# Programming in Java

Short notes for B.Tech CSE / IT · BCA · MCA

Uncodemy Handwritten Notes · free for students

## I. Introduction

- **Java** = a high-level, object-oriented, platform-independent language, first released by Sun Microsystems in 1995 (now owned by Oracle).
- Motto: **"Write Once, Run Anywhere"**. Java code compiles to **bytecode**, which runs on any machine with a JVM.

### ➤ How a Java program runs

```
Hello.java --javac--> Hello.class (bytecode) --JVM--> machine code
```

| Term | Meaning |
|---|---|
| JVM | Runs bytecode; has class loader, bytecode verifier, interpreter + JIT compiler |
| JRE | JVM + libraries needed to run Java programs |
| JDK | JRE + development tools (javac, java, javadoc, jar) |

- **JIT (Just-In-Time) compiler** converts frequently used bytecode into native code for speed.

### ➤ Features

- Simple, object-oriented, platform independent, secure (no pointers, bytecode verifier), robust (strong typing, exception handling, garbage collection), multithreaded, distributed.

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

- `public` (JVM can call it), `static` (no object needed), `void` (returns nothing), `String[] args` (command-line arguments).

# 2. Data Types, Operators and Control

| Type | Size | Default |
|---|---|---|
| byte | 1 byte | 0 |
| short | 2 bytes | 0 |
| int | 4 bytes | 0 |
| long | 8 bytes | 0L |
| float | 4 bytes | 0.0f |
| double | 8 bytes | 0.0 |
| char | 2 bytes (Unicode) | '\u0000' |
| boolean | JVM dependent | false |

- Sizes are fixed on every platform (unlike C).
- **Non-primitive types**: classes, arrays, interfaces, String.
- **Type casting**: widening (int → long) is automatic; narrowing (double → int) needs a cast: `int x = (int) 3.7;` gives 3.
- Operators are like C, plus `>>>` (unsigned right shift) and `instanceof`.
- Control statements: if, switch, for, while, do-while, enhanced for (`for (int x : arr)`), break, continue.

> **TIP** Default values apply to instance variables and array elements only. Local variables must be initialised before use, or the code will not compile.

## ➤ Arrays and Strings

- `int[] a = new int[5];` `a.length` gives 5. Arrays are objects.
- **String** is immutable. Each change creates a new object.
- **StringBuilder** (fast, not thread-safe) and **StringBuffer** (thread-safe, slower) are mutable.
- `==` compares references; `equals()` compares content.

> **EXAMPLE** `String a = "hi"; String b = "hi"; String c = new String("hi");` a b is true (string pool), a c is false, a.equals(c) is true.

# 3. Classes, Objects and Constructors

```
class Student {
    int roll; String name;
    Student(int r, String n) { roll = r; name = n; } // constructor
    void show() { System.out.println(roll + " " + name); }
}
Student s = new Student(1, "Asha");
```

- Objects are created on the `heap` with `new`. Unused objects are removed by the **garbage collector**.
- **Constructor**: same name as the class, no return type. Types: default, parameterized. Java has no destructor.
- `this` refers to the current object. `this(...)` calls another constructor of the same class (must be the first line).
- **static** members belong to the class, not to objects. A static method cannot use `this` or non-static members directly.
- **final**: final variable = constant, final method = cannot be overridden, final class = cannot be inherited.

### ➤ Access modifiers

| Modifier | Class | Package | Subclass (other package) | Everywhere |
|---|---|---|---|---|
| private | Yes | No | No | No |
| default (none) | Yes | Yes | No | No |
| protected | Yes | Yes | Yes | No |
| public | Yes | Yes | Yes | Yes |

# 4. Inheritance and Polymorphism

- `class Dog extends Animal { }`. Java supports single, multilevel and hierarchical inheritance with classes.
- **Multiple inheritance of classes is not allowed** (to avoid ambiguity); it is done with interfaces.
- `super` calls the parent's constructor (`super(...)`, first line) or parent's members (`super.show()`).
- Every class inherits from `java.lang.Object` by default.

| Overloading | Overriding |
|---|---|
| Same name, different parameters | Same name and same parameters |
| In the same class | In parent and child classes |
| Compile-time polymorphism | Run-time polymorphism |
| Return type can differ | Return type same (or covariant) |
| static methods can be overloaded | static methods are hidden, not overridden |

```
Animal a = new Dog();   // upcasting
a.sound();              // calls Dog's sound() at run time
```

- **Dynamic method dispatch**: the JVM picks the overridden method by the actual object type, not the reference type.

# 5. Abstract Classes, Interfaces and Packages

| Abstract class | Interface |
|---|---|
| `abstract class A { }` | `interface I { }` |
| Can have abstract and normal methods | Abstract methods; default and static methods allowed since Java 8 |
| Can have instance variables and constructors | Only `public static final` constants, no constructor |
| `extends` (only one) | `implements` (many) |
| Use for "is-a" with shared code | Use for a capability / contract |

```
interface Shape { double area(); }
class Circle implements Shape {
    double r; Circle(double r) { this.r = r; }
    public double area() { return Math.PI * r * r; }
}
```

- An abstract class cannot be instantiated, but can have a constructor.
- **Package** = group of related classes. `package mypack;` and `import java.util.*;`
- Built-in packages: `java.lang` (auto-imported), `java.util`, `java.io`, `java.net`, `java.sql`.

**REMEMBER** A class can extend one class and implement many interfaces: `class C extends A implements I1, I2`.

# 6. Exception Handling

- **Exception** = an abnormal event that breaks the normal flow at run time.
- Hierarchy: `Throwable` → `Error` (serious, e.g. OutOfMemoryError) and `Exception`.

| Checked exception | Unchecked exception |
|---|---|
| Checked at compile time | Found at run time |
| Must be caught or declared with `throws` | Not forced |
| IOException, SQLException, FileNotFoundException | ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException |
| Subclass of Exception (not RuntimeException) | Subclass of RuntimeException |

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

- `throw` raises one exception object. `throws` in a method signature declares possible exceptions.
- `finally` runs whether or not an exception happens (it is skipped only if the JVM exits).
- **Custom exception**: `class MyException extends Exception { }`.
- **try-with-resources** closes files automatically: `try (FileReader f = new FileReader("a.txt")) { ... }`.

# 7. Multithreading

- **Thread** = a light-weight unit of execution inside a program. Java has built-in thread support.
- Two ways to create a thread:
  - Extend `Thread` and override `run()`.
  - Implement `Runnable` and pass it to a `Thread` (better, because the class can still extend another class).

```
class Task implements Runnable {
    public void run() { System.out.println("Running"); }
}
Thread t = new Thread(new Task());
t.start();      // start() creates a new thread and calls run()
```

- Calling `run()` directly does not create a new thread; it runs like a normal method.
- **Thread states**: New → Runnable → Running → Blocked / Waiting → Terminated.
- Methods: `start()`, `sleep()`, `join()`, `yield()`, `setPriority()` (1 to 10, default 5).
- **Synchronization**: `synchronized` method or block lets only one thread use an object's lock at a time. Prevents race conditions.
- Inter-thread communication: `wait()`, `notify()`, `notifyAll()` (called inside synchronized code).

# 8. Collections, I/O and JDBC

## ➤ Collections framework

| Interface | Class | Note |
|---|---|---|
| List | ArrayList, LinkedList | Ordered, duplicates allowed |
| Set | HashSet, LinkedHashSet, TreeSet | No duplicates; TreeSet is sorted |
| Queue | PriorityQueue, ArrayDeque | FIFO / priority order |
| Map | HashMap, LinkedHashMap, TreeMap | Key-value pairs, unique keys |

| ArrayList | LinkedList |
|---|---|
| Dynamic array | Doubly linked list |
| Fast random access O(1) | Access O(n) |
| Slow insert in middle | Fast insert / delete at known position |

- `HashMap` allows one null key; `Hashtable` is synchronized and allows no null key or value.
- **Generics**: `ArrayList<String> list = new ArrayList<>();` gives type safety.

## ➤ I/O streams

- **Byte streams**: InputStream / OutputStream (FileInputStream). **Character streams**: Reader / Writer (FileReader, BufferedReader).
- **Serialization**: convert an object to bytes using `Serializable` and `ObjectOutputStream`. `transient` fields are not saved.

### ➤ JDBC steps

1. Load / register the driver. 2. Create a `Connection` with `DriverManager.getConnection(url, user, pass)`.
2. Create a `Statement` or `PreparedStatement`. 4. Execute the query and read the `ResultSet`. 5. Close the connection.

> **TIP** Prefer PreparedStatement: it is precompiled and protects against SQL injection.

# 9. Most Asked Questions (practice)

1. Explain JVM, JRE and JDK. Why is Java platform independent?
2. List the features of Java. Why does Java not support pointers?
3. Explain constructors, `this` and `super` with examples.
4. Method overloading vs method overriding.
5. Abstract class vs interface. Why does Java not support multiple inheritance of classes?
6. Explain access modifiers and packages.
7. Checked vs unchecked exceptions. Explain try, catch, finally, throw and throws.
8. Two ways to create a thread. What is synchronization?
9. String vs StringBuilder vs StringBuffer.
10. ArrayList vs LinkedList. HashMap vs Hashtable.

> **TIP** University exams ask theory plus short programs on inheritance, interfaces, exceptions and threads. Write complete small programs and also show the expected output.