

Типы данных в Java: от примитивов до коллекций

Лекция 2: Примитивы, ссылки, строки, StringBuilder, базовые коллекции

Александр Глускер

РУТ МИИТ/ВИШ

21 ноября 2025 г.

Содержание

- ① Примитивные и ссылочные типы:
стек vs куча
- ② Передача аргументов по значению,
семантика ссылок
- ③ Автоупаковка и обёртки: мост к
коллекциям
- ④ String: неизменяемость, пул строк,
сравнение через equals
- ⑤ StringBuilder: эффективные мутации
внутренний буфер
- ⑥ Коллекции: List, Set, Map —
интерфейсы и реализации
- ⑦ Практика: типичные ошибки и
анти-паттерны

Цели

- Различать хранение в стеке
и куче
- Понимать семантику
передачи ссылок
- Использовать автоупаковку
осознанно
- Избегать ошибок сравнения
строк
- Выбирать StringBuilder при
конкатенации в цикле
- Применять коллекции
вместо массивов

Навигация по лекции

- 1 Примитивные и ссылочные типы: стек vs куча
- 2 Автоупаковка и обёртки: мост к коллекциям
- 3 String: неизменяемость, пул строк, equals
- 4 StringBuilder: эффективные мутации, внутренний буфер
- 5 Коллекции: List, Set, Map — интерфейсы и реализации
- 6 Практика: типичные ошибки и анти-паттерны

Примитивные типы: хранение в стеке

Тип	Размер и диапазон
byte	1 байт, -128 до 127
short	2 байта, -32768 до 32767
int	4 байта, -2^{31} до $2^{31}-1$
long	8 байт, -2^{63} до $2^{63}-1$
float	4 байта, IEEE 754
double	8 байт, IEEE 754 (по умолчанию)
char	2 байта, UTF-16
boolean	не определён явно, обычно 1 байт

- Хранятся **в стеке вызовов (call stack)** — быстро, локально, автоматически освобождаются
- Не могут быть null, не являются объектами
- Копируются при присваивании и передаче в метод

Сылочные типы: хранение в куче

- Все объекты (включая массивы, строки, экземпляры классов) размещаются в **куче (heap)**
- Переменная-ссылка хранится в стеке и содержит **адрес** объекта в куче
- Объекты управляются сборщиком мусора (GC) — освобождаются, когда становятся недостижимы
- Ссылка может быть `null` — не указывает ни на какой объект

Пример

- `String s = new String("Hello");` — `s` (в стеке) → объект `String` (в куче)
- `int[] arr = new int[10];` — `arr` (в стеке) → массив (в куче)

Глубже: стек вызовов и кадры

```
1 void example() {  
2     int x = 10;           // x — в стеке текущего кадра  
3     String s = "Java";    // s — ссылка в стеке, объект в куче  
4     modify(x, s);  
5 }  
6  
7 void modify(int a, String t) {  
8     // a — копия x в новом кадре стека  
9     // t — копия ссылки s указывает на тот же объект в куче  
10    t = t + "!"; // создается новый объект в куче, t теперь ссылается на него  
11 }
```

Ключевое

Java всегда передаёт аргументы **по значению**: — для примитивов — копия значения, — для ссылок — копия адреса (но объект в куче — один).

Навигация по лекции

- 1 Примитивные и ссылочные типы: стек vs куча
- 2 Автоупаковка и обёртки: мост к коллекциям
- 3 String: неизменяемость, пул строк, equals
- 4 StringBuilder: эффективные мутации, внутренний буфер
- 5 Коллекции: List, Set, Map — интерфейсы и реализации
- 6 Практика: типичные ошибки и анти-паттерны

Автоупаковка: преобразование примитив ↔ объект

```
1 int x = 42;
2 Integer obj = x;           // автоупаковка: int → Integer (new Integer(x) до
3                           // Java 9, кэшированиепосле )
4
5 // В коллекциях—толькообъекты :
6 List<Integer> list = new ArrayList<>();
7 list.add(100);             // автоупаковка: 100 → Integer.valueOf(100)
8 int first = list.get(0);   // автораспаковка: Integer → int
```

Важные детали

- `ArrayList<int>` — **недопустимо** на уровне компиляции
- `Integer` — класс-обёртка, наследник `Object`
- Кэширование значений от -128 до 127 (спецификация JLS)
- `Integer a = 127; Integer b = 127; a == b → true` (из-за кэша), но `a = 128; b = 128; a == b → false`

Опасности автораспаковки: NullPointerException

```
1 List<Integer> numbers = new ArrayList<>();
2 numbers.add(null);           // допустимо— null ссылка—
3
4 // Попытка распаковать null:
5 int value = numbers.get(0); // ← java.lang.NullPointerException!
6
7 // Безопасная альтернатива:
8 Integer boxed = numbers.get(0);
9 if (boxed != null) {
10     int safe = boxed;
11 }
```

Рекомендация

Избегайте null в коллекциях примитивных обёрток. Используйте `Optional<Integer>` или явную проверку.

Навигация по лекции

- 1 Примитивные и ссылочные типы: стек vs куча
- 2 Автоупаковка и обёртки: мост к коллекциям
- 3 **String: неизменяемость, пул строк, equals**
- 4 StringBuilder: эффективные мутации, внутренний буфер
- 5 Коллекции: List, Set, Map — интерфейсы и реализации
- 6 Практика: типичные ошибки и анти-паттерны

String: immutable объект в куче

```
1 String s1 = "Hello";           // литерал—попадает в пул строк      (String Pool)
2 String s2 = "Hello";           // ссылается на тот же объект в пуле
3 String s3 = new String("Hello"); // НОВЫЙ объект в куче , вне пула
4 String s4 = s1.toUpperCase();   // НОВЫЙ объект — "HELLO"
5
6 System.out.println(s1 == s2);   // true — один объект в пуле
7 System.out.println(s1 == s3);   // false — разные объекты
8 System.out.println(s1.equals(s3)); // true — содержание идентично
```

Архитектурные следствия

- Безопасность в многопоточной среде — не требует синхронизации
- Экономия памяти за счёт пула строк
- Каждая операция возвращает **новый объект** → дорого при частом изменении

Сравнение: == vs .equals()

```
1 String input = new Scanner(System.in).nextLine();
2
3 // Правильная ошибка:
4 if (input == "exit") { ... } // сравнивает адреса, почти всегда false
5
6 // Правильно:
7 if ("exit".equals(input)) { ... } // сравнивает содержимое, безопасно даже при
     null
8
9 // Альтернатива с защитой от null:
10 if (input != null && input.equals("exit")) { ... }
```

Правило

== — сравнение **ссылок** (адресов в памяти), **.equals()** — сравнение **логического содержания** (переопределён в **String**).

Полезные методы String (java.lang.String)

```
1 String text = " Java 17 is Great! ";
2
3 text.length();           // 21 – количество символов (char)
4 text.charAt(2);         // 'v' – символ по индексу
5 text.substring(2, 6);   // "va 1" – [beginIndex, endIndex]
6 text.indexOf("17");    // 6 – позиция подстроки
7 text.replace("Great", "Cool"); // " Java 17 is Cool! "
8 text.trim();            // "Java 17 is Great!" – удаляет пробелы по краям
9 text.split("\\s+");     // ["Java", "17", "is", "Great!"] –
10 регулярное выражение
11 text.toLowerCase();    // " java 17 is great! "
12 text.startsWith(" J"); // true
13 text.endsWith("! ");   // true
```

Навигация по лекции

- 1 Примитивные и ссылочные типы: стек vs куча
- 2 Автоупаковка и обёртки: мост к коллекциям
- 3 String: неизменяемость, пул строк, equals
- 4 **StringBuilder: эффективные мутации, внутренний буфер**
- 5 Коллекции: List, Set, Map — интерфейсы и реализации
- 6 Практика: типичные ошибки и анти-паттерны

StringBuilder: изменяемый аналог String

```
1 StringBuilder sb = new StringBuilder(); // начальная ёмкость— 16 символов
2 sb.append("Hello");
3 sb.append(" ");
4 sb.append("World");
5 String result = sb.toString(); // "Hello World" —неизменяемая строка
6
7 // Цепочка вызовов :
8 sb.append("!").append("\n").append("Java").append(21);
```

Внутреннее устройство

- Инкапсулирует изменяемый char[] буфер в куче
- При переполнении — создаётся новый буфер с увеличенной ёмкостью (x2)
- Все операции — **мутирующие**, не создают промежуточные объекты

Производительность: StringBuilder vs String+

```
1 // ☠Антипаттерн -: O(n2) -создаётся n временных строк
2 String s = "";
3 for (int i = 0; i < 10000; i++) {
4     s = s + i; // каждый + → new String → копирование всего предыдущего
5 }
6
7 // ☠Оптимально : O(n) -модифицирует один буфер
8 StringBuilder sb = new StringBuilder();
9 for (int i = 0; i < 10000; i++) {
10     sb.append(i);
11 }
12 String result = sb.toString();
```

Рекомендация

Используйте `StringBuilder` при конкатенации **в цикле** или при сборке строки из многих частей. Для 2-3 конкатенаций — + допустим (компилятор может сам оптимизировать в `StringBuilder`).

Навигация по лекции

- 1 Примитивные и ссылочные типы: стек vs куча
- 2 Автоупаковка и обёртки: мост к коллекциям
- 3 String: неизменяемость, пул строк, equals
- 4 StringBuilder: эффективные мутации, внутренний буфер
- 5 Коллекции: List, Set, Map — интерфейсы и реализации
- 6 Практика: типичные ошибки и анти-паттерны

Иерархия коллекций: java.util.Collection

- Collection – корневой интерфейс (не включает Map)
- List – упорядоченная коллекция, допускает дубликаты, доступ по индексу
- Set – неупорядоченная, уникальные элементы, нет индексов
- Map – не наследует Collection, хранит пары ключ→значение, ключи уникальны

Стандартные реализации

- List → ArrayList (массив с динамическим ростом)
- Set → HashSet (хеш-таблица, O(1) вставка/поиск)
- Map → HashMap (хеш-таблица, O(1) операции)

List: ArrayList — динамический массив

```
1 List<String> names = new ArrayList<>(); // diamond operator (Java 7+)
2 names.add("Alice");      // [Alice]
3 names.add("Bob");        // [Alice, Bob]
4 names.add(0, "Eve");     // [Eve, Alice, Bob] –вставка по индексу
5 names.set(1, "Carol");   // [Eve, Carol, Bob] –замена
6 names.remove("Bob");    // [Eve, Carol]
7
8 System.out.println(names.get(0));      // "Eve"
9 System.out.println(names.size());      // 2
10 System.out.println(names.contains("Eve")); // true
11
12 // Итерация – for-each рекомендуется():
13 for (String name : names) {
14     System.out.println(name);
15 }
```

Set: HashSet — уникальность через хеш

```
1 Set<String> tags = new HashSet<>();
2 tags.add("java");           // true – добавлено
3 tags.add("spring");        // true – добавлено
4 tags.add("java");          // false – дубликат, игнорируется
5
6 System.out.println(tags.size());      // 2
7 System.out.println(tags.contains("java")); // true
8
9 // Итерация–порядок негарантирован !
10 for (String tag : tags) {
11     System.out.println(tag);
12 }
13
14 // Удаление:
15 tags.remove("spring");
```

Важно

Для корректной работы HashSet объекты должны корректно

Map: HashMap — ассоциативный массив

```
1 Map<String, Integer> scores = new HashMap<>();
2 scores.put("Alice", 95);           // добавление
3 scores.put("Bob", 87);
4 scores.put("Alice", 98);           // обновление значения по ключу
5
6 System.out.println(scores.get("Alice")); // 98
7 System.out.println(scores.containsKey("Bob")); // true
8 System.out.println(scores.size());       // 2
9
10 // Удаление:
11 scores.remove("Bob");
12
13 // Итерация по ключам :
14 for (String key : scores.keySet()) {
15     System.out.println(key + " → " + scores.get(key));
16 }
17
18 // Итерация по записям предпочтительно () :
19 for (Map.Entry<String, Integer> entry : scores.entrySet()) {
```



Фабричные методы: неизменяемые коллекции (Java 9+)

```
1 // Создание маленьких, неизменяемых коллекций – удобно и безопасно
2 List<String> colors = List.of("red", "green", "blue");
3 Set<Integer> primes = Set.of(2, 3, 5, 7);
4 Map<String, String> config = Map.of(
5     "host", "localhost",
6     "port", "8080"
7 );
8
9 // Попытка модификации → UnsupportedOperationException
10 colors.add("yellow"); // ← ошибка в момент выполнения !
11
12 // Для изменяемых коллекций из массива :
13 List<String> fromArray = new ArrayList<>(Arrays.asList("A", "B"));
```

Навигация по лекции

- 1 Примитивные и ссылочные типы: стек vs куча
- 2 Автоупаковка и обёртки: мост к коллекциям
- 3 String: неизменяемость, пул строк, equals
- 4 StringBuilder: эффективные мутации, внутренний буфер
- 5 Коллекции: List, Set, Map — интерфейсы и реализации
- 6 Практика: типичные ошибки и анти-паттерны

Типичные ошибки: диагностика и профилактика

1. NullPointerException при автораспаковке

- `List<Integer> list = Arrays.asList(1, null, 3);`
- `int x = list.get(1);` → `NullPointerException`
- **Решение:** избегайте `null` в коллекциях, проверяйте перед распаковкой

2. Сравнение строк через `==`

- `String cmd = scanner.nextLine();`
- `if (cmd == "quit") {...}` → почти всегда `false`
- **Решение:** всегда `.equals()`, лучше `"quit".equals(cmd)` для защиты от `null`

3. Конкатенация строк в цикле

- `String result = "";`

Что нужно запомнить

- Примитивы — в стеке, быстрые, не null; ссылки — адреса объектов в куче, могут быть null
- Передача аргументов — всегда по значению (копия примитива или копия ссылки)
- Автоупаковка — мост к коллекциям, но опасна при null
- String — immutable, пул строк, сравнение ТОЛЬКО через .equals()
- StringBuilder — mutable, эффективен при множественных изменениях
- Коллекции: ArrayList (индексы, дубли), HashSet (уникальность), HashMap (ключи-значения)
- Используйте фабричные методы (List.of()) для неизменяемых коллекций