

Java Black Belt

Java Cheat Sheet

Самые важные конструкции, методы и паттерны

1. Примитивные типы данных

Тип	Размер	Диапазон / Пример	Wrapper
<code>byte</code>	1 байт	-128..127	<code>Byte</code>
<code>short</code>	2 байт	-32768..32767	<code>Short</code>
<code>int</code>	4 байт	-2B..2B	<code>Integer</code>
<code>long</code>	8 байт	...L суффикс	<code>Long</code>
<code>float</code>	4 байт	3.14f	<code>Float</code>
<code>double</code>	8 байт	3.14	<code>Double</code>
<code>char</code>	2 байт	'A', '\n'	<code>Character</code>
<code>boolean</code>	1 бит	true / false	<code>Boolean</code>

2. String — ключевые методы

Основные методы

```
length() s.length()
charAt(i) s.charAt(0) // первый символ
substring() s.substring(2, 5)
contains() s.contains("Java")
indexOf() s.indexOf('a') // -1 если нет
replace() s.replace("old", "new")
toUpperCase() s.toUpperCase()
trim() s.trim() // убрать пробелы
split() s.split(",") // String[]
```

String vs StringBuilder

- String — immutable, новый объект при каждом изменении
- StringBuilder — mutable, эффективен в цикле

StringBuilder

```
append() sb.append("text")
insert() sb.insert(0, "!")
delete() sb.delete(1, 3)
reverse() sb.reverse()
toString() sb.toString()
```

3. Коллекции — шпаргалка

List

```
ArrayList // быстрый доступ O(1)
add(e) list.add(e)
remove(i) list.remove(i)
get(i) list.get(i)
size() list.size()
contains(e) list.contains(e)
sort(Comparator) list.sort(Comparator)
```

Map

```
HashMap // ключ-значение
put(k,v) map.put(k,v)
get(k) map.get(k)
remove(k) map.remove(k)
containsKey(k) map.containsKey(k)
keySet() map.keySet()
values() map.values()
```

Set

```
HashSet // уникальные элементы
add(e) set.add(e)
remove(e) set.remove(e)
contains(e) set.contains(e)
size() set.size()
```

Queue / Deque

```
LinkedList // FIFO
offer(e) queue.offer(e)
poll() queue.poll()
peek() queue.peek()
push(e) queue.push(e)
pop() queue.pop()
```

Итерация

```
// for-each
```

<pre>entrySet() map.entrySet()</pre>	<pre>for (String s : list) { } // Iterator Iterator<String> it = list.iterator(); // stream list.stream().forEach(System.out::println);</pre>
--------------------------------------	---

4. Stream API (Java 8+)

Основные операции

```
List<String> names = List.of("Alice", "Bob", "Charlie", "Anna");

// filter + map + collect
List<String> result = names.stream()
    .filter(n -> n.startsWith("A")) // Alice, Anna
    .map(String::toUpperCase)       // ALICE, ANNA
    .sorted()                       // ALICE, ANNA (по алфавиту)
    .collect(Collectors.toList());

// reduce
int sum = List.of(1,2,3,4,5).stream()
    .reduce(0, Integer::sum); // 15

// findFirst, anyMatch, count
Optional<String> first = names.stream().filter(n -> n.length() > 3).findFirst();
boolean hasAnna = names.stream().anyMatch(n -> n.equals("Anna"));
long count = names.stream().filter(n -> n.startsWith("A")).count();

// groupingBy
Map<Integer, List<String>> byLength = names.stream()
    .collect(Collectors.groupingBy(String::length));
```

5. Исключения

Иерархия

- Throwable
 - Error (не перехватывать!)
 - OutOfMemoryError, StackOverflowError
 - Exception
 - Checked: IOException, SQLException
 - Unchecked (RuntimeException):
 - NullPointerException,
 - IllegalArgumentException

try-catch-finally

```
try {
    int x = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println(e.getMessage());
} finally {
    // выполнится всегда
}
```

try-with-resources (Java 7+)

```
try (BufferedReader br =
    new BufferedReader(new FileReader("f.txt"))
    String line = br.readLine();
) catch (IOException e) {
    e.printStackTrace();
}
// br закроется автоматически
```

Свое исключение

```
public class AppException
    extends RuntimeException {
    public AppException(String msg) {
        super(msg);
    }
}
// использование:
throw new AppException("Ошибка!");
```

6. ООП — ключевые конструкции

Класс / Наследование

```
public class Animal {
    private String name;
    public Animal(String name) {
        this.name = name;
    }
    public void speak() {
        System.out.println("...");
    }
}

public class Dog extends Animal {
    public Dog(String name) {
        super(name); // вызов родителя
    }
    @Override
    public void speak() {
        System.out.println("Woof!");
    }
}
```

Интерфейс

```
public interface Runnable {
    void run(); // абстрактный
    default void stop() { // Java8
        System.out.println("stop");
    }
}

public class Task implements Runnable {
    @Override
    public void run() {
        System.out.println("running");
    }
}
```

Абстрактный класс

```
public abstract class Shape {
    abstract double area(); // без тела
    public void print() { // с телом
        System.out.println(area());
    }
}
```

7. Паттерны проектирования

Singleton

```
public class Singleton {
    private static Singleton instance;
    private Singleton() {}
    public static synchronized
    Singleton getInstance() {
        if (instance == null)
            instance = new Singleton();
        return instance;
    }
}
```

Builder

```
User user = new User.Builder()
    .name("Alice")
    .email("a@b.com")
    .age(25)
    .build();
```

Factory

```
public class ShapeFactory {
    public static Shape create(String t) {
        return switch (t) {
            case "circle" -> new Circle();
            case "square" -> new Square();
            default -> throw new
                IllegalArgumentException(t);
        };
    }
}
```

Strategy

```
interface SortStrategy {
    void sort(int[] arr);
}

class BubbleSort implements SortStrategy {
    public void sort(int[] arr) { /*...*/ }
}

class Sorter {
    private SortStrategy strategy;
    public void setStrategy(SortStrategy s)
        { this.strategy = s; }
    public void sort(int[] arr)
        { strategy.sort(arr); }
}
```

8. Многопоточность

Thread / Runnable

```
// Вариант 1: Runnable + Thread
Thread t = new Thread(() -> {
    System.out.println("Hello!");
});
t.start();

// Вариант 2: ExecutorService
ExecutorService pool =
    Executors.newFixedThreadPool(4);
pool.submit(() -> doWork());
pool.shutdown();
```

synchronized

```
public class Counter {
    private int count = 0;
    public synchronized void increment() {
        count++;
    }
    public synchronized int get() {
        return count;
    }
}
```

CompletableFuture

```
CompletableFuture<String>
    future = CompletableFuture
        .supplyAsync(() -> "result")
        .thenApply(s -> s.toUpperCase());

String result = future.get();
```

volatile vs synchronized

- **volatile** — гарантирует видимость изменений между потоками
- **synchronized** — гарантирует атомарность + видимость
- **AtomicInteger** — атомарные операции без блокировок

```
AtomicInteger counter = new AtomicInteger(0);
counter.incrementAndGet(); // thread-safe
```

9. Spring Boot — шпаргалка аннотаций

Аннотация	Назначение
@SpringBootApplication	Точка входа приложения: @Configuration + @EnableAutoConfiguration + @ComponentScan
@RestController	Контроллер REST API (объединяет @Controller + @ResponseBody)
@GetMapping / @PostMapping	Маппинг HTTP методов GET / POST на метод контроллера
@RequestBody	Десериализовать тело запроса в объект Java
@PathVariable	Извлечь параметр из URL: /users/{id}
@RequestParam	Параметр из query string: /users?page=1
@Service	Бизнес-логика — бин сервисного слоя
@Repository	Слой работы с БД — бин репозитория
@Autowired	Внедрение зависимости (лучше через конструктор)
@Entity / @Table	JPA: класс — таблица в БД
@Id / @GeneratedValue	JPA: первичный ключ и стратегия генерации
@Transactional	Транзакция: откат при исключении
@Value	Внедрить значение из application.properties
@Configuration	Класс конфигурации (источник бинов @Bean)

Хочешь стать Java-разработчиком с нуля?

Записывайся на бесплатную консультацию: t.me/JavaBlackBelt_Bot