Типы данных в Java: от примитивов до коллекций

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

Александр Глускер

РУТ МИИТ/ВИШ

17 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

Ссылочные 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 —встекетекущегокадра
3    String s = "Java";           // s —ссылкавстеке    , объект—вкуче
4    modify(x, s);
5  }
6
7  void modify(int a, String t) {
8    // a —копия  x в( новомкадрестека  )
9    // t —копияссылки   s указывает( натотжеобъектвкуче     )
10   t = t + "!"; // создаётсяНОВЫЙобъектвкуче     , t теперьссылаетсянанего
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
      Java 9, кэшированиепосле )
3 int y = obj;               // автораспаковка: Integer → int
4
5 // Вколлекциях–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— nullссылка—
3
4  // Попыткараспаковать null:
5  int value = numbers.get(0); // ← java.lang.NullPointerException!
6
7  // Безопаснаяальтернатива :
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

```
String s1 = "Hello";            // литерал—попадаетвпулстрок      (String Pool)
String s2 = "Hello";            // ссылаетсянатотжеобъектвпуле
String s3 = new String("Hello"); // НОВЫЙобъектвкуче   , внепула
String s4 = s1.toUpperCase();  // НОВЫЙобъект—   "HELLO"

System.out.println(s1 == s2);  // true —одинобъектвпуле
System.out.println(s1 == s3);  // false —разныеобъекты
System.out.println(s1.equals(s3)); // true —содержаниеидентично
```

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()

```
String input = new Scanner(System.in).nextLine();

// Опаснаяошибка :
if (input == "exit") { ... } // сравниваетадреса , почтивсегда false

// Правильно :
if ("exit".equals(input)) { ... } // сравниваетсодержимое , безопаснодажепри
    null

// Альтернативасзащитойот null:
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)

```
String text = "  Java 17 is Great!  ";

text.length();              // 21 —количествосимволов   (char)
text.charAt(2);             // 'v' —символпоиндексу
text.substring(2, 6);       // "va 1" — [beginIndex, endIndex)
text.indexOf("17");         // 6 —позицияподстроки
text.replace("Great", "Cool"); // "  Java 17 is Cool!  "
text.trim();                // "Java 17 is Great!" —удаляетпробелыпокраям
text.split("\\s+");         // ["Java", "17", "is", "Great!"] —
    регулярноевыражение
text.toLowerCase();         // "  java 17 is great!  "
text.startsWith("  J");     // true
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

```java
StringBuilder sb = new StringBuilder(); // начальнаяёмкость—   16 символов
sb.append("Hello");
sb.append(" ");
sb.append("World");
String result = sb.toString(); // "Hello World" —неизменяемаястрока

// Цепочкавызовов :
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+

```
// Антипаттерн: O(n²) — создаётся n временных строк
String s = "";
for (int i = 0; i < 10000; i++) {
  s = s + i; // каждый + → new String → копирование всего предыдущего
}

// Оптимально: O(n) — мутируется один буфер
StringBuilder sb = new StringBuilder();
for (int i = 0; i < 10000; i++) {
  sb.append(i);
}
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

```java
List<String> names = new ArrayList<>(); // diamond operator (Java 7+)
names.add("Alice");    // [Alice]
names.add("Bob");      // [Alice, Bob]
names.add(0, "Eve");   // [Eve, Alice, Bob] —вставкапоиндексу
names.set(1, "Carol"); // [Eve, Carol, Bob] —замена
names.remove("Bob");   // [Eve, Carol]

System.out.println(names.get(0));     // "Eve"
System.out.println(names.size());     // 2
System.out.println(names.contains("Eve")); // true

// Итерация—  for—each рекомендуется():
for (String name : names) {
  System.out.println(name);
}
```

Set: HashSet — уникальность через хеш

```java
Set<String> tags = new HashSet<>();
tags.add("java");      // true —добавлено
tags.add("spring");    // true —добавлено
tags.add("java");      // false —дубликат , игнорируется

System.out.println(tags.size());       // 2
System.out.println(tags.contains("java")); // true

// Итерация—порядокнегарантирован    !
for (String tag : tags) {
  System.out.println(tag);
}

// Удаление:
tags.remove("spring");
```

Важно

Для корректной работы HashSet объекты должны корректно

Map: HashMap — ассоциативный массив

```
Map<String, Integer> scores = new HashMap<>();
scores.put("Alice", 95);     // добавление
scores.put("Bob", 87);
scores.put("Alice", 98);     // обновлениезначенияпоключу

System.out.println(scores.get("Alice")); // 98
System.out.println(scores.containsKey("Bob")); // true
System.out.println(scores.size());       // 2

// Удаление:
scores.remove("Bob");

// Итерацияпоключам  :
for (String key : scores.keySet()) {
  System.out.println(key + " → " + scores.get(key));
}

// Итерацияпозаписямпредпочтительно   ():
for (Map.Entry<String, Integer> entry : scores.entrySet()) {
```

Фабричные методы: неизменяемые коллекции (Java 9+)

```
// Созданиемаленьких , неизменяемыхколлекций—удобноибезопасно
List<String> colors = List.of("red", "green", "blue");
Set<Integer> primes = Set.of(2, 3, 5, 7);
Map<String, String> config = Map.of(
  "host", "localhost",
  "port", "8080"
);

// Попыткамодификации→   UnsupportedOperationException
colors.add("yellow"); // ←ошибкавременивыполнения   !

// Дляизменяемыхколлекцийизмассива    :
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