Complete Java Chetasheet

~Developer Shaurya

Important Links

Java Cheatsheet:
https://developershaurya.com/rustcheatsheet/

Website: https://developershaurya.com/

All CheatSheets:
https://developershaurya.com/cheat-sheets/

Youtube:
https://www.youtube.com/@DeveloperShaurya

Developer Shaurya Community:
https://t.me/developer_shaurya (Telegram)

CheatSheet Java

Java Complete Cheatsheet – 2025 Ultimate Edition

by Developer Shaurya • October 22, 2025 • 0

1. Basic Structure & Syntax

```
class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Comments

```
// Single line
/* Multi-line */
```

2. Data Types & Variables

Type	Example	Bytes
byte	127	1
short	32000	2
int	100000	4
long	100000L	8
float	3.14f	4
double	3.14159	8
char	'A'	2
boolean	true/false	1
String	"Hello"	varies

Type Casting

```
int x = 10;
double y = x;       // implicit
int z = (int) y;    // explicit
```

3. Operators

`+ - * / %, ++ --, == != > < >= <=, && || !, = += -= *= /=, & | ^ ~ << >> >>>`

4. Control Flow

```
if (x > 10) { }
else if (x == 10) { }
else { }
```

Switch (Modern)

```
int day = 2;
String result = switch (day) {
    case 1 -> "Mon";
    case 2 -> "Tue";
    default -> "Weekend";
};
```

Loops

```
for (int i = 0; i < 5; i++) { }
while (condition) { }
do { } while (condition);
for (int i : arr) { }
```

5. Methods

```
static int add(int a, int b) { return a + b; }
```

Overloading

```
void print(int a) {}
void print(String s) {}
```

6. OOP: Classes & Objects

```
class Car {
    String color;
    Car(String color) { this.color = color; }
    void drive() { System.out.println(color + " car drives"); }
}
Car c = new Car("Red");
c.drive();
```

7. Access Modifiers

Modifier	Scope
public	Everywhere
protected	Same package + subclass
default	Same package
private	Same class

8. Inheritance

```
class Vehicle { void move(){} }
class Car extends Vehicle { void move(){ System.out.println("Car moves"); }
}
```

`super()` → calls parent constructor or method.

9. Polymorphism

```
Vehicle v = new Car();
v.move(); // Car's version
```

10. Abstraction

Abstract Class

```
abstract class Animal { abstract void sound(); }
class Dog extends Animal { void sound(){ System.out.println("Bark"); } }
```

Interface

```
interface Animal {
    void eat();
    default void sleep() { System.out.println("Sleeping"); }
}
```

11. Encapsulation

```
class Person {
    private String name;
    public String getName(){ return name; }
    public void setName(String n){ this.name = n; }
}
```

12. Packages

```
package com.example;
import java.util.*;
```

13. Exception Handling

```
try {
    int x = 10/0;
} catch (ArithmeticException e) {
    System.out.println(e);
} finally {
    System.out.println("Always runs");
}
```

Custom Exception

```
class MyException extends Exception {
    MyException(String msg){ super(msg); }
}
```

Try-With-Resources

```
try (Scanner sc = new Scanner(System.in)) {
    sc.nextLine();
}
```

14. Arrays

```
int[] nums = {1,2,3};
int[][] matrix = {{1,2}, {3,4}};
```

15. Strings

```
String s = "Hello";
s.length(); s.charAt(0); s.equals("Hi");
s.toUpperCase(); s.substring(1);
```

16. Wrapper Classes

Primitive	Wrapper
int	Integer
char	Character
boolean	Boolean
double	Double

17. Collections Framework

List

```
List<String> list = new ArrayList<>();
list.add("A");
```

Set

```
Set<Integer> set = new HashSet<>();
```

Map

```
Map<Integer, String> map = new HashMap<>();
map.put(1, "A");
```

Queue

```
Queue<String> q = new LinkedList<>();
q.add("Task");
```

18. Generics

```
class Box<T> { T val; Box(T val){this.val=val;} }
Box<Integer> b = new Box<>(5);
```

19. Lambda & Functional Programming

```
List<Integer> nums = List.of(1,2,3);
nums.forEach(n -> System.out.println(n));
```

Functional Interface

```
@FunctionalInterface
interface Greet { void say(String msg); }
```

20. Stream API

```
List<Integer> even = nums.stream()
                         .filter(n -> n%2==0)
                         .map(n -> n*2)
                         .toList();
```

Common methods: `filter()`, `map()`, `reduce()`, `collect()`, `forEach()`

21. Optional

```
Optional<String> name = Optional.ofNullable("Ravi");
name.ifPresent(System.out::println);
```

22. Multithreading

Extending Thread

```
class MyThread extends Thread {
    public void run(){ System.out.println("Running"); }
}
new MyThread().start();
```

Runnable Interface

```
Runnable r = () -> System.out.println("Thread running");
new Thread(r).start();
```

Executors (Modern)

```
ExecutorService pool = Executors.newFixedThreadPool(2);
pool.submit(() -> System.out.println("Task"));
pool.shutdown();
```

23. Synchronization

```
synchronized void printNumbers(){ ... }
```

24. Concurrent Utilities

```
Lock lock = new ReentrantLock();
CountDownLatch latch = new CountDownLatch(3);
AtomicInteger count = new AtomicInteger(0);
```

25. File I/O

```
try (FileWriter fw = new FileWriter("test.txt")) {
    fw.write("Hello");
}

BufferedReader br = new BufferedReader(new FileReader("test.txt"));
System.out.println(br.readLine());
```

26. Networking

Client-Server Example

```
// Server
ServerSocket ss = new ServerSocket(8080);
Socket s = ss.accept();
DataInputStream dis = new DataInputStream(s.getInputStream());
System.out.println(dis.readUTF());

// Client
Socket s = new Socket("localhost", 8080);
DataOutputStream dos = new DataOutputStream(s.getOutputStream());
dos.writeUTF("Hello Server");
```

27. JDBC (Database Connectivity)

```
import java.sql.*;

Connection con = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/test", "root", "password");
Statement st = con.createStatement();
ResultSet rs = st.executeQuery("SELECT * FROM users");
while(rs.next()) System.out.println(rs.getString("name"));
con.close();
```

28. Reflection API

```
Class<?> c = Class.forName("java.lang.String");
Method[] methods = c.getDeclaredMethods();
for (Method m : methods) System.out.println(m.getName());
```

29. Annotations

```
@interface MyAnno { String value(); }
@MyAnno("Hello")
public class Test {}
```

Built-in: `@Override`, `@Deprecated`, `@SuppressWarnings`, `@FunctionalInterface`

30. Serialization

```
ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("obj.ser"));
oos.writeObject(obj);
oos.close();

ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("obj.ser"));
MyClass obj = (MyClass) ois.readObject();
```

31. Java Modules (Java 9+)

module-info.java

```
module com.app {
    requires java.sql;
    exports com.app.service;
}
```

32. Design Patterns (Essentials)

Pattern	Description
Singleton	One instance globally
Factory	Creates objects without exposing logic
Builder	Step-by-step object creation
Observer	Notify all dependents on change
Strategy	Choose algorithm dynamically
Decorator	Add features dynamically
Adapter	Convert one interface to another

33. JVM & Memory

Area	Description
Heap	Objects
Stack	Method calls
Method Area	Class info
PC Register	Thread execution point
GC	Automatic cleanup

34. Performance & GC

Use flags:

```
java -Xms512m -Xmx1024m MyApp
```

GC Types: Serial, Parallel, G1, ZGC (Java 15+)

35. Security

```
MessageDigest md = MessageDigest.getInstance("SHA-256");
byte[] hash = md.digest("password".getBytes());
```

36. Modern Java Features (14–21)

Version	Feature	Example
14	Switch Expressions	`case 1 -> "One";`
15	Text Blocks	`"""Multi-line"""`
16	Records	`record Point(int x, int y) {}`
17	Sealed Classes	`sealed class Shape permits Circle {}`
19	Virtual Threads	`Thread.startVirtualThread(() -> {});`
21	Pattern Matching for Switch	`case String s -> ...;`

37. Testing (JUnit 5)

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class CalcTest {
    @Test
    void addTest() {
        assertEquals(4, 2 + 2);
    }
}
```

38. Build Tools

Maven

`pom.xml` – dependency management

Gradle

`build.gradle` – modern alternative