmd + Shift + F
Поиск класса	Ctrl + N	Cmd + O
Поиск файла	Ctrl + Shift + N	Cmd + Shift + O
Переход к определению	Ctrl + B	Cmd + B
Рефакторинг: переименование	Shift + F6	Shift + F6
Закомментировать строку	Ctrl + /	Cmd + /
Форматирование кода	Ctrl + Alt + L	Cmd + Option + L
Открыть структуру класса	Ctrl + F12	Cmd + F12

## 2.5 □ Полезные Live Templates (шаблоны кода)

Шаблон	Результат	Описание
sout	System.out.println();	Быстрый вывод в консоль
iter	for (Type item : collection) { }	Цикл for-each
psvm	public static void main(String[] args) { }	Главный метод
itar	for (int i = 0; i < arr.length; i++) { }	Цикл по индексу
ifn	if (var == null) { }	Проверка на null
inn	if (var != null) { }	Проверка на не-null

→ Просто введи шаблон и нажми Tab.

## 2.6 □ Рефакторинги в IntelliJ IDEA

### 2.6.1 □ Extract Method (Вынести метод)

Выдели код → Ctrl + Alt + M → дай имя методу → готово!

**Было:**

```
\KeywordTok{public} \DataTypeTok{void} \FunctionTok{process}\OperatorTok{()} \OperatorTok{\{\}
  \DataTypeTok{int}\NormalTok{ a }\OperatorTok{=} \DecValTok{5}\OperatorTok{;}
  \DataTypeTok{int}\NormalTok{ b }\OperatorTok{=} \DecValTok{10}\OperatorTok{;}
  \DataTypeTok{int}\NormalTok{ sum }\OperatorTok{=}\NormalTok{ a }\OperatorTok{+}\NormalTok{ b}\OperatorTok{;}
  \BuiltInTok{System}\OperatorTok{.}\FunctionTok{out}\OperatorTok{.}\FunctionTok{println}\OperatorTok{()}\StringTok{"Sum: "}
\OperatorTok{\}}
```

**Стало:**

```
\KeywordTok{public} \DataTypeTok{void} \FunctionTok{process}\OperatorTok{()} \OperatorTok{\{\}
  \FunctionTok{printSum}\OperatorTok{()}\DecValTok{5}\OperatorTok{,}\DecValTok{10}\OperatorTok{);}
\OperatorTok{\}}

\KeywordTok{private} \DataTypeTok{void} \FunctionTok{printSum}\OperatorTok{()}\DataTypeTok{int}\NormalTok{ a}\OperatorTok{,}
  \DataTypeTok{int}\NormalTok{ sum }\OperatorTok{=}\NormalTok{ a }\OperatorTok{+}\NormalTok{ b}\OperatorTok{;}
  \BuiltInTok{System}\OperatorTok{.}\FunctionTok{out}\OperatorTok{.}\FunctionTok{println}\OperatorTok{()}\StringTok{"Sum: "}
\OperatorTok{\}}
```

→ Улучшает читаемость и переиспользование.

---

### 2.6.2 □ Inline Method (Встроить метод)

Если метод слишком простой — можно “встроить” его обратно:

Ctrl + Alt + N

→ Полезно при оптимизации или упрощении.

---

## 2.7 □ Основы синтаксиса Java

### 2.7.1 Структура программы

```
\KeywordTok{public} \KeywordTok{class}\NormalTok{ HelloWorld }\OperatorTok{\{\}
  \KeywordTok{public} \DataTypeTok{static} \DataTypeTok{void} \FunctionTok{main}\OperatorTok{()}\BuiltInTok{String}\OperatorTok{ }
  \BuiltInTok{System}\OperatorTok{.}\FunctionTok{out}\OperatorTok{.}\FunctionTok{println}\OperatorTok{()}\StringTok{"Hello World"}
\OperatorTok{\}}
\OperatorTok{\}}
```

→ Каждая программа начинается с main.

---

### 2.7.2 Переменные и типы

```
\DataTypeTok{int}\NormalTok{ age }\OperatorTok{=} \DecValTok{25}\OperatorTok{;}  
\DataTypeTok{double}\NormalTok{ price }\OperatorTok{=} \FloatTok{19.99}\OperatorTok{;}  
\DataTypeTok{boolean}\NormalTok{ isActive }\OperatorTok{=} \KeywordTok{true}\OperatorTok{;}  
\BuiltInTok{String}\NormalTok{ name }\OperatorTok{=} \StringTok{"Alice"}\OperatorTok{;}
```

---

### 2.7.3 Управляющие конструкции

```
\ControlFlowTok{if} \OperatorTok{{}}\NormalTok{age }\OperatorTok{{}}\textgreater{} \DecValTok{18}\OperatorTok{{}} \OperatorTok{{}}  
  \BuiltInTok{System}\OperatorTok{{}}\FunctionTok{out}\OperatorTok{{}}\FunctionTok{println}\OperatorTok{{}}\StringTok{"Совер"}  
\OperatorTok{{}} \ControlFlowTok{else} \OperatorTok{{}}  
  \BuiltInTok{System}\OperatorTok{{}}\FunctionTok{out}\OperatorTok{{}}\FunctionTok{println}\OperatorTok{{}}\StringTok{"Несове"}  
\OperatorTok{{}}  
  
\ControlFlowTok{for} \OperatorTok{{}}\DataTypeTok{int}\NormalTok{ i }\OperatorTok{=} \DecValTok{0}\OperatorTok{;}\NormalTok{{}}  
  \BuiltInTok{System}\OperatorTok{{}}\FunctionTok{out}\OperatorTok{{}}\FunctionTok{println}\OperatorTok{{}}\NormalTok{i}\OperatorTok{{}}  
\OperatorTok{{}}  
  
\ControlFlowTok{while} \OperatorTok{{}}\NormalTok{condition}\OperatorTok{{}} \OperatorTok{{}}  
  \CommentTok{// ...}  
\OperatorTok{{}}
```

---

## 2.8 ☐ Советы для новичков

- Всегда проверяй, что `java -version` работает в терминале.
  - Не бойся использовать `Alt + Enter` — IntelliJ IDEA часто знает, как исправить ошибку.
  - Учись пользоваться рефакторингами — они экономят кучу времени.
- 

## 2.9 ☐ Полезные ссылки

- [Скачать IntelliJ IDEA Community](#)
- [OpenJDK \(Adoptium\)](#)
- [Горячие клавиши IntelliJ IDEA \(официальная шпаргалка\)](#)
- [Теория по Java](#)

# ☐ Лекция 2 — Примитивные типы, классы-обёртки, Пакеты Java, Object, equals/hashCode

---

## ☐ Основные темы

- Прimitives типы данных в Java (включая размер в битах)
- Классы-обёртки (Wrapper Classes)
- Автобоксинг и автораспаковка
- Передача параметров по значению
- Пакеты: `java.lang`, `java.util`, `java.io`
- Класс `Object` – корень иерархии
- Контракт `equals()` и `hashCode()`
- Почему важно переопределять их вместе
- Переполнение (overflow) примитивных типов
- `BigDecimal` – точные вычисления (для денег)
- `StringBuilder` – эффективная работа со строками
- Практические примеры и антипаттерны

---

## ## □ Прimitives типы данных

В Java 8 примитивных типов. Размер указан в **\*\*битах и байтах\*\***.

Тип	Размер (биты)	Размер (байты)	Диапазон значений	Значение по умолчанию
-----	-----	-----	-----	-----
`byte`	8 бит	1 байт	-128 до 127	`0`
`short`	16 бит	2 байта	-32768 до 32767	`0`
`int`	32 бита	4 байта	-2 <sup>31</sup> до 2 <sup>31</sup> -1	`0`
`long`	64 бита	8 байт	-2 <sup>63</sup> до 2 <sup>63</sup> -1	`0L`
`float`	32 бита	4 байта	±3.4e38 (7 знаков)	`0.0f`
`double`	64 бита	8 байт	±1.7e308 (15 знаков)	`0.0d`
`char`	16 бит	2 байта	`\u0000` до `\uffff` (Unicode)	`\u0000`
`boolean`	не определён*	не определён*	`true` / `false`	`false`

> □ `\*` – размер `boolean` не определён спецификацией JVM – зависит от реализации. Обычно хранится как `int` (32 бита)

---

## ## □ Классы-обёртки (Wrapper Classes)

Каждый примитив имеет объектный аналог:

Примитив	Класс-обёртка
-----	-----
`int`	`Integer`
`long`	`Long`
`double`	`Double`
`boolean`	`Boolean`
`char`	`Character`
...	...

### Зачем нужны?

- Для хранения в коллекциях (`List<Integer>`, а не `List<int>`).
- Для использования `null`.
- Для вызова методов: `Integer.parseInt()`, `Character.isDigit()` и т.д.

---

## ☐ Автобоксинг и автораспаковка

- **Автобоксинг** — автоматическое преобразование примитива в объект.
- **Автораспаковка** — наоборот.

```
```java
Integer a = 10;           // ← автобоксинг: int → Integer
int b = a;                // ← автораспаковка: Integer → int

List<Integer> list = new ArrayList<>();
list.add(5);              // ← автобоксинг
int first = list.get(0);  // ← автораспаковка
```

☐ **Осторожно с null!**

```
\BuiltInTok{Integer}\NormalTok{ x }\OperatorTok{=} \KeywordTok{null}\OperatorTok{;}
\DataTypeTok{int}\NormalTok{ y }\OperatorTok{=} \NormalTok{ x }\OperatorTok{;} \CommentTok{// ← NullPointerException!}
```

- Всегда проверяй на `null` перед распаковкой.

---

## 2.10 ☐ Передача параметров по значению

☐ **В Java всё передаётся по значению — даже объекты!**

- При передаче объекта — копируется **ссылка на объект**, а не сам объект.

### 2.10.1 Пример:

```
\KeywordTok{public} \DataTypeTok{static} \DataTypeTok{void} \FunctionTok{main}\OperatorTok{({}\BuiltInTok{String}\OperatorTok{}}
\NormalTok{    Person p }\OperatorTok{=} \KeywordTok{new} \FunctionTok{Person}\OperatorTok{({}\StringTok{"Alice"}\OperatorTok{}}
    \FunctionTok{changeName}\OperatorTok{({}\NormalTok{p}\OperatorTok{}});
    \BuiltInTok{System}\OperatorTok{.}\FunctionTok{out}\OperatorTok{.}\FunctionTok{println}\OperatorTok{({}\NormalTok{p}\OperatorTok{}}
\OperatorTok{\}}

\DataTypeTok{static} \DataTypeTok{void} \FunctionTok{changeName}\OperatorTok{({}\NormalTok{Person person}\OperatorTok{}}) \Op
\NormalTok{    person}\OperatorTok{.}\FunctionTok{name} \OperatorTok{=} \StringTok{"Bob"}\OperatorTok{;} \CommentTok{// ← и
\OperatorTok{\}}
```

- Объект изменился, потому что мы **изменяли данные по скопированной ссылке**.

### 2.10.2 Но:

```
\DataTypeTok{static} \DataTypeTok{void} \FunctionTok{reassign}\OperatorTok{(\}\NormalTok{Person person}\OperatorTok{)} \OperatorTok{=}\NormalTok{person }\OperatorTok{=}\KeywordTok{new} \FunctionTok{Person}\OperatorTok{(\}\StringTok{"Charlie"}\OperatorTok{)}\OperatorTok{\}
```

→ После вызова `reassign(p)` — `p.name` всё ещё "Alice".

---

## 2.11 Пакеты Java

Пакеты — это пространства имён для классов.

### 2.11.1 Основные пакеты:

- `java.lang` — автоматически импортируется (`String`, `Object`, `System`, `Math`).
- `java.util` — коллекции, дата/время, `Scanner`, `Random`.
- `java.io` — ввод/вывод, файлы.
- `java.time` — современные даты (Java 8+).

### 2.11.2 Импорт:

```
\KeywordTok{import} \ImportTok{java}\OperatorTok{.}\ImportTok{util}\OperatorTok{.}\ImportTok{List}\OperatorTok{;}\KeywordTok{import} \ImportTok{java}\OperatorTok{.}\ImportTok{util}\OperatorTok{.}\ImportTok{ArrayList}\OperatorTok{;}\KeywordTok{import} \ImportTok{java}\OperatorTok{.}\ImportTok{time}\OperatorTok{.}\ImportTok{LocalDate}\OperatorTok{;}\CommentTok{// Или импорт всего пакета:}\KeywordTok{import} \ImportTok{java}\OperatorTok{.}\ImportTok{util}\OperatorTok{.*};
```

□ \* — не нагружает приложение, только упрощает написание кода.

---

## 2.12 Класс `Object` — корень иерархии

→ Корневой класс всей иерархии в Java.

→ Любой класс — наследник `Object` (явно или неявно).

### 2.12.1 Основные методы:

- `toString()` — строковое представление объекта.
  - `equals(Object obj)` — сравнение объектов.
  - `hashCode()` — хеш-код для использования в хеш-таблицах.
  - `getClass()` — получить класс объекта.
  - `clone()` — неглубокое клонирование (осторожно!).
  - `finalize()` — освобождение ресурсов перед удалением, deprecated (Java 9+).
-



## 2.13 1.1.1 equals

```
\KeywordTok{public} \DataTypeTok{boolean} \FunctionTok{equals}\OperatorTok{{}}\AttributeTok{@Nullable} \BuiltInTok{Object}\N
```

□ **Назначение:** проверяет, равны ли два объекта *логически* (по содержимому), а не физически (по ссылке).

### 2.13.1 □ Поведение по умолчанию

→ В классе `Object` метод `equals()` сравнивает **ссылки**:

```
\BuiltInTok{Object}\NormalTok{ a }\OperatorTok{=} \KeywordTok{new} \BuiltInTok{Object}\OperatorTok{();}  
\BuiltInTok{Object}\NormalTok{ b }\OperatorTok{=} \KeywordTok{new} \BuiltInTok{Object}\OperatorTok{();}  
\BuiltInTok{System}\OperatorTok{.}\FunctionTok{out}\OperatorTok{.}\FunctionTok{println}\OperatorTok{({}\NormalTok{a}\OperatorTok{.}}
```

→ Это эквивалентно `a == b`.

---

### 2.13.2 □ Переопределение

→ В подклассах `equals()` **часто переопределяют**, чтобы сравнивать объекты по полям:

```
\AttributeTok{@Override}  
\KeywordTok{public} \DataTypeTok{boolean} \FunctionTok{equals}\OperatorTok{{}}\BuiltInTok{Object}\NormalTok{ o}\OperatorTok{.}  
    \ControlFlowTok{if} \OperatorTok{{}}\KeywordTok{this} \OperatorTok{==}\NormalTok{ o}\OperatorTok{}} \ControlFlowTok{return}  
    \ControlFlowTok{if} \OperatorTok{{}}\NormalTok{o} \OperatorTok{==} \KeywordTok{null} \OperatorTok{||} \FunctionTok{getClas}  
\NormalTok{ {  
    Person person }\OperatorTok{=} \OperatorTok{{}}\NormalTok{Person}\OperatorTok{}}\NormalTok{ o}\OperatorTok{.}  
    \ControlFlowTok{return}\NormalTok{ Objects}\OperatorTok{.}\FunctionTok{equals}\OperatorTok{{}}\NormalTok{name}\OperatorTok{.}  
\NormalTok{ {  
    Objects}\OperatorTok{.}\FunctionTok{equals}\OperatorTok{{}}\NormalTok{age}\OperatorTok{,}\NormalTok{ p  
\OperatorTok{\}}
```

### 2.13.3 □ Контракт `equals()`

Для **ненулевых объектов** метод `equals()` должен задавать **отношение эквивалентности**:

1. **Рефлексивность:** `x.equals(x) → true`
2. **Симметричность:** если `x.equals(y) == true`, то `y.equals(x) == true`
3. **Транзитивность:** если `x.equals(y) == true` и `y.equals(z) == true`, то `x.equals(z) == true`
4. **Консистентность:** если данные объекта не менялись, то `x.equals(y)` должен возвращать одно и то же значение при повторных вызовах.
5. Для **любого ненулевого x**: `x.equals(null) == false`

□ Нарушение контракта → непредсказуемое поведение в коллекциях (`HashSet`, `HashMap` и др.).

---

## 2.14 1.1.2 hashCode

```
\KeywordTok{public} \KeywordTok{native} \DataTypeTok{int} \FunctionTok{hashCode}\OperatorTok{();}
```

□ **Назначение:** возвращает целочисленный хеш-код объекта. Используется в хеш-структурах: `HashMap`, `HashSet`, `HashTable` и др.

---

### 2.14.1 □ Контракт `hashCode()`

1. **Консистентность:** если данные объекта не менялись, `hashCode()` должен возвращать **одно и то же значение** при каждом вызове.
  2. **Согласованность с `equals()`:** если `x.equals(y) == true`, то `x.hashCode() == y.hashCode()` — **обязательно**.
  3. **Необязательное условие:** если `x.equals(y) == false`, то `x.hashCode()` и `y.hashCode()` **могут совпадать** — это называется **коллизия хешей** (нормально для хеш-таблиц).
- 

### 2.14.2 □ Реализация по умолчанию

→ В ранних версиях JVM `hashCode()` возвращал **адрес объекта в памяти**.  
→ **Сейчас** — используется **псевдослучайное число**, которое: - Генерируется при первом вызове `hashCode()`.  
- Записывается в **заголовок объекта** (object header). - **Не меняется** в течение жизни объекта, даже если объект перемещается сборщиком мусора.

□ Почему изменили?

— При маленьком heap-е адреса были близки → хеши были не равномерны → **ухудшалась производительность хеш-таблиц**.

— Новая реализация даёт **лучшее распределение хешей** → меньше коллизий → быстрее работа `HashMap`.

---

### 2.14.3 □ Почему важно переопределять `hashCode()` вместе с `equals()`

→ Представь, что ты положил объект в `HashMap`, а потом изменил поле, участвующее в `equals()`, но не обновил `hashCode()`:

```
\NormalTok{Person p }\OperatorTok{=} \KeywordTok{new} \FunctionTok{Person}\OperatorTok{({}\StringTok{"Alice"}\OperatorTok{)};\n\NormalTok{map}\OperatorTok{.}\FunctionTok{put}\OperatorTok{({}\NormalTok{p}\OperatorTok{,} \StringTok{"value"}\OperatorTok{)};\n\n\NormalTok{p}\OperatorTok{.}\FunctionTok{setName}\OperatorTok{({}\StringTok{"Bob"}\OperatorTok{)}; \CommentTok{// ← изменили\n\n\NormalTok{map}\OperatorTok{.}\FunctionTok{get}\OperatorTok{({}\NormalTok{p}\OperatorTok{)}; \CommentTok{// → null! Потому ч
```

→ **Решение:** поля, используемые в `equals()` и `hashCode()`, должны быть **неизменяемыми (immutable)**.

---

## 2.15 □ Правильное переопределение

### 2.15.1 Способ 1: Вручную

```

\AttributeTok{@Override}
\KeywordTok{public} \DataTypeTok{boolean} \FunctionTok{equals}\OperatorTok{{}}\BuiltInTok{Object}\NormalTok{ o}\OperatorTok{
    \ControlFlowTok{if} \OperatorTok{{}}\KeywordTok{this} \OperatorTok{==}\NormalTok{ o}\OperatorTok{}} \ControlFlowTok{retu
    \ControlFlowTok{if} \OperatorTok{{}}\NormalTok{o } \OperatorTok{==} \KeywordTok{null} \OperatorTok{||} \FunctionTok{getCl
\NormalTok{    Person person } \OperatorTok{=} \OperatorTok{{}}\NormalTok{Person}\OperatorTok{}}\NormalTok{ o}\OperatorTok{;}}
    \ControlFlowTok{return}\NormalTok{ Objects}\OperatorTok{.}\FunctionTok{equals}\OperatorTok{{}}\NormalTok{name}\OperatorTok
\NormalTok{    Objects}\OperatorTok{.}\FunctionTok{equals}\OperatorTok{{}}\NormalTok{age}\OperatorTok{,}\NormalTok{ p
\OperatorTok{\}}

\AttributeTok{@Override}
\KeywordTok{public} \DataTypeTok{int} \FunctionTok{hashCode}\OperatorTok{()} \OperatorTok{\}
    \ControlFlowTok{return}\NormalTok{ Objects}\OperatorTok{.}\FunctionTok{hash}\OperatorTok{{}}\NormalTok{name}\OperatorTok
\OperatorTok{\}}

```

### 2.15.2 Способ 2: Через Lombok (если используется)

```

\AttributeTok{@EqualsAndHashCode}
\KeywordTok{public} \KeywordTok{class}\NormalTok{ Person } \OperatorTok{\}
    \KeywordTok{private} \BuiltInTok{String}\NormalTok{ name} \OperatorTok{;}
    \KeywordTok{private} \BuiltInTok{Integer}\NormalTok{ age} \OperatorTok{;}
\OperatorTok{\}}

```

### 2.15.3 Способ 3: В IntelliJ IDEA → Alt + Insert → equals() and hashCode()

→ IDE сгенерирует корректный код.

```

\AttributeTok{@Override}
\KeywordTok{public} \DataTypeTok{boolean} \FunctionTok{equals}\OperatorTok{{}}\BuiltInTok{Object}\NormalTok{ o}\OperatorTok{
    \ControlFlowTok{if} \OperatorTok{{}}\KeywordTok{this} \OperatorTok{==}\NormalTok{ o}\OperatorTok{}} \ControlFlowTok{retu
    \ControlFlowTok{if} \OperatorTok{{}}\NormalTok{o } \OperatorTok{==} \KeywordTok{null} \OperatorTok{||} \FunctionTok{getCl
\NormalTok{    Person person } \OperatorTok{=} \OperatorTok{{}}\NormalTok{Person}\OperatorTok{}}\NormalTok{ o}\OperatorTok{;}}
    \ControlFlowTok{return}\NormalTok{ Objects}\OperatorTok{.}\FunctionTok{equals}\OperatorTok{{}}\NormalTok{name}\OperatorTok
\NormalTok{    Objects}\OperatorTok{.}\FunctionTok{equals}\OperatorTok{{}}\NormalTok{age}\OperatorTok{,}\NormalTok{ p
\OperatorTok{\}}

\AttributeTok{@Override}
\KeywordTok{public} \DataTypeTok{int} \FunctionTok{hashCode}\OperatorTok{()} \OperatorTok{\}
    \ControlFlowTok{return}\NormalTok{ Objects}\OperatorTok{.}\FunctionTok{hash}\OperatorTok{{}}\NormalTok{name}\OperatorTok
\OperatorTok{\}}

```

□ `Objects.hash(...)` — безопасно обрабатывает null и генерирует хороший хеш на основе переданных полей.

## 2.16 □ Частые ошибки

- Сравнение через `==` для объектов → сравниваются ссылки, а не содержимое.
- Забыли `@Override` → можно случайно создать перегрузку, а не переопределение.
- Не переопределили `hashCode()` → проблемы с `HashMap`.

- Использовали изменяемые поля в hashCode() → объект “сломается” в HashSet, если поле изменится.
  - Использовали float/double в hashCode() без округления → нестабильность из-за точности.
- 

## 2.17 ☐ Советы

- Всегда переопределяй equals() и hashCode() **вместе**.
  - Используй java.util.Objects — безопасно и читаемо.
  - В IntelliJ IDEA: Alt + Insert → генерация методов — экономит время.
  - Тестируй поведение в HashMap — это частый вопрос на собеседованиях.
  - Поля в hashCode() и equals() — лучше делать final.
- 

## 2.18 ☐ Контракт equals() и hashCode()

Если два объекта равны по equals() — их hashCode() **должен быть одинаковым**.

### 2.18.1 Почему это важно?

→ HashMap, HashSet, Hashtable используют hashCode() для определения “корзины”, а equals() — для точного сравнения.

### 2.18.2 Антипаттерн:

```
\AttributeTok{@Override}
\KeywordTok{public} \DataTypeTok{boolean} \FunctionTok{equals}\OperatorTok{{}}\BuiltInTok{Object}\NormalTok{o}\OperatorTok{
    \CommentTok{// ... логика сравнения}
\OperatorTok{}}

\CommentTok{// ☐ Забыли переопределить hashCode()}
```

→ Объекты могут “потеряться” в HashMap.

---

## 2.19 ☐ Частые ошибки

- Сравнение через == для объектов → сравниваются ссылки, а не содержимое.
  - Забыли @Override → можно случайно создать перегрузку, а не переопределение.
  - Не переопределили hashCode() → проблемы с HashMap.
  - Использовали изменяемые поля в hashCode() → объект “сломается” в HashSet, если поле изменится.
- 

## 2.20 ☐ Переполнение (Overflow)

→ Происходит, когда результат операции **выходит за пределы диапазона типа**.

### 2.20.1 Пример с int:

```
\DataTypeTok{int}\NormalTok{ max }\OperatorTok{=} \BuiltInTok{Integer}\OperatorTok{.}\FunctionTok{MAX\_VALUE}\OperatorTok{;}\n\DataTypeTok{int}\NormalTok{ overflow }\OperatorTok{=} \NormalTok{ max }\OperatorTok{+} \DecValTok{1}\OperatorTok{;}\n\BuiltInTok{System}\OperatorTok{.}\FunctionTok{out}\OperatorTok{.}\FunctionTok{println}\OperatorTok{({}\NormalTok{overflow}\OperatorTok{.})}
```

→ Никакого исключения — просто “заворачивается” (как одометр в машине).

### 2.20.2 Как избежать?

- Используй `long` для больших чисел.
- Используй `Math.addExact()`, `Math.multiplyExact()` — бросают `ArithmeticException` при переполнении.

```
\ControlFlowTok{try} \OperatorTok{\}\n\DataTypeTok{int}\NormalTok{ result }\OperatorTok{=} \BuiltInTok{Math}\OperatorTok{.}\FunctionTok{addExact}\OperatorTok{({}\NormalTok{result}\OperatorTok{,}\NormalTok{1}\OperatorTok{)}\n\OperatorTok{\}\ControlFlowTok{catch} \OperatorTok{({}\BuiltInTok{ArithmeticException}\NormalTok{ e}\OperatorTok{)}\OperatorTok{\}\n\BuiltInTok{System}\OperatorTok{.}\FunctionTok{out}\OperatorTok{.}\FunctionTok{println}\OperatorTok{({}\StringTok{"Переполнение"}\OperatorTok{.})}
```

- Для критических вычислений — используй `BigInteger`.

---

## 2.21 BigDecimal — для точных вычислений (деньги!)

→ `float` и `double` **не подходят** для финансовых расчётов — из-за ошибок округления.

### 2.21.1 Пример проблемы:

```
\DataTypeTok{double}\NormalTok{ a }\OperatorTok{=} \FloatTok{0.1}\OperatorTok{;}\n\DataTypeTok{double}\NormalTok{ b }\OperatorTok{=} \FloatTok{0.2}\OperatorTok{;}\n\BuiltInTok{System}\OperatorTok{.}\FunctionTok{out}\OperatorTok{.}\FunctionTok{println}\OperatorTok{({}\NormalTok{a }\OperatorTok{+} \NormalTok{b}\OperatorTok{.})}
```

### 2.21.2 Решение — BigDecimal:

```
\BuiltInTok{BigDecimal}\NormalTok{ a }\OperatorTok{=} \KeywordTok{new} \BuiltInTok{BigDecimal}\OperatorTok{({}\StringTok{"0.1"}\OperatorTok{)}\n\BuiltInTok{BigDecimal}\NormalTok{ b }\OperatorTok{=} \KeywordTok{new} \BuiltInTok{BigDecimal}\OperatorTok{({}\StringTok{"0.2"}\OperatorTok{)}\n\BuiltInTok{BigDecimal}\NormalTok{ sum }\OperatorTok{=} \NormalTok{ a }\OperatorTok{.}\FunctionTok{add}\OperatorTok{({}\NormalTok{b}\OperatorTok{)}\n\BuiltInTok{System}\OperatorTok{.}\FunctionTok{out}\OperatorTok{.}\FunctionTok{println}\OperatorTok{({}\NormalTok{sum}\OperatorTok{.})}
```

 **Всегда создавай `BigDecimal` из `String`, а не из `double`!**

```
\KeywordTok{new} \BuiltInTok{BigDecimal}\OperatorTok{({}\FloatTok{0.1}\OperatorTok{)}} \CommentTok{// ← НЕПРАВИЛЬНО – сохраним\n\KeywordTok{new} \BuiltInTok{BigDecimal}\OperatorTok{({}\StringTok{"0.1"}\OperatorTok{)}} \CommentTok{// ← ПРАВИЛЬНО}
```

### 2.21.3 Операции:

→ Используй `BigDecimal` для: - Денег - Процентов - Точных научных расчётов

→ String в Java **неизменяем (immutable)** → каждая операция "a" + "b" создаёт новый объект.

→ Медленно и расходует память.

→ В **100+ раз быстрее** для больших объёмов.

□ Если нужна потокобезопасность — используй `StringBuffer` (но он медленнее из-за синхронизации).

- Всегда переопределяй equals() и hashCode() **вместе**.
- Используй java.util.Objects — безопасно и читаемо.
- В IntelliJ IDEA: Alt + Insert → генерация методов — экономит время.

- Для денег — только `BigDecimal`.
  - Для конкатенации строк в цикле — только `StringBuilder`.
  - Проверять переполнение в критических местах — используй `Math.*Exact()`.
- 

## 2.24 □ Полезные ссылки

- [Oracle: Primitive Data Types](#)
- [Oracle: BigDecimal](#)
- [Oracle: StringBuilder](#)
- [Хабр: Контракт equals/hashCode](#)
- [Baeldung: Guide to hashCode\(\)](#)
- [Baeldung: BigDecimal](#)

““

## 3 □ Лекция 3 — Структуры данных в Java

---

### 3.1 □ Основные темы

- Полная структура коллекций Java: иерархия интерфейсов, основные реализации, сложность операций, когда что использовать.

### 3.2 □ Иерархия интерфейсов и основные реализации

```
package java.util;
Collection.java
public interface Collection extends Iterable
```

#### 3.2.1 Iterable → Collection<E>

- `List<E>`
  - `ArrayList`
  - `LinkedList`
- `Set<E>`
  - `HashSet`
  - `LinkedHashSet`
  - `SortedSet<E>`
    - \* `TreeSet`
- `Queue<E>`
  - `PriorityQueue`
  - `Deque<E>`
    - \* `ArrayDeque`

□ `Map<K,V>` **не наследуется** от `Collection`, но входит в `Collections Framework`.

### 3.2.2 Map<K,V>

- SortedMap<K,V>
  - TreeMap
- HashMap
- ConcurrentMap<K,V>
  - ConcurrentHashMap

### 3.3 □ ArrayList — массив с динамическим ростом

`class ArrayList { transient Object[] elementData; // массив элементов private int size; // текущее количество элементов }`

При создании `List list = new ArrayList<>();` — создается массив длины 0 (lazy init). При первом добавлении — выделяется массив размером 10. При переполнении — новый размер: `newCapacity = oldCapacity + (oldCapacity » 1) → +50%` (не 2x!) □ Почему 1.5x, а не 2x? — Экономия памяти + баланс между частотой копирования и фрагментацией.

Capacity не уменьшается автоматически → используй `trimToSize()` для экономии.

### 3.4 □ Алгоритмическая сложность ArrayList

`list.get(i);` // O(1) Прямой доступ по индексу `list.add(value);` // O(1)\* - амортизированная, O(n) - худшая при ресайзе `list.add(i, value);` // O(n) Сдвиг всех элементов справа `list.remove(i);` // O(n) Сдвиг всех элементов слева `list.contains(v);` // O(n) Линейный поиск `iterator().next();` // O(1) Быстрый обход

### 3.5 □ LinkedList — двусвязный список

Двусвязный список. Прыгает по памяти, поэтому работает медленно.

Внутреннее устройство: `class Node { E item; Node next; Node prev }`

Каждый элемент — отдельный объект в куче → прыжки по памяти → плохая локальность → медленнее, чем ArrayList в большинстве случаев. Добавление/удаление в начале/конце — O(1) Доступ по индексу — O(n)

□ Цитата от Joshua Bloch (создатель LinkedList): “Does anyone actually use LinkedList? I wrote it, and I never use it.”

Когда использовать? □ Только если ты очень часто вставляешь/удаляешь в начале или середине списка, и не нужен доступ по индексу. □ В 95% случаев — ArrayList будет быстрее и эффективнее.

### 3.6 □ Queue / Deque — интерфейсы для очередей

ArrayDeque — циклический буфер (ring buffer). - Все операции в начале/конце — O(1) - Реализует Deque → можно использовать как стек или очередь. - Внутри — массив, растёт по формуле: `newSize = 2 * oldSize + 2` - Не потокобезопасен

□ Лучший выбор для стека/очереди в однопоточном коде → быстрее LinkedList.

### 3.7 □ HashMap — хеш-таблица

Внутреннее устройство:

```
class HashMap<K,V> { Node<K,V>[] table; // массив “корзин” int size; // количество элементов }
static class Node<K,V> implements Map.Entry<K,V> { final int hash; final K key; V value; Node<K,V> next;
// следующий узел в цепочке }
```



- Используется цепочечная адресация.
- Изначально table = null → создаётся при первом put() → размер 16.
- Load factor = 0.75 → при достижении 12 элементов → ресайз до 32.
- Ресайз → в 2 раза.
- Коллизии → цепочки (linked list) → с Java 8, если >8 элементов → красно-чёрное дерево. `hashCode()` → определяет корзину `equals()` → проверяет, тот ли ключ

### 3.8 `TreeMap` — отсортированная мапа

- Основана на красно-чёрном дереве → сбалансированное BST.
- Все операции —  $O(\log n)$
- Ключи должны реализовывать Comparable или передаваться Comparator
- Итерация — в отсортированном порядке
- `Используй`, если нужна сортировка по ключам или range-запросы (subMap, headMap, tailMap)

### 3.9 `HashSet`, `TreeSet`, `LinkedHashMap` — обёртки

#### 3.9.1 `HashSet`

- Внутри использует `HashMap<E, Object>` → значения — заглушки (PRESENT).
- Сложность операций — как у `HashMap`:  $O(1)$

#### 3.9.2 `TreeSet`

- Внутри использует `TreeMap<E, Object>`
- Сложность —  $O(\log n)$

#### 3.9.3 `LinkedHashMap`

- Наследуется от `HashMap`
- Добавляет двойной связанный список для сохранения порядка вставки (или доступа)
- Итерация — в порядке вставки (или LRU, если `accessOrder=true`)
- Небольшой overhead на поддержание списка при добавлении → остальные операции — как у `HashMap`
- `Отлично` подходит для LRU-кэшей → переопредели `removeEldestEntry()`

### 3.10 Практические советы на собеседовании

Почему `ArrayList` чаще `LinkedList`?

→ Локальность данных, меньше аллокаций, быстрее в реальных сценариях.

Что будет, если не переопределить `hashCode()` и `equals()` для ключа в `HashMap`?

→ Объекты не будут находиться → нарушение контракта.

Как работает `ConcurrentHashMap`?

→ Разделён на сегменты (до Java 8) → с Java 8 — использует CAS + synchronized на уровне корзины → высокая конкурентность.

Fail-Fast vs Fail-Safe?

→ `ArrayList.iterator()` — fail-fast → `ConcurrentModificationException`

→ `ConcurrentHashMap.keySet().iterator()` — fail-safe → работает с копией

Как уменьшить capacity ArrayList?  
→ trimToSize()

### 3.11 □ Рекомендуем к прочтению

- Effective Java — Joshua Bloch (главы 3, 6 — коллекции и equals/hashCode)
- Java Concurrency in Practice — Brian Goetz (глава 5 — коллекции в многопоточке)
- [Oracle Java Collections Tutorial](#)