Contents

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

1.1 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

1.2 Настройка Java-окружения

1.2.1 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.

1.2.2 Переменная окружения `PATH`

→ `PATH` — список директорий, где система ищет исполняемые файлы.
 Добавь путь к `bin` JDK в `PATH`:
Linux/macOS (в `~/.bashrc` или `~/.zshrc`):

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

Windows: - Панель управления → Система → Дополнительные параметры → Переменные среды → `PATH` → Добавить путь, например:
`C:\Program Files\Java\jdk-21\bin`

1.2.3 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.

1.3 Работа с IntelliJ IDEA

1.3.1 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.

1.4 ⌨ 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`

Действие	Windows/Linux	macOS
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`

1.5 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`.

1.6 Рефакторинги в IntelliJ IDEA

1.6.1 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}\Oper
    \BuiltInTok{System}\OperatorTok{.}\FunctionTok{out}\OperatorTok{.}\FunctionTok{println}\OperatorTo
\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}\NormalT
    \DataTypeTok{int}\NormalTok{ sum }\OperatorTok{=}\NormalTok{ a }\OperatorTok{+}\NormalTok{ b}\Oper
    \BuiltInTok{System}\OperatorTok{.}\FunctionTok{out}\OperatorTok{.}\FunctionTok{println}\OperatorTo
\OperatorTok{\}}
```

→ Улучшает читаемость и переиспользование.

1.6.2 🔄 Inline Method (Встроить метод)

Если метод слишком простой — можно “встроить” его обратно:
`Ctrl + Alt + N`
→ Полезно при оптимизации или упрощении.

1.7 🧠 Основы синтаксиса Java

1.7.1 Структура программы

```
    \KeywordTok{public} \KeywordTok{class}\NormalTok{ HelloWorld }\OperatorTok{\{}
        \KeywordTok{public} \DataTypeTok{static} \DataTypeTok{void} \FunctionTok{main}\OperatorTok{(}\Buil
            \BuiltInTok{System}\OperatorTok{.}\FunctionTok{out}\OperatorTok{.}\FunctionTok{println}\Operat
        \OperatorTok{\}}
    \OperatorTok{\}}
```

→ Каждая программа начинается с `main`.

1.7.2 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{;}
```

1.7.3 Управляющие конструкции

```
    \ControlFlowTok{if} \OperatorTok{(}\NormalTok{age }\OperatorTok{\textgreater{}=} \DecValTok{18}\Operat
        \BuiltInTok{System}\OperatorTok{.}\FunctionTok{out}\OperatorTok{.}\FunctionTok{println}\OperatorTo
    \OperatorTok{\}} \ControlFlowTok{else} \OperatorTok{\{}
        \BuiltInTok{System}\OperatorTok{.}\FunctionTok{out}\OperatorTok{.}\FunctionTok{println}\OperatorTo
    \OperatorTok{\}}

    \ControlFlowTok{for} \OperatorTok{(}\DataTypeTok{int}\NormalTok{ i }\OperatorTok{=} \DecValTok{0}\Oper
        \BuiltInTok{System}\OperatorTok{.}\FunctionTok{out}\OperatorTok{.}\FunctionTok{println}\OperatorTo
    \OperatorTok{\}}

    \ControlFlowTok{while} \OperatorTok{(}\NormalTok{condition}\OperatorTok{)} \OperatorTok{\{}
        \CommentTok{// ...}
    \OperatorTok{\}}
```

1.8 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.

1.9 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

- Примитивные 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

---

## Примитивные типы данных

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

| Тип      | Размер (биты) | Размер (байты) | Диапазон значений                | Значение по умо
|----------|---------------|----------------|----------------------------------|---------------
| `byte`   | 8 бит         | 1 байт         | -128 до 127                      | `0`
| `short`  | 16 бит        | 2 байта        | -32768 до 32767                  | `0`
```

| `int` | 32 бита | 4 байта | -2^31 до 2^31-1 | `0`
| `long` | 64 бита | 8 байт | -2^63 до 2^63-1 | `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 храни

□ 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{// ← NullPoin
```

→ Всегда проверяй на `null` перед распаковкой.

1.10 🔁 Передача параметров по значению

🔹 В Java всё передаётся по значению — даже объекты!

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

1.10.1 Пример:

```
    \KeywordTok{public} \DataTypeTok{static} \DataTypeTok{void} \FunctionTok{main}\OperatorTok{(}\BuiltInT
    \NormalTok{    Person p }\OperatorTok{=} \KeywordTok{new} \FunctionTok{Person}\OperatorTok{(}\StringTo
        \FunctionTok{changeName}\OperatorTok{(}\NormalTok{p}\OperatorTok{);}
        \BuiltInTok{System}\OperatorTok{.}\FunctionTok{out}\OperatorTok{.}\FunctionTok{println}\OperatorTo
    \OperatorTok{\}}

    \DataTypeTok{static} \DataTypeTok{void} \FunctionTok{changeName}\OperatorTok{(}\NormalTok{Person perso
    \NormalTok{    person}\OperatorTok{.}\FunctionTok{name} \OperatorTok{=} \StringTok{"Bob"}\OperatorTok{
    \OperatorTok{\}}
```

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

1.10.2 Но:

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

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

1.11 📦 Пакеты Java

Пакеты — это пространства имён для классов.

1.11.1 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+).

1.11.2 Импорт:

```
\KeywordTok{import} \ImportTok{java}\OperatorTok{.}\ImportTok{util}\OperatorTok{.}\ImportTok{List}\Ope
\KeywordTok{import} \ImportTok{java}\OperatorTok{.}\ImportTok{util}\OperatorTok{.}\ImportTok{ArrayList
\KeywordTok{import} \ImportTok{java}\OperatorTok{.}\ImportTok{time}\OperatorTok{.}\ImportTok{LocalDate

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

 * — не нагружает приложение, только упрощает написание кода.

1.12 Класс `Object` — корень иерархии

→ Корневой класс всей иерархии в Java.
→ Любой класс — наследник `Object` (явно или неявно).

1.12.1 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+).

1.13 1.1.1 `equals`

```
\KeywordTok{public} \DataTypeTok{boolean} \FunctionTok{equals}\OperatorTok{(}\AttributeTok{@Nullable}
```

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

1.13.1 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{(}
```

→ Это эквивалентно `a == b`.

1.13.2 🔁 Переопределение

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

```
\AttributeTok{@Override}
\KeywordTok{public} \DataTypeTok{boolean} \FunctionTok{equals}\OperatorTok{(}\BuiltInTok{Object}\Norma
    \ControlFlowTok{if} \OperatorTok{(}\KeywordTok{this} \OperatorTok{==}\NormalTok{ o}\OperatorTok{)}
    \ControlFlowTok{if} \OperatorTok{(}\NormalTok{o }\OperatorTok{==} \KeywordTok{null} \OperatorTok{|
\NormalTok{    Person person }\OperatorTok{=} \OperatorTok{(}\NormalTok{Person}\OperatorTok{)}\NormalT
    \ControlFlowTok{return}\NormalTok{ Objects}\OperatorTok{.}\FunctionTok{equals}\OperatorTok{(}\Norm
\NormalTok{           Objects}\OperatorTok{.}\FunctionTok{equals}\OperatorTok{(}\NormalTok{age}\Operat
\OperatorTok{\}}
```

1.13.3 🔁 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.).

1.14 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.

1.14.1 🔁 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).

1.14.2 🔹 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`.

1.14.3 🔹 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{"A
    \NormalTok{map}\OperatorTok{.}\FunctionTok{put}\OperatorTok{(}\NormalTok{p}\OperatorTok{,} \StringTok{

    \NormalTok{p}\OperatorTok{.}\FunctionTok{setName}\OperatorTok{(}\StringTok{"Bob"}\OperatorTok{);} \Com

    \NormalTok{map}\OperatorTok{.}\FunctionTok{get}\OperatorTok{(}\NormalTok{p}\OperatorTok{);} \CommentTo
```

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

1.15 🔹 Правильное переопределение

1.15.1 Способ 1: Вручную

```
    \AttributeTok{@Override}
    \KeywordTok{public} \DataTypeTok{boolean} \FunctionTok{equals}\OperatorTok{(}\BuiltInTok{Object}\Norma
        \ControlFlowTok{if} \OperatorTok{(}\KeywordTok{this} \OperatorTok{==}\NormalTok{ o}\OperatorTok{)}
        \ControlFlowTok{if} \OperatorTok{(}\NormalTok{o }\OperatorTok{==} \KeywordTok{null} \OperatorTok{|
    \NormalTok{    Person person }\OperatorTok{=} \OperatorTok{(}\NormalTok{Person}\OperatorTok{)}\NormalT
        \ControlFlowTok{return}\NormalTok{ Objects}\OperatorTok{.}\FunctionTok{equals}\OperatorTok{(}\Norm
    \NormalTok{           Objects}\OperatorTok{.}\FunctionTok{equals}\OperatorTok{(}\NormalTok{age}\Operat
    \OperatorTok{\}}

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

1.15.2 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{\}}
```

1.15.3 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}\Norma
    \ControlFlowTok{if} \OperatorTok{(}\KeywordTok{this} \OperatorTok{==}\NormalTok{ o}\OperatorTok{)}
    \ControlFlowTok{if} \OperatorTok{(}\NormalTok{o }\OperatorTok{==} \KeywordTok{null} \OperatorTok{|
\NormalTok{    Person person }\OperatorTok{=} \OperatorTok{(}\NormalTok{Person}\OperatorTok{)}\NormalT
    \ControlFlowTok{return}\NormalTok{ Objects}\OperatorTok{.}\FunctionTok{equals}\OperatorTok{(}\Norm
\NormalTok{           Objects}\OperatorTok{.}\FunctionTok{equals}\OperatorTok{(}\NormalTok{age}\Operat
\OperatorTok{\}}

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

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

1.16 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.

1.17 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`.

1.18 🔁 Контракт `equals()` и `hashCode()`

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

1.18.1 Почему это важно?

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

1.18.2 Антипаттерн:

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

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

→ Объекты могут “потеряться” в `HashMap`.

1.19 ⚠️ 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.

1.20 🔢 Переполнение (Overflow)

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

1.20.1 Пример с `int`:

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

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

1.20.2 Как избежать?

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

```
\ControlFlowTok{try} \OperatorTok{\{}
    \DataTypeTok{int}\NormalTok{ result }\OperatorTok{=} \BuiltInTok{Math}\OperatorTok{.}\FunctionTok{
\OperatorTok{\}} \ControlFlowTok{catch} \OperatorTok{(}\BuiltInTok{ArithmeticException}\NormalTok{ e}\
    \BuiltInTok{System}\OperatorTok{.}\FunctionTok{out}\OperatorTok{.}\FunctionTok{println}\OperatorTo
\OperatorTok{\}}
```

- Для критических вычислений — используй `BigInteger`.

1.21 🔢 `BigDecimal` — для точных вычислений (деньги!)

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

1.21.1 Пример проблемы:

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

1.21.2 Решение — `BigDecimal`:

```
\BuiltInTok{BigDecimal}\NormalTok{ a }\OperatorTok{=} \KeywordTok{new} \BuiltInTok{BigDecimal}\Operato
\BuiltInTok{BigDecimal}\NormalTok{ b }\OperatorTok{=} \KeywordTok{new} \BuiltInTok{BigDecimal}\Operato
\BuiltInTok{BigDecimal}\NormalTok{ sum }\OperatorTok{=}\NormalTok{ a}\OperatorTok{.}\FunctionTok{add}\
\BuiltInTok{System}\OperatorTok{.}\FunctionTok{out}\OperatorTok{.}\FunctionTok{println}\OperatorTok{(}
```

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

```
\KeywordTok{new} \BuiltInTok{BigDecimal}\OperatorTok{(}\FloatTok{0.1}\OperatorTok{)}   \CommentTok{//
\KeywordTok{new} \BuiltInTok{BigDecimal}\OperatorTok{(}\StringTok{"0.1"}\OperatorTok{)} \CommentTok{//
```

1.21.3 Операции:

```
\NormalTok{a}\OperatorTok{.}\FunctionTok{add}\OperatorTok{(}\NormalTok{b}\OperatorTok{)}
\NormalTok{a}\OperatorTok{.}\FunctionTok{subtract}\OperatorTok{(}\NormalTok{b}\OperatorTok{)}
\NormalTok{a}\OperatorTok{.}\FunctionTok{multiply}\OperatorTok{(}\NormalTok{b}\OperatorTok{)}
\NormalTok{a}\OperatorTok{.}\FunctionTok{divide}\OperatorTok{(}\NormalTok{b}\OperatorTok{,} \DecValTok
```

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

1.22 🧵 `StringBuilder` — эффективная работа со строками

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

1.22.1 Проблема:

```
\BuiltInTok{String}\NormalTok{ result }\OperatorTok{=} \StringTok{""}\OperatorTok{;}
\ControlFlowTok{for} \OperatorTok{(}\DataTypeTok{int}\NormalTok{ i }\OperatorTok{=} \DecValTok{0}\Oper
\NormalTok{    result }\OperatorTok{+=} \StringTok{"a"}\OperatorTok{;} \CommentTok{// ← создаёт 1000 п
\OperatorTok{\}}
```

→ Медленно и расходует память.

1.22.2 Решение — `StringBuilder`:

```
\BuiltInTok{StringBuilder}\NormalTok{ sb }\OperatorTok{=} \KeywordTok{new} \BuiltInTok{StringBuilder}\
\ControlFlowTok{for} \OperatorTok{(}\DataTypeTok{int}\NormalTok{ i }\OperatorTok{=} \DecValTok{0}\Oper
\NormalTok{    sb}\OperatorTok{.}\FunctionTok{append}\OperatorTok{(}\StringTok{"a"}\OperatorTok{);} \C
\OperatorTok{\}}
\BuiltInTok{String}\NormalTok{ result }\OperatorTok{=}\NormalTok{ sb}\OperatorTok{.}\FunctionTok{toStr
```

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

1.22.3 Основные методы:

```
\NormalTok{sb}\OperatorTok{.}\FunctionTok{append}\OperatorTok{(}\StringTok{"text"}\OperatorTok{)}
\NormalTok{sb}\OperatorTok{.}\FunctionTok{insert}\OperatorTok{(}\DecValTok{0}\OperatorTok{,} \StringTo
\NormalTok{sb}\OperatorTok{.}\FunctionTok{delete}\OperatorTok{(}\DecValTok{0}\OperatorTok{,} \DecValTo
\NormalTok{sb}\OperatorTok{.}\FunctionTok{reverse}\OperatorTok{()}
\NormalTok{sb}\OperatorTok{.}\FunctionTok{toString}\OperatorTok{()}
```

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

1.23 🧠 Советы

- 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`.
- Проверяй переполнение в критических местах — используй `Math.*Exact()`.

1.24 🔗 Полезные ссылки

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

‘‘‘