



Gurukul Technicals Institute

Computer education · Robotics · Science projects · Delhi

FREE EBOOK

PROGRAMMING AND WEB

Java Study Book

The complete study notes, practice questions and glossary for Java

INSIDE THIS BOOK

- The Java Platform and Your First Program
- Data Types, Variables and Operators
- Control Flow: Decisions and Loops
- Arrays and Strings
- Classes, Objects and Constructors
- ...and 2 more chapters, practice questions and a glossary

Beginner

Students learning Java for school, college or a first job

engineergurukul.com

+91 97116 99511

Java Study Book

Learn Java 17 from the ground up: how the JDK, JRE and JVM fit together, data types, control flow, arrays, strings, classes, inheritance, interfaces, packages, exceptions and the ArrayList and HashMap collections.

WHO IT IS FOR	Students learning Java for school, college or a first job
LEVEL	Beginner
SUBJECT AREA	Programming and web
WHAT IS INSIDE	7 chapters, practice questions with answers, and a glossary

CONTENTS

1	The Java Platform and Your First Program	3
2	Data Types, Variables and Operators	5
3	Control Flow: Decisions and Loops	7
4	Arrays and Strings	9
5	Classes, Objects and Constructors	12
6	Inheritance, Interfaces and Packages	14
7	Exceptions and Collections	17
	Practice questions	19
	Answers	23
	Glossary	25

Free ebook: you may read, print and share it for learning; it may not be sold or changed. © 2026 Gurukul Technicals Institute. First edition, October 2026. Contact details are on the back cover.

The Java Platform and Your First Program

How Java runs

A Java program runs in two stages. First, the compiler `javac` turns your source file (`.java`) into bytecode (`.class`). Bytecode is not machine code for any particular processor; it is run by the Java Virtual Machine, which translates it for the computer it is on. Because there is a JVM for each operating system, the same `.class` file runs on Windows, Linux and macOS. This is what "write once, run anywhere" means.

JDK, JRE and JVM

Part	What it is	Who needs it
JVM	Java Virtual Machine: runs bytecode and manages memory	Every computer that runs Java
JRE	Java Runtime Environment: the JVM plus the standard class libraries	Anyone who only runs programs
JDK	Java Development Kit: a complete runtime plus tools such as <code>javac</code> , <code>jar</code> and <code>javadoc</code>	Programmers

So the JDK contains everything in the JRE, and the JRE contains the JVM. With modern versions such as Java 17, a long-term support release, you normally install just the JDK, because it already includes everything needed to run programs.

The JVM also frees memory automatically. When no variable refers to an object any more, the garbage collector reclaims its space, so Java has no `free` or `delete`.

Your first program

JAVA

```
public class Hello {
    // Program execution starts in main
    public static void main(String[] args) {
        System.out.println("Namaste, Rahul!");
        System.out.print("Welcome to Java ");
        System.out.println(17);
    }
}
```

Save it as `Hello.java`, then compile and run it:

```
javac Hello.java
java Hello
```

`javac` creates `Hello.class`, and `java Hello` runs it; note that you give the class name without `.class`. Since Java 11 you can also run a single source file directly with `java Hello.java`.

Reading the program

- `public class Hello` declares a class. A file can contain only one public top-level class, and the file must be named after it, so this one must be saved as `Hello.java`.
- In Java 17, `public static void main(String[] args)` is the entry point. It is `public` so the JVM can call it, `static` so it runs without creating an object, and `void` because it returns nothing. `args` holds any command-line arguments.
- `System.out.println` prints a value and then moves to a new line, while `System.out.print` stays on the same line. So the program's second line of output is `Welcome to Java 17`.
- Statements end with a semicolon, blocks are wrapped in braces, and Java is case-sensitive, so `System` cannot be written as `system`.

Comments work as in C: `//` starts a comment that runs to the end of the line, and a block comment can span several lines. A block comment that begins with a slash and two stars is a Javadoc comment, which the `javadoc` tool turns into documentation pages.

KEY POINTS

- `javac` compiles `java` source into `.class` bytecode, and the JVM runs the bytecode.
- The JDK contains the JRE, and the JRE contains the JVM.
- The same bytecode runs on any operating system that has a JVM.
- A public class must be saved in a file with exactly the same name.
- In Java 17, execution starts at `public static void main(String[] args)`.

TRY THESE EXERCISES

1. Type the Hello program from this lesson with your own name, compile it with `javac Hello.java` and run it with `java Hello`, then list the files in the folder to find the `Hello.class` file that `javac` created.
2. Rename the class in `Hello.java` to `Greeting` without renaming the file, compile it, and copy the error message into your notebook; then fix it so that the file name and the public class name match exactly.
3. Open a command prompt and run `java -version` and `javac -version` to note the Java version installed, then write a program that prints your name, your course and today's date (typed in as text) on three lines with `println`, followed by one line built from two `print` calls.

Data Types, Variables and Operators

Primitive types

Java is statically typed: every variable has a type, fixed when it is declared. There are eight primitive types, and unlike C, the size of each numeric type is fixed by the language, so it is the same on every platform:

Type	Size	Example
<code>byte</code>	8 bits	<code>byte level = 3;</code>
<code>short</code>	16 bits	<code>short year = 2024;</code>
<code>int</code>	32 bits	<code>int qty = 250;</code>
<code>long</code>	64 bits	<code>long mobile = 9876543210L;</code>
<code>float</code>	32 bits	<code>float rate = 18.5f;</code>
<code>double</code>	64 bits	<code>double price = 499.99;</code>
<code>char</code>	16 bits (one UTF-16 unit)	<code>char section = 'B';</code>
<code>boolean</code>	holds <code>true</code> or <code>false</code>	<code>boolean paid = true;</code>

Some literal rules:

- A whole number like `250` is an `int` by default. Add `L` for a `long` value too big for an `int`.
- A decimal like `18.5` is a `double` by default, so a `float` needs the `f` suffix; `float rate = 18.5;` does not compile.
- Underscores can group digits, so `int amount = 1_00_000;` is one lakh.

Everything else, including `String`, arrays and your own classes, is a reference type. A reference variable holds a link to an object, or `null` when it refers to nothing.

Variables, constants and var

JAVA FRAGMENT

```
int quantity = 4;
double rate = 125.5;
final double DELIVERY = 40;           // final: cannot be changed
var total = quantity * rate + DELIVERY; // var infers double
System.out.println("Total: Rs " + total); // Total: Rs 542.0
```

`var`, available since Java 10, lets the compiler infer the type of a local variable from its initial value. The variable is still statically typed: `total` above is a `double` for good.

Type conversion

Java converts automatically from a smaller type to a larger one, such as `int` to `long` or `double`. Going the other way needs an explicit cast, and any fractional part is cut off: `(int) 9.99` is `9`.

Operators

- Arithmetic: `+`, `-`, `*`, `/` and `%`. With two integers, `/` drops the fraction, so `10 / 4` is `2`, while `10 / 4.0` is `2.5`.
- Increment and decrement: `++` and `--`, in prefix or postfix form, as in C.
- Relational: `==`, `!=`, `<`, `>`, `<=` and `>=`. They produce a `boolean`.
- Logical: `&&`, `||` and `!`. The first two short-circuit: the right side is skipped when the left side already decides the result.
- Conditional: `condition ? valueIfTrue : valueIfFalse`.

Strings and the + operator

`+` also joins strings, and Java evaluates it from left to right:

JAVA FRAGMENT

```
System.out.println("Sum: " + 2 + 3);    // Sum: 23
System.out.println(2 + 3 + " items");   // 5 items
System.out.println("Sum: " + (2 + 3));  // Sum: 5
```

Once one side of `+` is a `String`, the other side is converted to text and joined, so add brackets when you want the arithmetic done first.

KEY POINTS

- Java has eight primitive types, and each numeric type has the same size on every platform.
- Whole-number literals are `int` and decimal literals are `double` unless you add `L` or `f`.
- Narrowing conversions need a cast, which cuts off any fraction.
- Integer division drops the fraction: `10 / 4` is `2`.
- Once a `String` is involved, `+` joins text from left to right.

TRY THESE EXERCISES

1. Write a program that declares one variable of each of the eight primitive types with a sensible value, such as a long mobile number with the `L` suffix and a float rate with the `f` suffix, and prints them all; then remove the `f` from the float value and note the compiler error.
2. Write a program that works out simple interest on a principal of `1_00_000` rupees at 7.5 percent a year for 3 years using double variables and a final constant for the rate, and prints the interest and the total amount; check that the interest comes to `22500.0`.
3. In your notebook, predict what `System.out.println` prints for `10 / 4`, `10 / 4.0`, `10 % 4`, `(int) 9.99`, `"Sum: " + 2 + 3` and `2 + 3 + " items"`, then print them all in a program and compare the results with your predictions.

Control Flow: Decisions and Loops

Reading input with Scanner

JAVA

```
import java.util.Scanner;

public class Table {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int n = sc.nextInt();

        if (n <= 0) {
            System.out.println("Please enter a positive number");
        } else {
            for (int i = 1; i <= 10; i++) {
                System.out.println(n + " x " + i + " = " + (n * i));
            }
        }
        sc.close();
    }
}
```

`Scanner`, from the `java.util` package, reads typed input: `nextInt()` and `nextDouble()` for numbers, `next()` for one word and `nextLine()` for a whole line.

if and else

`if`, `else if` and `else` work as in C, with one important difference: the condition must be a `boolean`. If `n` is an `int`, `if (n)` does not compile, and the classic mistake `if (n = 5)` is caught by the compiler instead of hiding as a bug. Write `if (n != 0)` instead.

Conditions are usually built from relational and logical operators, as in `if (age >= 18 && hasLicence)`. Because `&&` short-circuits, the right side is not evaluated at all when the left side is false.

The switch statement

A classic `switch` compares a value against `case` labels. In Java 17 it works with `int`, `char`, `byte`, `short`, their wrapper classes, `String` (since Java 7) and enums, but not with `long`, `float`, `double` or `boolean`. As in C, a matching case without `break` falls through into the next one:

JAVA FRAGMENT

```
int day = 6;
switch (day) {
    case 6:
    case 7:
        System.out.println("Weekend");
        break;
    default:
        System.out.println("Weekday");
}
```

Since Java 14, switch expressions with arrows are a standard feature. They never fall through, can list several labels in one case, and produce a value:

JAVA FRAGMENT

```
String dayName = "SAT";
String type = switch (dayName) {
    case "SAT", "SUN" -> "Weekend";
    default -> "Weekday";
};
System.out.println(type);    // Weekend
```

Loops

- `for (init; condition; update)` suits a known number of rounds.
- `while (condition)` checks before each round, so its body may run zero times.
- `do { ... } while (condition);` checks after each round, so its body always runs at least once.
- The enhanced `for`, as in `for (int m : marks)`, visits each element of an array or collection in order, without an index.

JAVA FRAGMENT

```
int n = 9045, digits = 0;
while (n > 0) {
    n /= 10;
    digits++;
}
System.out.println(digits);    // 4
```

`break` leaves the loop and `continue` jumps to the next round. To leave an outer loop from inside an inner one, put a label before the outer loop, such as `outer:`, and write `break outer;`.

Use the enhanced for loop when you only need each value, and a classic for loop when you also need the index.

KEY POINTS

- Scanner reads keyboard input with methods such as `nextInt` and `nextLine`.
- An `if` condition must be a `boolean`; an `int` is not accepted.
- A classic `switch` falls through without `break`; arrow-style `switch` expressions do not.
- `switch` cannot use `long`, `float`, `double` or `boolean` values.
- `do-while` always runs its body at least once.

TRY THESE EXERCISES

1. Write a program that reads marks out of 100 with Scanner and prints a grade with an `if`, `else if` and `else` ladder, printing `Invalid` for any mark below 0 or above 100.
2. Write a program that reads a three-letter day name such as `MON` or `SAT` and uses an arrow-style `switch` expression to store `Weekday` or `Weekend` in a `String`; then rewrite it as a classic `switch` with `break` statements and check that both versions give the same answer for all seven days.
3. Write a program that reads whole numbers with `nextInt` inside a `do-while` loop until the user enters 0, and then prints how many numbers were entered (not counting the 0), their sum and the largest of them.

CHAPTER 4

Arrays and Strings

Arrays

An array holds a fixed number of values of one type. In Java an array is an object, created with `new` or with an initialiser:

JAVA FRAGMENT

```
int[] marks = new int[5];           // five ints, all 0
String[] names = new String[3];     // three nulls
double[] prices = {45.0, 12.5, 180.0};
System.out.println(prices.length); // 3
```

- New elements get default values: `0` for numbers, `false` for `boolean` and `null` for objects.
- Indexes run from `0` to `length - 1`. `length` is a field, so it has no brackets after it.
- An index outside that range throws `ArrayIndexOutOfBoundsException` at run time. Java checks every access, unlike C.

Because an array's size cannot change, a program that does not know in advance how many items it will store usually uses an `ArrayList` instead, which you will meet in the last lesson.

JAVA

```
import java.util.Arrays;

public class Sales {
    public static void main(String[] args) {
        int[] sales = {1200, 950, 1430, 800, 1600};
        int total = 0;
        for (int s : sales) {
            total += s;
        }
        Arrays.sort(sales);
        System.out.println("Total: Rs " + total);           // Total: Rs
5980
        System.out.println(Arrays.toString(sales));        // [800,
950, 1200, 1430, 1600]
        System.out.println("Highest: Rs " + sales[sales.length - 1]); // Highest:
Rs 1600
    }
}
```

The class `java.util.Arrays` has helpers such as `sort`, `toString`, `fill` and `equals`. A two-dimensional array is an array of arrays: after `int[][] grid = new int[3][4]`; you can write `grid[1][2] = 7`;

Strings

`String` is a class, and string objects are immutable: methods such as `toUpperCase()` return a new string and leave the original unchanged. Java keeps a single shared copy of identical string literals, which is safe precisely because no one can change them.

JAVA FRAGMENT

```
String city = "Chandigarh";
System.out.println(city.length());           // 10
System.out.println(city.charAt(0));          // C
System.out.println(city.substring(0, 6));    // Chandi
System.out.println(city.indexOf("garh"));    // 6
System.out.println(city.toUpperCase());      // CHANDIGARH
System.out.println(city.contains("dig"));    // true
```

`substring(begin, end)` includes `begin` but stops before `end`, and `indexOf` returns `-1` when the text is not found. Other everyday methods are `trim()`, which removes spaces from both ends, `replace`, `split` and `isEmpty()`. For a string, `length()` is a method with brackets; for an array, `length` is a field.

Comparing strings

`==` checks whether two variables refer to the same object. To compare the characters, use `equals`, or `equalsIgnoreCase` to ignore case:

JAVA FRAGMENT

```
String a = "Delhi";
String b = new String("Delhi");
System.out.println(a == b);           // false
System.out.println(a.equals(b));      // true
```

`compareTo` gives alphabetical order as a negative number, zero or a positive number.

StringBuilder

Because strings are immutable, joining many pieces in a loop creates many temporary objects.

`StringBuilder` is a changeable buffer built for this job:

JAVA FRAGMENT

```
StringBuilder sb = new StringBuilder();
for (int i = 1; i <= 5; i++) {
    sb.append(i).append(' ');
}
System.out.println(sb.toString().trim()); // 1 2 3 4 5
```

KEY POINTS

- An array has a fixed size, and its elements start at default values such as 0 or null.
- Use `array.length` (a field) for arrays and `str.length()` (a method) for strings.
- A bad index throws `ArrayIndexOutOfBoundsException`.
- Strings are immutable; compare their text with `equals`, not `==`.
- Use `StringBuilder` to build a string from many pieces.

TRY THESE EXERCISES

1. Store a week's daily sales in an `int` array, print the total and the daily average using an enhanced for loop, then sort the array with `Arrays.sort`, print it with `Arrays.toString`, and print the lowest and highest values from its first and last elements.
2. Write a program that reads a full name with `nextLine` and prints its length, the name in capital letters, its first character, the position of the first space found with `indexOf`, and the first name taken out with `substring`; if `indexOf` returns -1 because there is no space, print the whole name as the first name instead of calling `substring`, which would throw an exception.
3. Write a program that reads one word with `next()`, builds its reverse with a `StringBuilder` by appending characters from the last index to the first with `charAt`, and uses `equalsIgnoreCase` to report whether the word is a palindrome; test it with Malayalam and Delhi.

Classes, Objects and Constructors

Classes and objects

A class describes a kind of thing: the data it holds, called fields, and what it can do, called methods. An object is one instance of a class, created with `new`, and each object has its own copy of the fields. Fields you do not set get default values, just like array elements: 0 for numbers, false for `boolean` and `null` for references. Local variables have no default and must be assigned before use.

JAVA

```
public class Student {
    private String name;
    private int rollNo;
    private static int count = 0;

    public Student(String name, int rollNo) {
        this.name = name;
        this.rollNo = rollNo;
        count++;
    }

    public Student(String name) {
        this(name, 0);
    }

    public String getName() {
        return name;
    }

    public int getRollNo() {
        return rollNo;
    }

    public static int getCount() {
        return count;
    }

    public static void main(String[] args) {
        Student s1 = new Student("Ishita", 21);
        Student s2 = new Student("Kabir");
        System.out.println(s1.getName() + " " + s1.getRollNo());
        System.out.println(s2.getName() + " " + s2.getRollNo());
        System.out.println("Students created: " + Student.getCount());
    }
}
```

The program prints `Ishita 21`, `Kabir 0` and `Students created: 2`.

Constructors

A constructor sets up a new object. It has the same name as the class and no return type.

- Java gives a class a default no-argument constructor only when the class declares no constructor at all. Once you write one, `new Student()` no longer compiles unless you also write a version with no parameters.
- Constructors can be overloaded. `this(name, 0)` calls another constructor of the same class, and in Java 17 it must be the first statement.
- `this.name = name` copies the parameter `name` into the field `name`. Without `this`, the parameter would hide the field.

Encapsulation

Making fields `private` and offering public methods, called getters and setters, is encapsulation. The class controls every change, so a setter can reject a negative roll number before storing it. Here `name` and `rollNo` can be read from outside the class but never changed.

static members

A `static` field belongs to the class rather than to any one object, so all objects share it. `count` above goes up by one for every new student. A `static` method, such as `getCount()` or `Math.sqrt()`, is called on the class name and cannot use instance fields directly, because it runs without an object.

References, not copies

A variable of a class type holds a reference to an object, not the object itself, so assigning it copies only the reference:

JAVA FRAGMENT

```
StringBuilder a = new StringBuilder("Pune");
StringBuilder b = a;           // b refers to the same object
b.append(" city");
System.out.println(a);       // Pune city
```

When no variable refers to an object any more, the garbage collector reclaims it automatically.

KEY POINTS

- A class defines fields and methods; `new` creates an object from it.
- Java supplies a default constructor only when a class declares none.
- `this.field = parameter` copies a constructor parameter into a field of the same name.
- A static field is shared by all objects of the class.
- Assigning an object variable copies the reference, not the object.

TRY THESE EXERCISES

1. Write a class `Book` with private fields `title`, `author` and `price`, a constructor that uses this to copy its parameters into the fields, and a getter for each field, then create three `Book` objects in `main` and print their details.
2. Add a setter `setPrice` to your `Book` class that refuses a negative price and prints a message instead, and add a second constructor that takes only a title and calls the first one with `this(...)`, passing `Unknown` as the author and `0` as the price.
3. Add a private static `int` counter to `Book` that goes up by one in the constructor and a static method that returns it, and print it after creating your books; then make two `StringBuilder` variables refer to the same object, change it through one and print it through the other.

CHAPTER 6

Inheritance, Interfaces and Packages

Inheritance with extends

A class can inherit fields and methods from another class with `extends`. The parent is the superclass and the child is the subclass. A Java class can have only one superclass, and every class ultimately inherits from `java.lang.Object`, which provides methods such as `toString()` and `equals()`.

- `super(...)` calls the superclass constructor and, in Java 17, must be the first statement in the subclass constructor.
- A subclass overrides a method by declaring one with the same name and parameters. Put `@Override` above it so the compiler checks that it really overrides something.
- A `final` method cannot be overridden and a `final` class cannot be extended. `String` is a final class.

Abstract classes and interfaces

An `abstract` class cannot be instantiated and may contain abstract methods, which have no body and must be implemented by its concrete subclasses. An interface is a contract: a set of methods a class promises to provide. A class `implements` an interface, and it can implement many interfaces while extending only one class. Interface methods are public and abstract unless written otherwise; since Java 8 an interface can also contain `default` and `static` methods with bodies.

```

interface Payable {
    double amountDue();
}

abstract class Employee implements Payable {
    protected final String name;

    Employee(String name) {
        this.name = name;
    }

    abstract String role();

    @Override
    public String toString() {
        return name + " (" + role() + ") Rs " + amountDue();
    }
}

class Permanent extends Employee {
    private final double salary;

    Permanent(String name, double salary) {
        super(name);
        this.salary = salary;
    }

    @Override
    String role() { return "Permanent"; }

    @Override
    public double amountDue() { return salary; }
}

class Contract extends Employee {
    private final int hours;
    private final double rate;

    Contract(String name, int hours, double rate) {
        super(name);
        this.hours = hours;
        this.rate = rate;
    }

    @Override
    String role() { return "Contract"; }

    @Override
    public double amountDue() { return hours * rate; }
}

public class Payroll {
    public static void main(String[] args) {

```

```

    Employee[] staff = {
        new Permanent("Neha Joshi", 42000),
        new Contract("Vikram Rao", 60, 350)
    };
    for (Employee e : staff) {
        System.out.println(e);
    }
}

```

This prints Neha Joshi (Permanent) Rs 42000.0 and Vikram Rao (Contract) Rs 21000.0. The loop works through `Employee` references, yet each object runs its own `amountDue()`. This is run-time polymorphism.

Packages and access

A package groups related classes. The `package` statement, such as `package com.shop.billing;`, must come first in a file, and the file sits in a matching folder. Classes from other packages are used with `import`, as in `import java.util.ArrayList;`, while `java.lang` classes such as `String`, `Math` and `System` are imported automatically.

Modifier	Same class	Same package	Subclass elsewhere	Any class
<code>private</code>	Yes	No	No	No
none (package-private)	Yes	Yes	No	No
<code>protected</code>	Yes	Yes	Yes	No
<code>public</code>	Yes	Yes	Yes	Yes

KEY POINTS

- extends inherits from one superclass; every class ultimately extends `Object`.
- In Java 17, `super(...)` must be the first statement of a subclass constructor.
- Use `@Override` so the compiler checks that a method really overrides.
- A class can implement many interfaces but extend only one class.
- `java.lang` is imported automatically; other packages need an import.

TRY THESE EXERCISES

1. Type the Payroll program from this lesson, add a third subclass Intern that is paid a fixed stipend, and add one Intern to the staff array without changing the loop that prints the staff.
2. Write an interface Shape with a method double area(), implement it in classes Circle and Rectangle using @Override (declare each area() public, because interface methods are public and a class cannot make them less accessible), store one of each in a Shape array and print every area in a loop; then misspell area in one class and note what the @Override annotation makes the compiler report.
3. Create a folder named billing containing a public class Invoice whose first line is package billing; and give it one public method and one method with no access modifier; then use Invoice from a class outside the folder with import billing.Invoice; compile both from the folder above billing, and note which of the two methods you can call.

CHAPTER 7

Exceptions and Collections

Handling exceptions

An exception is an object that signals an error at run time. If nothing handles it, the program stops and prints a stack trace. You handle exceptions with `try` and `catch`:

JAVA

```
public class SafeDivide {
    public static void main(String[] args) {
        String[] inputs = {"12", "0", "abc"};
        for (String text : inputs) {
            try {
                int n = Integer.parseInt(text);
                System.out.println("100 / " + n + " = " + (100 / n));
            } catch (ArithmeticException e) {
                System.out.println("Cannot divide by zero");
            } catch (NumberFormatException e) {
                System.out.println(text + " is not a number");
            } finally {
                System.out.println("Checked " + text);
            }
        }
    }
}
```

For "12" it prints `100 / 12 = 8`, for "0" it prints `Cannot divide by zero`, and for "abc" it prints `abc is not a number`. `Checked` is printed after every one of them.

- A `try` block can have several `catch` blocks, and the first one whose type matches runs.
- `finally` runs whether or not an exception occurred, so it suits clean-up work.

- Integer division by zero throws `ArithmeticException`, but dividing a non-zero `double` by zero gives `Infinity` or `-Infinity` instead.

Checked and unchecked exceptions

Checked exceptions, such as `IOException`, must be caught or declared in the method header with `throws`, or the code does not compile. Unchecked exceptions are subclasses of `RuntimeException`, such as `NullPointerException`, `ArithmeticException` and `NumberFormatException`; they usually signal bugs and need no declaration. You raise an exception yourself with `throw`, as in `throw new IllegalArgumentException("Amount must be positive")`; . Since Java 7, try-with-resources closes a resource such as a file reader automatically when the block ends.

ArrayList and HashMap

The collections framework in `java.util` provides ready-made data structures. `ArrayList` is a resizable list that keeps insertion order and allows duplicates. `HashMap` stores key and value pairs with unique keys.

JAVA

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class Shop {
    public static void main(String[] args) {
        List<String> cart = new ArrayList<>();
        cart.add("Pen");
        cart.add("Notebook");
        cart.add("Pen");
        System.out.println(cart + " size " + cart.size());    // [Pen, Notebook,
Pen] size 3

        Map<String, Integer> price = new HashMap<>();
        price.put("Pen", 10);
        price.put("Notebook", 45);
        price.put("Pen", 12);                                // replaces 10

        int total = 0;
        for (String item : cart) {
            total += price.getDefault(item, 0);
        }
        System.out.println("Total: Rs " + total);            // Total: Rs 69
    }
}
```

- `ArrayList` methods include `add`, `get(index)`, `set`, `remove`, `size()` and `contains`.
- Calling `put` with an existing key replaces its value. `get` returns `null` for a missing key, while `getOrDefault` lets you choose the fallback.

- A `HashMap` keeps no particular order. Use `LinkedHashMap` for insertion order or `TreeMap` for sorted keys.
- Collections hold objects, so a list of numbers is `ArrayList<Integer>`; Java converts between `int` and `Integer` automatically, which is called autoboxing.

KEY POINTS

- Code that may fail goes in try; catch blocks handle specific exception types.
- `finally` runs whether or not an exception was thrown.
- Checked exceptions such as `IOException` must be caught or declared with `throws`.
- `ArrayList` is a resizable, ordered list that allows duplicates.
- `HashMap` keys are unique: put with an existing key replaces its value.

TRY THESE EXERCISES

1. Write a program that reads two values as text, converts them with `Integer.parseInt` and divides the first by the second inside try, with separate catch blocks for `NumberFormatException` and `ArithmeticException` and a finally block that prints Done; test it with 20 and 4, 20 and 0, and 20 and abc.
2. Write a method `withdraw(double amount)` that throws an `IllegalArgumentException` with a clear message when the amount is zero or negative, call it from main inside try with one valid and one negative amount, and print the message in the catch block with `getMessage()`.
3. Write a shopping-list program that adds five items to an `ArrayList` of `String`, removes one, uses `contains` to check whether Milk is on the list and prints the size; then store each item's price in a `HashMap` of `String` to `Integer` and print the total bill using `getOrDefault`.

CHECK YOURSELF

Practice questions

Choose the best answer for each question. The answers, with a short explanation, are in the next chapter.

1. Which JDK tool compiles a .java source file into bytecode?

- a) java
- b) javadoc
- c) jar
- d) javac

2. What is the job of the Java Virtual Machine (JVM)?

- a) It compiles .java source files into .class bytecode files
- b) It stores Java programs online so other computers can download them
- c) It checks the spelling and grammar of comments in the code
- d) It runs Java bytecode, translating it for the computer it is running on

3. Compiling a file that contains public class Hello with javac produces which file?

- a) Hello.class
- b) Hello.exe
- c) Hello.obj
- d) Hello.jar

4. A source file contains public class Invoice. What must the file be named?

- a) invoice.java
- b) Invoice.class
- c) Main.java
- d) Invoice.java

5. Why is Java described as platform independent?

- a) Java programs do not need an operating system
- b) Its compiled bytecode runs on any operating system that has a JVM
- c) javac produces separate machine code for every operating system at once
- d) Java programs can run only inside a web browser

6. How many primitive data types does Java have?

- a) 8
- b) 4
- c) 6
- d) 10

7. What does System.out.println(10 / 4); print?

- a) 2.5
- b) 2.0
- c) 2
- d) 3

8. What does System.out.println("Sum: " + 2 + 3); print?

- a) Sum: 23
- b) Sum: 5
- c) Sum: 2 + 3
- d) It does not compile

9. Which of these declarations compiles in Java?

- a) float rate = 18.5;
- b) float rate = 18.5f;
- c) int qty = 10.0;
- d) char c = "A";

10. What is the size of an int in Java?

- a) 16 bits on every platform
- b) It depends on the operating system
- c) 32 bits on every platform
- d) 64 bits on 64-bit computers

11. What does this code print? int x = 3; switch (x) { case 1: System.out.print("one "); case 3: System.out.print("three "); case 4: System.out.print("four "); default: System.out.print("other"); }

- a) three
- b) three other
- c) one three four other
- d) three four other

12. A classic switch statement in Java 17 cannot switch on a value of which type?

- a) String
- b) char
- c) int
- d) double

13. In Java, what happens with int n = 5; if (n) { System.out.println("yes"); } ?

- a) It does not compile, because an if condition must be a boolean
- b) It prints yes, because any non-zero int counts as true
- c) It prints nothing, because only 1 counts as true in Java
- d) It compiles, but throws an exception when it runs

- 14. How many times is Hi printed? `int i = 0; do { System.out.println("Hi"); i++; } while (i < 0);`**
- a) Never
 - b) Once
 - c) Twice
 - d) Forever
- 15. What does the enhanced for loop for (`int m : marks`) do?**
- a) It visits only the first element of marks
 - b) It visits each element of marks in turn, storing it in m
 - c) It visits the indexes of marks instead of the elements
 - d) It sorts marks before looping
- 16. What is the default value of each element in `new int[4]`?**
- a) null
 - b) 0
 - c) 1
 - d) An unpredictable value
- 17. How do you get the number of elements in an array named `prices`?**
- a) `prices.length()`
 - b) `prices.length`
 - c) `prices.size()`
 - d) `prices.count()`
- 18. What does `"Mumbai".substring(1, 4)` return?**
- a) umba
 - b) Mum
 - c) umb
 - d) mba
- 19. Why should strings be compared with `equals()` rather than `==`?**
- a) `==` cannot be used with String objects, so the code does not compile at all
 - b) `equals()` runs faster than `==` because it compares only the lengths of the strings
 - c) `==` checks whether both variables refer to the same object, while `equals()` compares the characters
 - d) `==` ignores the difference between capital and small letters, while `equals()` does not
- 20. What happens when a program reads `marks[5]` from an array created with `new int[5]`?**
- a) It returns 0, the default value for an int element
 - b) It wraps around and returns the last element instead
 - c) It returns null because that element does not exist
 - d) An `ArrayIndexOutOfBoundsException` is thrown at run time
- 21. Which keyword creates a new object in Java?**
- a) create
 - b) object
 - c) new
 - d) make
- 22. When does Java automatically provide a no-argument default constructor for a class?**
- a) Always, for every class
 - b) Only when the class has static methods
 - c) Only when the class declares no constructor at all
 - d) Never
- 23. Inside a constructor with a parameter called `name`, what does `this.name = name;` do?**
- a) It copies the field `name` into the parameter `name`
 - b) It creates a new object called `this`
 - c) It copies the parameter `name` into the object's field `name`
 - d) It calls another constructor

24. A class has private static int count = 0; and its constructor runs count++. After three objects are created, what is count?
- a) 1
 - b) 0
 - c) 3
 - d) 6
25. What does this code print? class Box { int size; } then in main: Box a = new Box(); a.size = 5; Box b = a; b.size = 9; System.out.println(a.size);
- a) 5
 - b) 9
 - c) 0
 - d) 14
26. Which keyword does a Java class use to inherit from another class?
- a) extends
 - b) implements
 - c) inherits
 - d) super
27. Can a Java class extend two classes at the same time?
- a) Yes, by listing both classes after extends, separated by a comma
 - b) Yes, but only if both of the parent classes are declared abstract
 - c) No, and once it extends a class it cannot implement any interfaces either
 - d) No, a class can extend only one class, though it can implement many interfaces
28. What does this code print? class Animal { String sound() { return "Some sound"; } } class Dog extends Animal { String sound() { return "Woof"; } } then: Animal a = new Dog(); System.out.println(a.sound());
- a) Woof
 - b) Some sound
 - c) null
 - d) It does not compile
29. Which package is imported automatically into every Java source file?
- a) java.util
 - b) java.lang
 - c) java.io
 - d) java.awt
30. From where can a field with no access modifier (package-private) be accessed?
- a) Only from classes in the same package
 - b) From every class in the program
 - c) Only from inside the same class
 - d) Only from subclasses in other packages
31. Which block runs whether or not an exception is thrown in the try block?
- a) catch
 - b) throw
 - c) throws
 - d) finally
32. Which of these is a checked exception that must be caught or declared with throws?
- a) ArithmeticException
 - b) NullPointerException
 - c) IOException
 - d) ArrayIndexOutOfBoundsException
33. What does Integer.parseInt("12a") do?
- a) It throws a NumberFormatException
 - b) It returns 12 and ignores the letter
 - c) It returns 0 because the text is not a valid number
 - d) It returns null to show that parsing failed

16. **(b) 0.** Java initialises the elements of a new numeric array to zero, boolean elements to false and object elements to null.
17. **(b) prices.length.** length is a field of every array, written without brackets; length() with brackets is the String method.
18. **(c) umb.** substring includes the start index 1 and stops before index 4, giving the characters at positions 1, 2 and 3.
19. **(c) == checks whether both variables refer to the same object, while equals() compares the characters.** Two different String objects can hold the same text, and only equals() compares the text itself.
20. **(d) An ArrayIndexOutOfBoundsException is thrown at run time.** The valid indexes are 0 to 4, and Java checks every array access, throwing an exception for a bad index.
21. **(c) new.** new allocates the object, runs its constructor and returns a reference to it.
22. **(c) Only when the class declares no constructor at all.** As soon as a class declares any constructor, Java stops providing the default one.
23. **(c) It copies the parameter name into the object's field name.** this.name refers to the field of the current object, while plain name refers to the parameter, which hides the field.
24. **(c) 3.** A static field belongs to the class and is shared by all objects, so each constructor call adds to the same count.
25. **(b) 9.** Box b = a copies the reference, so a and b refer to the same object, and changing b.size changes what a.size shows.
26. **(a) extends.** A class extends one superclass; implements is used for interfaces.
27. **(d) No, a class can extend only one class, though it can implement many interfaces.** Java allows single inheritance of classes, and interfaces give a class the ability to take on several contracts.
28. **(a) Woof.** The method that runs is chosen by the actual object, a Dog, even though the variable's type is Animal.
29. **(b) java.lang.** Classes such as String, Math and System come from java.lang, which never needs an import statement.
30. **(a) Only from classes in the same package.** With no modifier, access is limited to the package; protected adds subclasses in other packages, and public opens it to all.
31. **(d) finally.** A finally block runs after the try block and any matching catch, whether or not an exception was thrown, which makes it the place for clean-up code.
32. **(c) IOException.** IOException is checked; the other three are subclasses of RuntimeException, which are unchecked.
33. **(a) It throws a NumberFormatException.** parseInt accepts only a valid whole number, and "12a" contains a letter, so it throws NumberFormatException.
34. **(a) 3.** An ArrayList allows duplicate values, so all three additions are kept and the size is 3.
35. **(d) The old value for that key is replaced by the new one.** Keys in a HashMap are unique, so put overwrites the value stored for an existing key.

Glossary

Abstract class

A class declared with the keyword `abstract` that cannot be instantiated and may contain abstract methods without a body, which its concrete subclasses must implement.

ArrayList

A resizable list class in `java.util` that keeps its elements in insertion order, allows duplicates and grows automatically as items are added.

Autoboxing

Java's automatic conversion of a primitive value such as an `int` into an object of its wrapper class such as `Integer`, which lets collections like `ArrayList` hold numbers; the reverse conversion, from `Integer` back to `int`, is called unboxing.

Bytecode

The instructions in a `.class` file produced by `javac`, which are not tied to any particular processor and are run by the Java Virtual Machine.

Checked exception

An exception such as `IOException` that the compiler forces you either to catch or to declare in the method header with `throws`.

Class

A blueprint that describes the fields an object holds and the methods it provides, from which objects are created with `new`.

Constructor

A special block of code with the same name as its class and no return type that sets up a new object when it is created with `new`.

Encapsulation

Keeping a class's fields private and allowing access only through public methods such as getters and setters, so that the class controls every change to its data.

final

A keyword that stops a variable from being assigned a new value once it has one (for an object variable the reference is fixed, though the object itself may still change), stops a method from being overridden, or stops a class from being extended.

Garbage collector

The part of the JVM that automatically frees the memory of objects the program can no longer reach, for example because no variable refers to them any more, which is why Java has no `free` or `delete`.

HashMap

A collection class in `java.util` that stores key and value pairs with unique keys and keeps them in no particular order.

Immutable

Describes an object whose contents cannot change after it is created, such as a `String`, whose methods return new strings instead of changing the original.

Inheritance

The feature that lets a subclass take on the fields and methods of a superclass using the keyword `extends`; a Java class can extend only one class.

Interface

A contract listing methods that a class promises to provide, which a class adopts with the keyword implements; one class can implement many interfaces.

JDK

The Java Development Kit, which contains everything needed to run Java programs plus development tools such as the javac compiler, jar and javadoc.

JRE

The Java Runtime Environment, made up of the JVM and the standard class libraries, which is enough to run Java programs but not to compile them; in current versions it usually comes as part of the JDK.

JVM

The Java Virtual Machine, the program that runs bytecode on a particular operating system and manages memory, which lets the same .class file run on Windows, Linux and macOS.

Method overriding

Redefining, in a subclass, a method inherited from its superclass with the same name and parameters, usually marked with @Override so that the compiler checks it.

Object

One instance of a class, created with new, that has its own copy of the class's instance fields.

Package

A named group of related classes, declared with a package statement at the top of each file, whose classes are used from other packages through import.

Primitive type

One of Java's eight built-in basic types, byte, short, int, long, float, double, char and boolean, whose numeric sizes are fixed by the language on every platform.

Reference type

Any type that is not primitive, such as String, arrays and your own classes, whose variables hold a reference to an object or the value null.

static

A keyword marking a field or method that belongs to the class itself rather than to any one object, so a static field is shared by all objects of that class.

StringBuilder

A class that holds a changeable sequence of characters and is used to build a string efficiently from many pieces, for example inside a loop.

Unchecked exception

An exception that is a subclass of RuntimeException, such as NullPointerException or ArithmeticException, which usually signals a bug and need not be caught or declared.



Gurukul Technicals Institute

Computer education institute in Delhi, with authorised member centres, robotics classes and science projects, working models and DIY kits.

COURSES

Computer & Office Skills · Accounting & Business Software · Programming & Web Development · Design & Digital Skills · Hardware & Networking · Robotics & Coding · Other Programmes

PROJECTS AND KITS

Ready-made school projects · DIY and STEM kits · Robotics and coding · Components and spares · Engineering projects

CONTACT

Head office: A-10, Chief Commissioner Colony, opposite Rana Pratap Bagh, Delhi 110007

Phone: 011 2721 1512 · **Mobile and WhatsApp:** +91 97116 99511

Email: engineergurukul@yahoo.co.in

Website: engineergurukul.com · **Instagram:** [@science_projects_club](https://www.instagram.com/science_projects_club)

More free ebooks for computer courses, robotics and science projects: engineergurukul.com/resources/ebooks/

Free to read, print and share. Not for sale.