CORE JAVA

Complete Student Guide

All 23 Chapters — From Basics to JFC Swing

Chapters Covered:

Chapter 1: Introduction to Java

1.1 History of Java

Java was developed by James Gosling at Sun Microsystems in 1991. Originally called 'Oak', it was renamed Java in 1995. Java is a platform-independent, object-oriented programming language that follows the principle of 'Write Once, Run Anywhere' (WORA).

1.2 Features of Java

Feature	Description
Simple	Java syntax is clean and easy to learn, based on C/C++
Object-Oriented	Everything in Java is an object — supports OOP concepts
Platform Independent	Java bytecode runs on any OS with JVM installed
Secure	Has built-in security features; no pointer arithmetic
Robust	Strong type checking, exception handling, garbage collection
Multithreaded	Built-in support for concurrent programming
Distributed	RMI and sockets make networking easy
Dynamic	Java programs can adapt at runtime by loading new classes
Portable	Same Java code runs identically on any platform
High Performance	JIT compiler converts bytecode to native machine code

1.3 How Java Works – JVM, JRE, JDK

Component	Description
JVM (Java Virtual Machine)	Executes Java bytecode; platform-specific but bytecode is universal
JRE (Java Runtime Environment)	JVM + libraries needed to run Java programs
JDK (Java Development Kit)	JRE + compiler (javac) + development tools

1.4 Java vs C vs C++

Feature	Java	C / C++
Platform	Platform Independent	Platform Dependent
Pointers	No pointers	Supports pointers
Memory Management	Automatic (Garbage Collector)	Manual (malloc/free)
Multiple Inheritance	Via interfaces only	Supported directly

Compilation	Source → Bytecode → JVM	Source → Machine Code
Security	Built-in security	Limited

1.5 Java Program Execution Steps

1. Write source code in a .java file
2. Compile using: javac HelloWorld.java (creates HelloWorld.class)
3. Run using: java HelloWorld (JVM executes the bytecode)

First Java Program

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

Output:

```
Hello, World!
```

Note: Every Java program must have a class whose name matches the filename. The main() method is the entry point.

Chapter 2: Java Basics – Data Types, Variables, Operators

2.1 Structure of a Java Program

```
// Comments
import java.util.*;            // Import packages

public class MyClass {         // Class declaration
    // Instance variables
    int x = 10;

    public static void main(String[] args) {  // main method
        // Statements
        System.out.println("Running!");
    }
}
```

2.2 Primitive Data Types

Data Type	Size / Range / Example
byte	8-bit | -128 to 127 | byte b = 100;
short	16-bit | -32768 to 32767 | short s = 5000;
int	32-bit | -2,147,483,648 to 2,147,483,647 | int x = 42;
long	64-bit | Very large integers | long l = 99999L;
float	32-bit decimal | float f = 3.14f;
double	64-bit decimal | double d = 3.14159;
char	16-bit Unicode character | char c = 'A';
boolean	true or false | boolean flag = true;

2.3 Variables and Constants

```
public class Variables {
    public static void main(String[] args) {
        int age = 20;           // integer variable
        double salary = 55000.50; // double variable
        char grade = 'A';       // character variable
        boolean active = true;  // boolean variable
        final int MAX = 100;    // constant (final)

        System.out.println("Age: " + age);
        System.out.println("Salary: " + salary);
        System.out.println("Grade: " + grade);
        System.out.println("Active: " + active);
        System.out.println("Max: " + MAX);
    }
}
```

Output:

```
Age: 20
Salary: 55000.5
Grade: A
Active: true
```

```
Max: 100
```

2.4 Type Casting

Type	Description
Widening (Implicit)	Smaller → Larger type automatically: int x = 10; double d = x;
Narrowing (Explicit)	Larger → Smaller requires cast: double d = 9.99; int x = (int)d; // x=9

2.5 Operators

Arithmetic Operators

```
public class ArithOps {
    public static void main(String[] args) {
        int a = 20, b = 6;
        System.out.println("a + b = " + (a + b));  // 26
        System.out.println("a - b = " + (a - b));  // 14
        System.out.println("a * b = " + (a * b));  // 120
        System.out.println("a / b = " + (a / b));  // 3 (integer division)
        System.out.println("a % b = " + (a % b));  // 2 (remainder)
    }
}
```

```
Output:
a + b = 26
a - b = 14
a * b = 120
a / b = 3
a % b = 2
```

Relational & Logical Operators

Operator	Meaning	Example
==	Equal to	a == b → false
!=	Not equal	a != b → true
>	Greater than	a > b → true
<	Less than	a < b → false
&&	Logical AND	a>5 && b>5 → true
||	Logical OR	a>5 || b>30 → true
!	Logical NOT	!(a==b) → true

Increment & Decrement

```
int x = 5;
System.out.println(x++);  // prints 5, then x becomes 6 (post-increment)
System.out.println(++x);  // x becomes 7, then prints 7 (pre-increment)
System.out.println(x--);  // prints 7, then x becomes 6 (post-decrement)
System.out.println(--x);  // x becomes 5, then prints 5 (pre-decrement)
```

Ternary Operator

```
// Syntax: condition ? value_if_true : value_if_false
int a = 10, b = 20;
int max = (a > b) ? a : b;
System.out.println("Max = " + max);  // Max = 20
```

2.6 Input using Scanner

```
import java.util.Scanner;

public class InputDemo {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = sc.nextLine();
        System.out.print("Enter your age: ");
        int age = sc.nextInt();
        System.out.println("Hello " + name + ", you are " + age + " years 
old.");
    }
}
```

Chapter 3: Control Statements

3.1 if / else if / else

```
public class IfDemo {
    public static void main(String[] args) {
        int marks = 75;
        if (marks >= 75) {
            System.out.println("Distinction");
        } else if (marks >= 60) {
            System.out.println("First Class");
        } else if (marks >= 40) {
            System.out.println("Pass");
        } else {
            System.out.println("Fail");
        }
    }
}
```

Output:

```
Distinction
```

3.2 switch Statement

```
public class SwitchDemo {
    public static void main(String[] args) {
        int day = 3;
        switch(day) {
            case 1: System.out.println("Monday"); break;
            case 2: System.out.println("Tuesday"); break;
            case 3: System.out.println("Wednesday"); break;
            case 4: System.out.println("Thursday"); break;
            case 5: System.out.println("Friday"); break;
            default: System.out.println("Weekend");
        }
    }
}
```

Output:

```
Wednesday
```

3.3 Loops

while Loop

```
public class WhileDemo {
    public static void main(String[] args) {
        int i = 1;
        while (i <= 5) {
            System.out.println("Count: " + i);
            i++;
        }
    }
}
```

Output:

```
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
```

do-while Loop

```
// do-while executes at least once
int i = 1;
do {
    System.out.println("Number: " + i);
    i++;
} while (i <= 3);
```

```
Output:
Number: 1
Number: 2
Number: 3
```

for Loop

```
public class ForDemo {
    public static void main(String[] args) {
        // Print multiplication table of 5
        for (int i = 1; i <= 10; i++) {
            System.out.println("5 x " + i + " = " + (5 * i));
        }
    }
}
```

```
Output:
5 x 1 = 5
5 x 2 = 10
...
5 x 10 = 50
```

Nested Loops – Multiplication Table

```
public class NestedLoop {
    public static void main(String[] args) {
        for (int i = 1; i <= 3; i++) {
            for (int j = 1; j <= 3; j++) {
                System.out.print(i * j + "\t");
            }
            System.out.println();
        }
    }
}
```

```
Output:
1  2  3
2  4  6
3  6  9
```

3.4 break and continue

```
// break: exits the loop immediately
for (int i = 1; i <= 10; i++) {
    if (i == 5) break;
    System.out.println(i);
}
// Output: 1 2 3 4

// continue: skips current iteration
for (int i = 1; i <= 5; i++) {
    if (i == 3) continue;
    System.out.println(i);
}
// Output: 1 2 4 5
```

Chapter 4: Arrays

4.1 Introduction to Arrays

An array is a collection of elements of the same data type stored in contiguous memory locations. Arrays in Java are objects and are always allocated on the heap.

Concept	Details
Declaration	int[] arr; or int arr[];
Initialization	int[] arr = new int[5];
Index	Zero-based: arr[0] is the first element
Length	arr.length gives number of elements

4.2 Single-Dimensional Arrays

```
public class ArrayDemo {
    public static void main(String[] args) {
        // Declare and initialize
        int[] marks = {85, 90, 75, 92, 88};

        int sum = 0;
        for (int i = 0; i < marks.length; i++) {
            sum += marks[i];
        }
        double avg = (double) sum / marks.length;
        System.out.println("Total: " + sum);
        System.out.println("Average: " + avg);
    }
}
```

Output:

```
Total: 430
Average: 86.0
```

4.3 Two-Dimensional Arrays

```
public class Matrix {
    public static void main(String[] args) {
        int[][] matrix = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };

        System.out.println("Matrix:");
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print(matrix[i][j] + "  ");
            }
            System.out.println();
        }
    }
}
```

Output:

```
Matrix:
```

```
1  2  3
4  5  6
7  8  9
```

4.4 Array Sorting using Arrays.sort()

```
import java.util.Arrays;

public class SortDemo {
    public static void main(String[] args) {
        int[] nums = {64, 25, 12, 22, 11};
        System.out.println("Before: " + Arrays.toString(nums));
        Arrays.sort(nums);
        System.out.println("After:  " + Arrays.toString(nums));

        // Binary Search
        int pos = Arrays.binarySearch(nums, 25);
        System.out.println("25 found at index: " + pos);
    }
}
```

```
Output:
Before: [64, 25, 12, 22, 11]
After:  [11, 12, 22, 25, 64]
25 found at index: 3
```

Chapter 5: Methods

5.1 Introduction

A method is a block of code that performs a specific task. Methods avoid code repetition and make programs modular.

Method Type	Syntax
No args, No return	void greet() { ... }
With args, No return	void add(int a, int b) { ... }
No args, With return	int getValue() { return x; }
With args, With return	int add(int a, int b) { return a+b; }

5.2 Method Examples

Type 1: No Arguments, No Return Value

```
public class MethodDemo1 {
    void greet() {
        System.out.println("Hello from greet() method!");
    }
    public static void main(String[] args) {
        MethodDemo1 obj = new MethodDemo1();
        obj.greet();
    }
}
```

```
Output:
Hello from greet() method!
```

Type 2: With Arguments, No Return Value

```
public class MethodDemo2 {
    void add(int a, int b) {
        int sum = a + b;
        System.out.println("Sum = " + sum);
    }
    public static void main(String[] args) {
        MethodDemo2 obj = new MethodDemo2();
        obj.add(15, 25);
    }
}
```

```
Output:
Sum = 40
```

Type 3: No Arguments, With Return Value

```
public class MethodDemo3 {
    int getSquare() {
        int x = 7;
        return x * x;
    }
    public static void main(String[] args) {
        MethodDemo3 obj = new MethodDemo3();
        int result = obj.getSquare();
        System.out.println("Square = " + result);
    }
}
Output:
Square = 49
```

Type 4: With Arguments, With Return Value

```
public class MethodDemo4 {
    int multiply(int a, int b) {
        return a * b;
    }
    public static void main(String[] args) {
        MethodDemo4 obj = new MethodDemo4();
        System.out.println("Product = " + obj.multiply(6, 7));
    }
}
Output:
Product = 42
```

5.3 Method Overloading

Multiple methods with the same name but different parameters (different number or type).

```
public class Overload {
    int add(int a, int b) { return a + b; }
    double add(double a, double b) { return a + b; }
    int add(int a, int b, int c) { return a + b + c; }

    public static void main(String[] args) {
        Overload obj = new Overload();
        System.out.println(obj.add(5, 10));        // 15
        System.out.println(obj.add(1.5, 2.5));    // 4.0
        System.out.println(obj.add(1, 2, 3));      // 6
    }
}
Output:
15
4.0
6
```

5.4 Recursion

```
public class Factorial {
    int fact(int n) {
        if (n == 0 || n == 1) return 1;
        return n * fact(n - 1);  // recursive call
    }
    public static void main(String[] args) {
        Factorial obj = new Factorial();
        System.out.println("5! = " + obj.fact(5));
    }
}
Output:
5! = 120
```

Chapter 6: Object-Oriented Programming – Classes & Objects

6.1 OOP Concepts

Concept	Description
Class	Blueprint/template for creating objects
Object	Instance of a class; has state and behavior
Encapsulation	Bundling data and methods together inside a class
Inheritance	A class acquires properties of another class
Polymorphism	Same method name, different behaviors
Abstraction	Hiding implementation details, showing only essentials

6.2 Defining a Class and Creating Objects

```
public class Student {
    // Instance variables (attributes)
    String name;
    int age;
    double marks;

    // Method (behavior)
    void display() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Marks: " + marks);
    }

    public static void main(String[] args) {
        Student s1 = new Student();  // create object
        s1.name = "Alice";
        s1.age = 20;
        s1.marks = 92.5;
        s1.display();
    }
}
```

Output:

```
Name: Alice
Age: 20
Marks: 92.5
```

6.3 Constructors

A constructor is a special method called automatically when an object is created. It has the same name as the class and no return type.

```
public class BankAccount {
    String owner;
    double balance;

    // Default constructor
    BankAccount() {
        owner = "Unknown";
        balance = 0.0;
    }

    // Parameterized constructor
    BankAccount(String name, double amount) {
        owner = name;
        balance = amount;
    }

    void display() {
        System.out.println(owner + " -> Rs. " + balance);
    }

    public static void main(String[] args) {
        BankAccount acc1 = new BankAccount();
        BankAccount acc2 = new BankAccount("Bob", 50000);
        acc1.display();
        acc2.display();
    }
}
```

```
Output:
Unknown -> Rs. 0.0
Bob -> Rs. 50000.0
```

6.4 Access Specifiers

Specifier	Access Scope
public	Accessible from anywhere
private	Accessible only within the same class
protected	Accessible within same class, subclasses, and same package
default (no keyword)	Accessible within the same package only

6.5 this Keyword

```
public class Employee {
    String name;
    int id;

    Employee(String name, int id) {
        this.name = name;  // 'this' refers to current object's variable
        this.id = id;
    }

    void display() {
        System.out.println("ID: " + this.id + ", Name: " + this.name);
    }

    public static void main(String[] args) {
        Employee e = new Employee("Carol", 101);
        e.display();
    }
}
```

Output:

```
ID: 101, Name: Carol
```

6.6 static Keyword

```
public class Counter {
    static int count = 0;  // shared among all objects

    Counter() { count++; }

    public static void main(String[] args) {
        Counter c1 = new Counter();
        Counter c2 = new Counter();
        Counter c3 = new Counter();
        System.out.println("Objects created: " + Counter.count);
    }
}
```

Output:

```
Objects created: 3
```

Chapter 7: Inheritance

7.1 Introduction

Inheritance allows a new class (subclass/child) to acquire properties and methods of an existing class (superclass/parent). It promotes code reuse. The keyword extends is used.

Type	Description
Single Inheritance	One child inherits from one parent
Multilevel Inheritance	A → B → C (chain of inheritance)
Hierarchical Inheritance	Multiple children from one parent

7.2 Single Inheritance

```
class Animal {
    String name = "Animal";
    void breathe() {
        System.out.println(name + " breathes air.");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("Dog barks! Woof!");
    }
}

public class InheritDemo {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.name = "Rex";
        d.breathe();  // inherited from Animal
        d.bark();     // Dog's own method
    }
}
```

```
Output:
Rex breathes air.
Dog barks! Woof!
```

7.3 Multilevel Inheritance

```
class Vehicle { void move() { System.out.println("Vehicle moves"); } }
class Car extends Vehicle { void honk() { System.out.println("Car honks"); } }
class SportsCar extends Car { void turbo() { System.out.println("Turbo
ON!"); } }

public class MultiLevel {
    public static void main(String[] args) {
        SportsCar sc = new SportsCar();
        sc.move();    // from Vehicle
        sc.honk();    // from Car
        sc.turbo();   // from SportsCar
    }
}
```

```
Output:
```

```
Vehicle moves
Car honks
Turbo ON!
```

7.4 super Keyword

```
class Shape {
    String color;
    Shape(String color) {
        this.color = color;
        System.out.println("Shape constructor: " + color);
    }
}

class Circle extends Shape {
    double radius;
    Circle(String color, double radius) {
        super(color);  // calls parent constructor
        this.radius = radius;
        System.out.println("Circle radius: " + radius);
    }
}

public class SuperDemo {
    public static void main(String[] args) {
        Circle c = new Circle("Red", 5.5);
    }
}
```

Output:

```
Shape constructor: Red
Circle radius: 5.5
```

7.5 Method Overriding

```
class Parent {
    void show() { System.out.println("Parent show()"); }
}

class Child extends Parent {
    @Override
    void show() { System.out.println("Child show() - overrides Parent"); }
}

public class OverrideDemo {
    public static void main(String[] args) {
        Parent p = new Parent();
        p.show();  // Parent show()
        Child c = new Child();
        c.show();  // Child show()
        // Runtime polymorphism:
        Parent ref = new Child();
        ref.show(); // Child show() -- decided at runtime
    }
}
```

Output:

```
Parent show()
Child show() - overrides Parent
Child show() - overrides Parent
```

7.6 Abstract Classes

An abstract class cannot be instantiated directly. It may have abstract methods (no body) that must be implemented by subclasses.

```
abstract class Shape {
    abstract double area();  // abstract method – no body
    void display() {
        System.out.println("Area = " + area());
    }
}

class Rectangle extends Shape {
    double l, w;
    Rectangle(double l, double w) { this.l = l; this.w = w; }
    double area() { return l * w; }
}

class Circle extends Shape {
    double r;
    Circle(double r) { this.r = r; }
    double area() { return 3.14159 * r * r; }
}

public class AbstractDemo {
    public static void main(String[] args) {
        Shape s1 = new Rectangle(4, 5);
        Shape s2 = new Circle(3);
        s1.display();
        s2.display();
    }
}
```

```
Output:
Area = 20.0
Area = 28.27431
```

7.7 final Keyword

Usage	Effect
final variable	Constant – cannot be changed: final int MAX = 100;
final method	Cannot be overridden in a subclass
final class	Cannot be extended/inherited: final class MyClass {}

Chapter 8: Interfaces

8.1 Introduction

An interface is a fully abstract type that defines a contract. All methods in an interface are public and abstract by default. A class uses the implements keyword to fulfill the interface contract.

Key difference: While a class can extend only ONE class, it can implement MULTIPLE interfaces, achieving multiple inheritance in Java.

8.2 Defining and Implementing an Interface

```
interface Printable {
    void print();  // abstract by default
}

class Document implements Printable {
    public void print() {
        System.out.println("Printing document...");
    }
}

public class InterfaceDemo {
    public static void main(String[] args) {
        Document d = new Document();
        d.print();
    }
}
```

```
Output:
Printing document...
```

8.3 Multiple Interfaces (Multiple Inheritance)

```
interface Flyable {
    void fly();
}

interface Swimmable {
    void swim();
}

class Duck implements Flyable, Swimmable {
    public void fly()  { System.out.println("Duck flies!"); }
    public void swim() { System.out.println("Duck swims!"); }
}

public class MultiInterface {
    public static void main(String[] args) {
        Duck d = new Duck();
        d.fly();
        d.swim();
    }
}
```

```
Output:
Duck flies!
Duck swims!
```

8.4 Interface with Constants

```
interface MathConstants {
    double PI = 3.14159;  // public static final by default
    double E  = 2.71828;
}

class Calculator implements MathConstants {
    double circleArea(double r) {
        return PI * r * r;  // using interface constant
    }
}

public class ConstantInterface {
    public static void main(String[] args) {
        Calculator c = new Calculator();
        System.out.println("Area = " + c.circleArea(5));
    }
}
```

```
Output:
Area = 78.53975
```

8.5 Extending Interfaces

```
interface A { void methodA(); }
interface B extends A { void methodB(); }

class MyClass implements B {
    public void methodA() { System.out.println("Method A"); }
    public void methodB() { System.out.println("Method B"); }
}

public class ExtendInterface {
    public static void main(String[] args) {
        MyClass obj = new MyClass();
        obj.methodA();
        obj.methodB();
    }
}
```

```
Output:
Method A
Method B
```

Abstract Class vs Interface	Details
Abstract Class	Can have constructors, instance variables, concrete methods
Interface	No constructors; all fields are static final; all methods abstract
Inheritance	Class extends only one abstract class
Multiple	Class can implement many interfaces

Chapter 9: Exception Handling

9.1 Introduction

An exception is an event that disrupts normal program execution. Java has a robust exception-handling mechanism using try, catch, finally, throw, and throws.

9.2 Common Java Exceptions

Exception	Cause
ArithmeticException	Division by zero
ArrayIndexOutOfBoundsException	Accessing invalid array index
NullPointerException	Using a null object reference
NumberFormatException	Converting invalid string to number
NegativeArraySizeException	Creating array with negative size
FileNotFoundException	Requested file does not exist
ClassCastException	Invalid type casting
StackOverflowError	Infinite recursion

9.3 try-catch Block

```
public class TryCatch {
    public static void main(String[] args) {
        try {
            int a = 10, b = 0;
            int result = a / b;  // causes ArithmeticException
            System.out.println(result);
        } catch (ArithmeticException e) {
            System.out.println("Error: Division by zero!");
        }
        System.out.println("Program continues after exception handling.");
    }
}
```

Output:

```
Error: Division by zero!
Program continues after exception handling.
```

9.4 Multiple catch Blocks

```
public class MultiCatch {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3};
        try {
            System.out.println(arr[5]);  // ArrayIndexOutOfBoundsException
            int x = 10 / 0;             // ArithmeticException
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array index error: " + e.getMessage());
        } catch (ArithmeticException e) {
            System.out.println("Arithmetic error: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("General error: " + e.getMessage());
        }
    }
}
```

```
Output:
Array index error: Index 5 out of bounds for length 3
```

9.5 finally Block

```
public class FinallyDemo {
    public static void main(String[] args) {
        try {
            System.out.println("In try block");
            int x = 5 / 0;
        } catch (ArithmeticException e) {
            System.out.println("In catch block");
        } finally {
            // always executes – used for cleanup
            System.out.println("In finally block – always runs!");
        }
    }
}
```

```
Output:
In try block
In catch block
In finally block – always runs!
```

9.6 throw and throws

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

public class ThrowsDemo {
    static void checkAge(int age) throws InvalidAgeException {
        if (age < 18) {
            throw new InvalidAgeException("Age must be >= 18. Got: " + age);
        }
        System.out.println("Age valid: " + age);
    }

    public static void main(String[] args) {
        try {
            checkAge(15);
        } catch (InvalidAgeException e) {
            System.out.println("Caught: " + e.getMessage());
        }
    }
}
```

```
Output:
Caught: Age must be >= 18. Got: 15
```

Chapter 10: Packages & java.lang

10.1 Introduction to Packages

Packages are a way to organize related classes and interfaces. They prevent naming conflicts and control access.

Package	Contents & Use
java.lang	Fundamental classes – String, Math, Integer, Thread (auto-imported)
java.util	Utility classes – ArrayList, LinkedList, Scanner, Date
java.io	Input/Output operations and file handling
java.net	Network programming – sockets, URLs
java.applet	Creating web applets
java.awt	GUI components – Button, Label, TextField
java.sql	Database connectivity – JDBC
javax.swing	Modern GUI components – JFrame, JButton

10.2 Import Statement

```
// Import all classes from a package
import java.util.*;

// Import a specific class
import java.util.Scanner;
import java.util.ArrayList;

// java.lang is auto-imported – no need to write:
// import java.lang.*;
```

10.3 Wrapper Classes

Wrapper classes convert primitive types to objects. They are in java.lang and provide utility methods.

Primitive	Wrapper Class
int	Integer
double	Double
float	Float
long	Long
char	Character
boolean	Boolean
byte	Byte
short	Short

Integer Class Methods

```
public class WrapperDemo {
    public static void main(String[] args) {
        // String to int
        int a = Integer.parseInt("42");
        System.out.println("Parsed int: " + a);

        // Conversions
        int num = 48;
        System.out.println("Binary:  " + Integer.toBinaryString(num));
        System.out.println("Hex:     " + Integer.toHexString(num));
        System.out.println("Octal:   " + Integer.toOctalString(num));

        // Autoboxing and unboxing
        Integer obj = 100;    // autoboxing: int -> Integer
        int val = obj;        // unboxing: Integer -> int
        System.out.println("Value: " + val);
    }
}
```

Output:

```
Parsed int: 42
Binary:  110000
Hex:     30
Octal:   60
Value: 100
```

10.4 Math Class

Method	Description & Example
Math.sqrt(64)	Square root → 8.0
Math.abs(-15)	Absolute value → 15
Math.pow(2, 10)	Power → 1024.0
Math.ceil(4.3)	Round up → 5.0
Math.floor(4.9)	Round down → 4.0
Math.round(4.5)	Round nearest → 5
Math.max(10, 20)	Maximum → 20
Math.min(10, 20)	Minimum → 10
Math.log(Math.E)	Natural log → 1.0
Math.sin(Math.PI/2)	Sine of 90° → 1.0
Math.random()	Random 0.0 to 1.0

```
public class MathDemo {
    public static void main(String[] args) {
        System.out.println(Math.sqrt(144));     // 12.0
        System.out.println(Math.pow(3, 4));     // 81.0
        System.out.println(Math.abs(-99));       // 99
        System.out.println(Math.max(55, 88));   // 88
        // Trigonometry (angle in radians)
        double angle = Math.toRadians(90);
        System.out.println(Math.sin(angle));    // 1.0
    }
}
```

```
Output:
12.0
81.0
99
88
1.0
```

10.5 Character Class

Method	Description
Character.isDigit(c)	Returns true if c is a digit (0-9)
Character.isLetter(c)	Returns true if c is a letter
Character.isLetterOrDigit(c)	True if letter or digit
Character.isLowerCase(c)	True if lowercase letter
Character.isUpperCase(c)	True if uppercase letter
Character.isWhitespace(c)	True if space, tab, newline
Character.toLowerCase(c)	Converts to lowercase
Character.toUpperCase(c)	Converts to uppercase

```
public class CharDemo {
    public static void main(String[] args) {
        char ch = 'A';
        System.out.println(Character.isLetter(ch));      // true
        System.out.println(Character.isUpperCase(ch));   // true
        System.out.println(Character.toLowerCase(ch));   // a
        char num = '7';
        System.out.println(Character.isDigit(num));      // true
    }
}
```

```
Output:
true
true
a
true
```

Chapter 11: String & StringBuffer

11.1 Introduction to String

A String in Java is an immutable sequence of characters. Once created, a String object cannot be changed. String is a class in java.lang.

```
// Two ways to create a String
String s1 = "Hello";                  // String literal
String s2 = new String("World");       // Using new keyword
System.out.println(s1 + " " + s2);    // Hello World
```

11.2 String Methods

Method	Description & Example
length()	Returns length: "Hello".length() → 5
charAt(i)	Character at index: "Java".charAt(0) → 'J'
indexOf(str)	First occurrence: "Hello".indexOf('l') → 2
substring(start,end)	Substring: "Hello World".substring(6,11) → "World"
toUpperCase()	Uppercase: "hello".toUpperCase() → "HELLO"
toLowerCase()	Lowercase: "JAVA".toLowerCase() → "java"
trim()	Remove whitespace: " hi ".trim() → "hi"
replace(old,new)	Replace: "abcd".replace('b','x') → "axcd"
equals(s)	Exact match: s1.equals(s2) → true/false
equalsIgnoreCase(s)	Case-insensitive compare
startsWith(prefix)	"Hello".startsWith("He") → true
endsWith(suffix)	"Hello".endsWith("lo") → true
contains(s)	"Hello World".contains("World") → true
split(regex)	"a,b,c".split(",") → {"a","b","c"}
compareTo(s)	Lexicographic compare: returns 0 if equal
isEmpty()	Returns true if length == 0
toCharArray()	Converts string to char array

String Methods Program

```
public class StringMethods {
    public static void main(String[] args) {
        String str = "  Hello Java World  ";
        System.out.println("Original:     '" + str + "'");
        System.out.println("Trimmed:      '" + str.trim() + "'");
        System.out.println("Length:        " + str.trim().length());
        System.out.println("Uppercase:     " + str.trim().toUpperCase());
        System.out.println("Lowercase:     " + str.trim().toLowerCase());
        System.out.println("Substring:     " + str.trim().substring(6, 10));
        System.out.println("Replace:       " + str.trim().replace('l', 'L'));
        System.out.println("Contains Java: " + str.contains("Java"));
        System.out.println("Starts with H: " + str.trim().startsWith("H"));
        System.out.println("Index of 'o':  " + str.indexOf('o'));
        System.out.println("charAt(8):     " + str.trim().charAt(8));
    }
}
```

```
Output:
Original:     '  Hello Java World  '
Trimmed:      'Hello Java World'
Length:        16
Uppercase:     HELLO JAVA WORLD
Lowercase:     hello java world
Substring:     Java
Replace:       HeLLo Java WorLd
Contains Java: true
Starts with H: true
Index of 'o':  4
charAt(8):     J
```

11.3 StringBuffer – Mutable Strings

StringBuffer is mutable – you can change its content without creating new objects. Better performance for frequent modifications.

String vs StringBuffer	Details
String	Immutable – once created, cannot be changed
StringBuffer	Mutable – content can be modified in place
Performance	StringBuffer is faster for repeated modifications
Thread Safety	StringBuffer is synchronized (thread-safe)

StringBuffer Methods

```
public class StringBufferDemo {
    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer("Hello");
        System.out.println("Initial:   " + sb);

        sb.append(" Java");         // Add to end
        System.out.println("Append:    " + sb);

        sb.insert(5, ",");          // Insert at position 5
        System.out.println("Insert:    " + sb);

        sb.replace(7, 11, "World"); // Replace from 7 to 11
        System.out.println("Replace:   " + sb);

        sb.delete(5, 7);            // Delete from 5 to 7
        System.out.println("Delete:    " + sb);

        sb.reverse();               // Reverse string
        System.out.println("Reverse:   " + sb);

        System.out.println("Length:    " + sb.length());
    }
}
```

```
Output:
Initial:   Hello
```

```
Append:    Hello Java
Insert:    Hello, Java
Replace:   Hello, World
Delete:    HelloWorld
Reverse:   dlroWolleH
Length:    10
```

Chapter 12: File I/O – Input and Output

12.1 Introduction

Java provides the java.io package for reading and writing data to files. Main classes: FileWriter, FileReader, BufferedWriter, BufferedReader, and the File class.

Class	Purpose
FileWriter	Write characters to a file
FileReader	Read characters from a file
BufferedWriter	Buffered writing – faster for large data
BufferedReader	Buffered reading – faster, supports readLine()
File	Represents file/directory path; create/delete/inspect files
PrintWriter	Formatted text output to file

12.2 Writing to a File

```
import java.io.*;

public class WriteFile {
    public static void main(String[] args) {
        try {
            FileWriter fw = new FileWriter("output.txt");
            BufferedWriter bw = new BufferedWriter(fw);

            bw.write("Hello, File World!");
            bw.newLine();
            bw.write("Second line of text.");
            bw.newLine();
            bw.write("Third line of text.");

            bw.close();
            fw.close();
            System.out.println("File written successfully!");
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

```
Output:
File written successfully!
```

12.3 Reading from a File

```
import java.io.*;

public class ReadFile {
    public static void main(String[] args) {
        try {
            FileReader fr = new FileReader("output.txt");
            BufferedReader br = new BufferedReader(fr);
            String line;
            System.out.println("File contents:");
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }

            br.close();
            fr.close();
        } catch (FileNotFoundException e) {
            System.out.println("File not found!");
        } catch (IOException e) {
            System.out.println("Read error: " + e.getMessage());
        }
    }
}
```

```
Output:
File contents:
Hello, File World!
Second line of text.
Third line of text.
```

12.4 File Class Methods

```
import java.io.*;

public class FileDemo {
    public static void main(String[] args) {
        File f = new File("output.txt");
        System.out.println("Name:    " + f.getName());
        System.out.println("Path:    " + f.getPath());
        System.out.println("Exists:  " + f.exists());
        System.out.println("Readable:" + f.canRead());
        System.out.println("Writable:" + f.canWrite());
        System.out.println("Size:    " + f.length() + " bytes");
    }
}
```

```
Output:
Name:    output.txt
Path:    output.txt
Exists:  true
Readable:true
Writable:true
Size:    62 bytes
```

Chapter 13: Collections Framework

13.1 Introduction

The Java Collections Framework provides ready-to-use data structures. It is in java.util. Main interfaces: List, Set, Map, Queue.

Collection	Description & Use
ArrayList	Dynamic array; allows duplicates; ordered; fast random access
LinkedList	Doubly-linked list; fast insert/delete at ends
HashSet	No duplicates; no order; uses hashing for fast lookup
TreeSet	No duplicates; sorted ascending; uses tree structure
HashMap	Key-value pairs; no order; fast lookup by key
LinkedHashMap	Key-value pairs; maintains insertion order
Stack	Last In First Out (LIFO) structure
Vector	Synchronized ArrayList (thread-safe)

13.2 ArrayList

```
import java.util.*;

public class ArrayListDemo {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<>();

        // Add elements
        list.add("Apple");
        list.add("Banana");
        list.add("Cherry");
        list.add(1, "Avocado");  // insert at index 1

        System.out.println("List: " + list);
        System.out.println("Size: " + list.size());
        System.out.println("Get(2): " + list.get(2));

        list.remove("Banana");  // remove by value
        list.remove(0);         // remove by index
        System.out.println("After remove: " + list);

        // Iterate
        for (String fruit : list) {
            System.out.println("  " + fruit);
        }
    }
}
```

Output:

```
List: [Apple, Avocado, Banana, Cherry]
Size: 4
Get(2): Banana
```

```
After remove: [Cherry]
  Cherry
```

13.3 LinkedList

```
import java.util.*;

public class LinkedListDemo {
    public static void main(String[] args) {
        LinkedList<String> ll = new LinkedList<>();
        ll.add("A");
        ll.add("B");
        ll.add("C");
        ll.addFirst("Start");
        ll.addLast("End");
        System.out.println("List:      " + ll);
        System.out.println("First:     " + ll.getFirst());
        System.out.println("Last:      " + ll.getLast());
        ll.removeFirst();
        ll.removeLast();
        System.out.println("After trim:" + ll);
    }
}
```

Output:

```
List:      [Start, A, B, C, End]
First:     Start
Last:      End
After trim:[A, B, C]
```

13.4 HashSet

```
import java.util.*;

public class HashSetDemo {
    public static void main(String[] args) {
        HashSet<String> hs = new HashSet<>();
        hs.add("Mango");
        hs.add("Apple");
        hs.add("Banana");
        hs.add("Mango");  // duplicate – not added
        System.out.println("Set: " + hs);          // no duplicates
        System.out.println("Size: " + hs.size()); // 3
        System.out.println("Contains Apple: " + hs.contains("Apple"));
        hs.remove("Banana");
        System.out.println("After remove: " + hs);
    }
}
```

Output:

```
Set: [Apple, Mango, Banana]  (order may vary)
Size: 3
Contains Apple: true
After remove: [Apple, Mango]
```

13.5 TreeSet – Sorted Set

```
import java.util.*;

public class TreeSetDemo {
    public static void main(String[] args) {
        TreeSet<Integer> ts = new TreeSet<>();
        ts.add(50); ts.add(10); ts.add(30); ts.add(20); ts.add(40);
        System.out.println("Sorted: " + ts);       // [10, 20, 30, 40, 50]
        System.out.println("First: " + ts.first()); // 10
        System.out.println("Last: " + ts.last());   // 50
    }
}
```

Output:

```
Sorted: [10, 20, 30, 40, 50]
First: 10
Last: 50
```

13.6 HashMap – Key-Value Pairs

```
import java.util.*;

public class HashMapDemo {
    public static void main(String[] args) {
        HashMap<String, int[]> scores = new HashMap<>();
        scores.put("Alice", new int[]{95});
        scores.put("Bob", new int[]{87});
        scores.put("Carol", new int[]{92});

        // Access
        System.out.println("Alice: " + scores.get("Alice")[0]);

        // Iterate
        for (Map.Entry<String, int[]> e : scores.entrySet()) {
            System.out.println(e.getKey() + " -> " + e.getValue()[0]);
        }
        System.out.println("Contains Bob: " + scores.containsKey("Bob"));
        scores.remove("Bob");
        System.out.println("After remove size: " + scores.size());
    }
}
```

Output:

```
Alice: 95
Alice -> 95
Bob -> 87
Carol -> 92
Contains Bob: true
After remove size: 2
```

13.7 Iterator

```
import java.util.*;

public class IteratorDemo {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<>();
        list.add("Red"); list.add("Green"); list.add("Blue");

        Iterator<String> it = list.iterator();
        while (it.hasNext()) {
            String item = it.next();
            System.out.println(item);
            if (item.equals("Green")) it.remove(); // safe removal
        }
        System.out.println("After removal: " + list);
    }
}
```

Output:

```
Red
Green
Blue
After removal: [Red, Blue]
```

Chapter 14: Multithreading

14.1 Introduction

A thread is the smallest unit of execution. Multithreading allows multiple threads to run concurrently, making better use of CPU resources.

Thread State	Description
New	Thread created but not started
Runnable	Thread is ready to run, waiting for CPU
Running	Thread is currently executing
Waiting/Sleeping	Thread is paused temporarily
Terminated/Dead	Thread has finished execution

14.2 Creating Threads – Method 1: Extending Thread

```
class MyThread extends Thread {
    String name;
    MyThread(String name) { this.name = name; }

    public void run() {
        for (int i = 1; i <= 3; i++) {
            System.out.println(name + " - Count: " + i);
            try { Thread.sleep(500); } catch (Exception e) {}
        }
    }
}

public class ThreadDemo1 {
    public static void main(String[] args) {
        MyThread t1 = new MyThread("Thread-A");
        MyThread t2 = new MyThread("Thread-B");
        t1.start();
        t2.start();
    }
}
```

```
Output (order may vary):
Thread-A - Count: 1
Thread-B - Count: 1
Thread-A - Count: 2
Thread-B - Count: 2
Thread-A - Count: 3
Thread-B - Count: 3
```

14.3 Creating Threads – Method 2: Implementing Runnable

```
class Task implements Runnable {
    String taskName;
    Task(String name) { this.taskName = name; }

    public void run() {
        System.out.println(taskName + " started");
        try { Thread.sleep(1000); } catch (Exception e) {}
        System.out.println(taskName + " finished");
    }
}

public class RunnableDemo {
    public static void main(String[] args) {
        Thread t1 = new Thread(new Task("Download"));
        Thread t2 = new Thread(new Task("Upload"));
        t1.start();
        t2.start();
    }
}
```

```
Output:
Download started
Upload started
Download finished
Upload finished
```

14.4 Thread Methods

Method	Description
start()	Begin thread execution (calls run())
run()	Contains thread logic; do not call directly
sleep(ms)	Pause thread for specified milliseconds
join()	Wait for this thread to finish before continuing
isAlive()	Returns true if thread is still running
getName()	Returns thread name
setPriority(n)	Set priority 1 (MIN) to 10 (MAX)
interrupt()	Interrupt a sleeping/waiting thread

join() and isAlive() Example

```
public class JoinDemo {
    public static void main(String[] args) throws Exception {
        Thread t1 = new Thread(() -> {
            System.out.println("Worker thread running...");
            try { Thread.sleep(2000); } catch (Exception e) {}
            System.out.println("Worker thread done.");
        });

        t1.start();
        System.out.println("Is alive: " + t1.isAlive()); // true
        t1.join();  // main thread waits for t1 to finish
        System.out.println("Is alive: " + t1.isAlive()); // false
        System.out.println("Main continues after worker.");
    }
}
```

```
Output:
Worker thread running...
Is alive: true
Worker thread done.
Is alive: false
Main continues after worker.
```

14.5 Synchronization

When multiple threads access a shared resource simultaneously, synchronization ensures only one thread accesses it at a time, preventing data corruption.

```
class BankAccount {
    private int balance = 1000;

    synchronized void withdraw(String who, int amount) {
        if (balance >= amount) {
            System.out.println(who + " withdrawing " + amount);
            balance -= amount;
            System.out.println(who + " done. Balance: " + balance);
        } else {
            System.out.println(who + ": Insufficient funds!");
        }
    }
}

public class SyncDemo {
    public static void main(String[] args) {
        BankAccount acc = new BankAccount();
        Thread t1 = new Thread(() -> acc.withdraw("Alice", 700));
        Thread t2 = new Thread(() -> acc.withdraw("Bob", 700));
        t1.start();
        t2.start();
    }
}
```

Output:

```
Alice withdrawing 700
Alice done. Balance: 300
Bob: Insufficient funds!
```

Chapter 15: Applets

15.1 Introduction

An applet is a Java program that runs inside a web browser or applet viewer. Applets extend the java.applet.Applet class. They have no main() method – they use a lifecycle with init(), start(), stop(), and destroy().

15.2 Applet Life Cycle

Method	Description
init()	Called once when applet loads – initialize variables here
start()	Called each time applet becomes visible/focus
paint(Graphics g)	Called when applet needs to be drawn/redrawn
stop()	Called when applet is hidden or browser tab changes
destroy()	Called when applet is unloaded from memory

15.3 Simple Applet Example

```
import java.applet.*;
import java.awt.*;

// HTML tag to run: <applet code=HelloApplet width=300 height=200></applet>
public class HelloApplet extends Applet {
    public void paint(Graphics g) {
        g.setColor(Color.blue);
        g.setFont(new Font("Arial", Font.BOLD, 24));
        g.drawString("Hello from Java Applet!", 50, 100);
    }
}
```

15.4 Applet with Parameters

```
import java.applet.*;
import java.awt.*;

// HTML: <applet code=ParamApplet width=300 height=200>
//         <param name='msg' value='Welcome to Java!'>
//       </applet>
public class ParamApplet extends Applet {
    String message;
    public void init() {
        message = getParameter("msg");  // get HTML param
        if (message == null) message = "Default Message";
    }
    public void paint(Graphics g) {
        g.drawString(message, 50, 100);
    }
}
```

15.5 Applet vs Application

Applet	Application
Runs in browser / applet viewer	Runs standalone on OS
No main() method	Has public static void main()
Loaded from server	Installed on local machine
Restricted security	Full system access
Extends Applet class	May or may not extend any class

Chapter 16: AWT – Abstract Window Toolkit (Graphics)

16.1 Introduction to AWT

The Abstract Window Toolkit (AWT) is Java's original GUI framework. It provides classes for creating windows, drawing shapes, and handling events. AWT components are heavyweight – they rely on the OS's native widgets.

Package	Purpose
java.awt.*	Core AWT classes – Graphics, Color, Font, Component
java.awt.event.*	Event handling – ActionListener, MouseListener

16.2 Colors in AWT

Predefined Color	Custom Color
Color.red, Color.blue, Color.green	new Color(255, 0, 0) – RGB values 0-255
Color.yellow, Color.pink, Color.cyan	new Color(128, 64, 192) – custom purple
Color.white, Color.black, Color.gray	new Color(0, 128, 255) – custom blue

16.3 Drawing Basic Shapes

```
import java.applet.*;
import java.awt.*;

public class DrawShapes extends Applet {
    public void paint(Graphics g) {
        // Draw line
        g.setColor(Color.black);
        g.drawLine(50, 50, 200, 50);

        // Draw rectangle (outline)
        g.setColor(Color.blue);
        g.drawRect(50, 80, 150, 80);

        // Filled rectangle
        g.setColor(Color.red);
        g.fillRect(50, 180, 150, 80);

        // Draw oval/ellipse (outline)
        g.setColor(Color.green);
        g.drawOval(250, 80, 150, 100);

        // Filled oval
        g.setColor(Color.yellow);
        g.fillOval(250, 200, 150, 100);

        // Rounded rectangle
        g.setColor(Color.magenta);
        g.fillRoundRect(450, 80, 150, 100, 30, 30);
        // Draw arc
        g.setColor(Color.cyan);
        g.fillArc(450, 200, 150, 100, 0, 180);
    }
}
```

16.4 Drawing Polygons

```
public void paint(Graphics g) {
    g.setColor(Color.red);
    int[] x = {200, 300, 100, 300, 100, 200};
    int[] y = {50, 300, 100, 100, 300, 50};
    g.fillPolygon(x, y, 6);  // 6 points = star shape
}
```

16.5 Working with Fonts

```
import java.applet.*;
import java.awt.*;

public class FontDemo extends Applet {
    public void paint(Graphics g) {
        Font f1 = new Font("Arial", Font.BOLD, 20);
        g.setFont(f1);
        g.setColor(Color.blue);
        g.drawString("Bold Arial 20pt", 50, 60);

        Font f2 = new Font("Times New Roman", Font.ITALIC, 24);
        g.setFont(f2);
        g.setColor(new Color(150, 0, 150));
        g.drawString("Italic Times 24pt", 50, 100);

        Font f3 = new Font("Courier New", Font.BOLD + Font.ITALIC, 18);
        g.setFont(f3);
        g.setColor(Color.red);
        g.drawString("Bold Italic Courier 18pt", 50, 140);
    }
}
```

Note: Font styles: Font.PLAIN, Font.BOLD, Font.ITALIC, Font.BOLD+Font.ITALIC

16.6 Graphics Methods Summary

Method	Description
drawLine(x1,y1,x2,y2)	Draw a straight line
drawRect(x,y,w,h)	Draw rectangle outline
fillRect(x,y,w,h)	Draw filled rectangle
drawOval(x,y,w,h)	Draw oval/circle outline
fillOval(x,y,w,h)	Draw filled oval/circle
drawRoundRect(x,y,w,h,arcW,arcH)	Rounded rectangle outline
fillRoundRect(x,y,w,h,arcW,arcH)	Filled rounded rectangle
drawArc(x,y,w,h,start,angle)	Draw arc
fillArc(x,y,w,h,start,angle)	Filled arc (pie segment)

drawPolygon(xArr,yArr,n)	Draw polygon outline
fillPolygon(xArr,yArr,n)	Filled polygon
drawString(str,x,y)	Draw text string
setColor(Color c)	Set drawing color
setFont(Font f)	Set text font

Chapter 17: Event Handling

17.1 Introduction to Event-Driven Programming

Java programs respond to user actions (events) like mouse clicks, key presses, button clicks. The Event Delegation Model routes events from source to listener.

Event	Listener Interface
Button click / menu selection	ActionListener – actionPerformed()
Window close/open	WindowListener – windowClosing()
Mouse press/click/move	MouseListener – mouseClicked(), mousePressed()
Mouse drag/move	MouseMotionListener – mouseMoved(), mouseDragged()
Key press/release/type	KeyListener – keyPressed(), keyReleased()
Component gain/lose focus	FocusListener – focusGained(), focusLost()
Item selection change	ItemListener – itemStateChanged()

17.2 Handling Mouse Events

```java
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class MouseDemo extends Applet implements MouseListener {
    String msg = "";
    int x = 0, y = 0;

    public void init() {
        addMouseListener(this);
    }

    public void mouseClicked(MouseEvent me) {
        x = me.getX(); y = me.getY();
        msg = "Clicked at (" + x + ", " + y + ")";
        repaint();
    }
    public void mousePressed(MouseEvent me)  { msg = "Mouse Pressed";
repaint(); }
    public void mouseReleased(MouseEvent me) { msg = "Mouse Released";
repaint(); }
    public void mouseEntered(MouseEvent me)  { msg = "Mouse Entered";
repaint(); }
    public void mouseExited(MouseEvent me)   { msg = "Mouse Exited";
repaint(); }

    public void paint(Graphics g) {
        g.drawString(msg, 50, 50);
    }
}
```

17.3 Mouse Motion Listener

```
public class MouseMotionDemo extends Applet implements MouseMotionListener {
    String msg = "";

    public void init() {
        addMouseMotionListener(this);
    }

    public void mouseMoved(MouseEvent me) {
        msg = "Mouse at (" + me.getX() + ", " + me.getY() + ")";
        repaint();
    }
    public void mouseDragged(MouseEvent me) {
        msg = "Dragging at (" + me.getX() + ", " + me.getY() + ")";
        repaint();
    }
    public void paint(Graphics g) { g.drawString(msg, 10, 50); }
}
```

17.4 Keyboard Event Handling

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class KeyDemo extends Applet implements KeyListener {
    String msg = "";

    public void init() {
        addKeyListener(this);
    }

    public void keyTyped(KeyEvent ke)    { msg = "Key Typed: " +
ke.getKeyChar(); repaint(); }
    public void keyPressed(KeyEvent ke)  { msg = "Key Pressed: code=" +
ke.getKeyCode(); repaint(); }
    public void keyReleased(KeyEvent ke) { msg = "Key Released"; repaint(); }

    public void paint(Graphics g) {
        g.drawString(msg, 50, 50);
    }
}
```

17.5 Adapter Classes

Listener interfaces with multiple methods can be inconvenient. Adapter classes provide empty implementations so you only override the methods you need.

Listener Interface	Adapter Class
MouseListener (5 methods)	MouseAdapter
MouseMotionListener (2 methods)	MouseMotionAdapter
KeyListener (3 methods)	KeyAdapter
WindowListener (7 methods)	WindowAdapter
ComponentListener (4 methods)	ComponentAdapter

FocusListener (2 methods)	FocusAdapter
ActionListener (1 method)	None – only 1 method

```
// Using adapter class – only override what you need
class MyMouseHandler extends MouseAdapter {
    public void mouseClicked(MouseEvent e) {
        System.out.println("Clicked at: " + e.getX() + ", " + e.getY());
        // Only implement mouseClicked – don't need to implement the other 4
    }
}
```

Chapter 18: AWT Controls

18.1 Introduction to UI Controls

AWT provides a set of UI components that users interact with. All components extend java.awt.Component.

Control	Purpose
Label	Display static text – non-interactive
Button	Clickable button that generates ActionEvent
TextField	Single-line text input
TextArea	Multi-line text input
Checkbox	Toggle on/off; can be in groups
CheckboxGroup	Radio-button behavior (only one selected)
Choice	Drop-down list (combo box)
List	Scrollable list of items
Scrollbar	Horizontal or vertical scrollbar
Panel	Container for grouping components

18.2 Label

```
import java.applet.*; import java.awt.*;

public class LabelDemo extends Applet {
    public void init() {
        Label l1 = new Label("Name:");
        Label l2 = new Label("Welcome to Java!");
        l2.setForeground(Color.blue);
        add(l1);
        add(l2);
    }
}
```

18.3 Button

```
import java.applet.*; import java.awt.*; import java.awt.event.*;

public class ButtonDemo extends Applet implements ActionListener {
    Label result;
    public void init() {
        Button b1 = new Button("Red");
        Button b2 = new Button("Blue");
        result = new Label("Click a button!");
        b1.addActionListener(this);
        b2.addActionListener(this);
        add(b1); add(b2); add(result);
    }
    public void actionPerformed(ActionEvent ae) {
        String cmd = ae.getActionCommand();
        if (cmd.equals("Red"))  setBackground(Color.red);
        else                    setBackground(Color.blue);
        result.setText("You clicked: " + cmd);
    }
}
```

18.4 TextField – Calculator

```
import java.applet.*; import java.awt.*; import java.awt.event.*;

public class CalcDemo extends Applet implements ActionListener {
    TextField t1, t2;
    Label result;
    public void init() {
        add(new Label("Number 1:")); t1 = new TextField(10); add(t1);
        add(new Label("Number 2:")); t2 = new TextField(10); add(t2);
        Button add = new Button("Add"); add.addActionListener(this); add(add);
        Button sub = new Button("Sub"); sub.addActionListener(this); add(sub);
        result = new Label("Result: "); add(result);
    }
    public void actionPerformed(ActionEvent ae) {
        int a = Integer.parseInt(t1.getText());
        int b = Integer.parseInt(t2.getText());
        if (ae.getActionCommand().equals("Add"))
            result.setText("Result: " + (a + b));
        else
            result.setText("Result: " + (a - b));
    }
}
```

18.5 Checkbox and CheckboxGroup (Radio Buttons)

```
import java.applet.*; import java.awt.*; import java.awt.event.*;

public class CheckDemo extends Applet implements ItemListener {
    Checkbox cb1, cb2, cb3;
    Label msg;
    public void init() {
        // Regular checkboxes
        cb1 = new Checkbox("Java");
        cb2 = new Checkbox("Python");
        cb3 = new Checkbox("C++");
        msg = new Label("Select language(s)");
        cb1.addItemListener(this); cb2.addItemListener(this);
cb3.addItemListener(this);
        add(new Label("Languages:")); add(cb1); add(cb2); add(cb3); add(msg);

        // Radio buttons (CheckboxGroup)
        CheckboxGroup grp = new CheckboxGroup();
        Checkbox r1 = new Checkbox("Male", grp, true);
        Checkbox r2 = new Checkbox("Female", grp, false);
        add(new Label("Gender:")); add(r1); add(r2);
    }
    public void itemStateChanged(ItemEvent ie) {
        String sel = "";
        if (cb1.getState()) sel += "Java ";
        if (cb2.getState()) sel += "Python ";
        if (cb3.getState()) sel += "C++ ";
        msg.setText("Selected: " + sel);
    }
}
```

18.6 Choice (Drop-Down) and List

```
import java.applet.*; import java.awt.*; import java.awt.event.*;

public class ChoiceDemo extends Applet implements ItemListener {
    Choice country;
    Label selected;
    public void init() {
        country = new Choice();
        country.add("India");
        country.add("USA");
        country.add("UK");
        country.add("Japan");
        selected = new Label("Select a country");
        country.addItemListener(this);
        add(new Label("Country:")); add(country); add(selected);
    }
    public void itemStateChanged(ItemEvent ie) {
        selected.setText("Selected: " + country.getSelectedItem());
    }
}
```

18.7 Component Methods Reference

Method	Description
setSize(w, h)	Set component dimensions
setFont(Font f)	Set text font
setEnabled(bool)	Enable or disable component
setVisible(bool)	Show or hide component
setForeground(Color)	Set text/foreground color
setBackground(Color)	Set background color
setBounds(x,y,w,h)	Set position and size
getText()	Get text (TextField/TextArea)
setText(str)	Set text (Label/TextField)

Chapter 19: Layout Managers

19.1 Introduction

Layout managers control how components are arranged in a container. Java provides several layout manager classes.

Layout Manager	Description
FlowLayout	Components arranged left-to-right, wrapping to next line
GridLayout	Components in a grid of rows x columns
BorderLayout	North, South, East, West, Center regions
CardLayout	Shows one component at a time (like cards)
GridBagLayout	Most flexible; positions using constraints
null (none)	Absolute positioning using setBounds()

19.2 FlowLayout

The default layout for Applet and Panel. Components flow left-to-right, wrap to next row.

```
import java.applet.*; import java.awt.*;

public class FlowDemo extends Applet {
    public void init() {
        setLayout(new FlowLayout(FlowLayout.CENTER, 10, 10));
        // FlowLayout(alignment, hGap, vGap)
        add(new Button("Button 1"));
        add(new Button("Button 2"));
        add(new Button("Button 3"));
        add(new Button("Button 4"));
        add(new Button("Button 5"));
    }
}
```

19.3 GridLayout

Arranges components in a grid. Each cell is the same size.

```
import java.applet.*; import java.awt.*;

public class GridDemo extends Applet {
    public void init() {
        setLayout(new GridLayout(3, 3, 5, 5)); // 3 rows, 3 cols, gaps
        for (int i = 1; i <= 9; i++) {
            add(new Button("Cell " + i));
        }
    }
}
```

19.4 BorderLayout

Divides container into 5 regions: North, South, East, West, Center.

```
import java.awt.*; import java.applet.*;
```

```
public class BorderDemo extends Applet {
    public void init() {
        setLayout(new BorderLayout(5, 5));
        add("North",  new Button("NORTH – Header"));
        add("South",  new Button("SOUTH – Footer"));
        add("East",   new Button("EAST – Right"));
        add("West",   new Button("WEST – Left"));
        add("Center", new TextArea("CENTER – Main Content Area"));
    }
}
```

Note: The Center component expands to fill remaining space. You can use any component in each region.

19.5 CardLayout

Shows one 'card' (panel) at a time. Useful for wizards, tabs, or sliding views.

```
import java.applet.*; import java.awt.*; import java.awt.event.*;

public class CardDemo extends Applet implements ActionListener {
    CardLayout cards;
    Panel cardPanel;
    Button next, prev;

    public void init() {
        cards = new CardLayout();
        cardPanel = new Panel();
        cardPanel.setLayout(cards);

        Panel p1 = new Panel(); p1.add(new Label("Card 1 – Introduction"));
        Panel p2 = new Panel(); p2.add(new Label("Card 2 – Details"));
        Panel p3 = new Panel(); p3.add(new Label("Card 3 – Summary"));
        cardPanel.add(p1, "Card1");
        cardPanel.add(p2, "Card2");
        cardPanel.add(p3, "Card3");

        next = new Button("Next"); prev = new Button("Prev");
        next.addActionListener(this); prev.addActionListener(this);

        setLayout(new BorderLayout());
        add("Center", cardPanel);
        Panel btnPanel = new Panel();
        btnPanel.add(prev); btnPanel.add(next);
        add("South", btnPanel);
    }
    public void actionPerformed(ActionEvent ae) {
        if (ae.getSource() == next) cards.next(cardPanel);
        else cards.previous(cardPanel);
    }
}
```

Chapter 20: AWT Frames and Panels

20.1 Frame

A Frame is a top-level window with a title bar, borders, and optionally menus. It is used to create standalone GUI applications.

```
import java.awt.*;
import java.awt.event.*;

public class MyFrame extends Frame implements WindowListener {
    MyFrame() {
        super("My Java Frame");
        setSize(400, 300);
        setVisible(true);
        addWindowListener(this);
    }

    public void paint(Graphics g) {
        g.setColor(Color.blue);
        g.setFont(new Font("Arial", Font.BOLD, 24));
        g.drawString("Hello from AWT Frame!", 80, 150);
    }

    // WindowListener methods
    public void windowClosing(WindowEvent e) {
        dispose();
        System.exit(0);
    }
    public void windowOpened(WindowEvent e) {}
    public void windowClosed(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}
    public void windowActivated(WindowEvent e) {}
    public void windowDeactivated(WindowEvent e) {}

    public static void main(String[] args) {
        new MyFrame();
    }
}
```

20.2 Using WindowAdapter (Simpler)

```
import java.awt.*;
import java.awt.event.*;

class AppFrame extends Frame {
    AppFrame() {
        super("Frame with Adapter");
        setSize(400, 300);
        setBackground(Color.lightGray);

        // WindowAdapter – only override what you need
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                dispose();
                System.exit(0);
            }
        });
        setVisible(true);
    }
    public void paint(Graphics g) {
        g.setFont(new Font("Arial", Font.PLAIN, 18));
        g.drawString("Click X to close this frame", 80, 150);
    }
}

public class FrameAdapterDemo {
    public static void main(String[] args) { new AppFrame(); }
}
```

20.3 Scrollbar

```
import java.applet.*; import java.awt.*; import java.awt.event.*;

public class ScrollDemo extends Applet implements AdjustmentListener {
    Scrollbar sb;
    Label val;
    public void init() {
        sb = new Scrollbar(Scrollbar.HORIZONTAL, 0, 10, 0, 255);
        val = new Label("Value: 0");
        sb.addAdjustmentListener(this);
        add(new Label("Brightness:")); add(sb); add(val);
    }
    public void adjustmentValueChanged(AdjustmentEvent ae) {
        int v = sb.getValue();
        val.setText("Value: " + v);
        setBackground(new Color(v, v, v));
    }
}
```

Chapter 21: AWT Menus, PopupMenus and Dialogs

21.1 MenuBar, Menu and MenuItem

Menus are attached to Frames using MenuBar. A MenuBar contains Menus, and each Menu contains MenuItems.

```
import java.awt.*;
import java.awt.event.*;

public class MenuDemo extends Frame implements ActionListener {
    Label status;

    MenuDemo() {
        super("Menu Demo");
        MenuBar mb = new MenuBar();

        // File menu
        Menu file = new Menu("File");
        MenuItem newItem  = new MenuItem("New");
        MenuItem openItem = new MenuItem("Open");
        MenuItem saveItem = new MenuItem("Save");
        MenuItem exitItem = new MenuItem("Exit");
        file.add(newItem); file.add(openItem); file.addSeparator();
        file.add(saveItem); file.add(exitItem);

        // Edit menu
        Menu edit = new Menu("Edit");
        edit.add(new MenuItem("Cut"));
        edit.add(new MenuItem("Copy"));
        edit.add(new MenuItem("Paste"));

        mb.add(file); mb.add(edit);
        setMenuBar(mb);

        status = new Label("Ready");
        add(status);
        exitItem.addActionListener(this);
        setLayout(new FlowLayout());
        setSize(400, 300);
        setVisible(true);
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) { System.exit(0); }
        });
    }

    public void actionPerformed(ActionEvent ae) {
        if (ae.getActionCommand().equals("Exit")) System.exit(0);
    }

    public static void main(String[] args) { new MenuDemo(); }
}
```

21.2 PopupMenu

A popup menu appears when the user right-clicks (mouse pressed). It is context-sensitive.

```
import java.awt.*;
import java.awt.event.*;
```

```
public class PopupDemo extends Frame implements ActionListener {
    PopupMenu popup;

    PopupDemo() {
        super("Right-Click for Menu");
        popup = new PopupMenu();
        popup.add(new MenuItem("Copy"));
        popup.add(new MenuItem("Paste"));
        popup.addSeparator();
        MenuItem quit = new MenuItem("Quit");
        quit.addActionListener(this);
        popup.add(quit);
        add(popup);

        addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent e) {
                if (e.isPopupTrigger()) {
                    popup.show(e.getComponent(), e.getX(), e.getY());
                }
            }
        });
        setSize(300, 300);
        setVisible(true);
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) { System.exit(0); }
        });
    }
    public void actionPerformed(ActionEvent ae) { System.exit(0); }
    public static void main(String[] args) { new PopupDemo(); }
}
```

21.3 Dialog Box

A Dialog is a popup window used to display messages or get input. A modal dialog blocks input to the parent window until closed.

```
import java.awt.*;
import java.awt.event.*;

class MyDialog extends Dialog {
    MyDialog(Frame parent) {
        super(parent, "Alert!", true);  // modal = true
        Label msg = new Label("This is a dialog message!");
        Button ok = new Button("OK");
        ok.addActionListener(e -> dispose());
        setLayout(new FlowLayout());
        add(msg); add(ok);
        setSize(250, 150);
        setVisible(true);
    }
}

public class DialogDemo extends Frame {
    DialogDemo() {
        super("Dialog Demo");
        Button btn = new Button("Open Dialog");
        btn.addActionListener(e -> new MyDialog(this));
        add(btn);
        setLayout(new FlowLayout());
        setSize(300, 200);
        setVisible(true);
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) { System.exit(0); }
        });
    }
    public static void main(String[] args) { new DialogDemo(); }
}
```

21.4 FileDialog

FileDialog provides a standard OS file chooser dialog for opening or saving files.

```
// Open File Dialog
FileDialog fdOpen = new FileDialog(parent, "Open File", FileDialog.LOAD);
fdOpen.setVisible(true);
String dir = fdOpen.getDirectory();
String file = fdOpen.getFile();
if (file != null) System.out.println("Selected: " + dir + file);

// Save File Dialog
FileDialog fdSave = new FileDialog(parent, "Save File", FileDialog.SAVE);
fdSave.setVisible(true);
if (fdSave.getFile() != null) {
    String path = fdSave.getDirectory() + fdSave.getFile();
    System.out.println("Saving to: " + path);
}
```

FileDialog Method	Description
getFile()	Returns selected filename as String (null if cancelled)
getDirectory()	Returns the directory path
setFile(String)	Sets default filename in dialog
setDirectory(String)	Sets starting directory

Chapter 22: JFC – Java Foundation Classes & Swing

22.1 Introduction to Swing

Swing (javax.swing) is Java's advanced GUI toolkit introduced in JDK 1.2. Unlike AWT, Swing components are lightweight – they are drawn by Java, not the OS. This makes them look the same on all platforms (Pluggable Look and Feel – PL&F).

AWT vs Swing	Details
AWT	Heavyweight – uses OS native components
Swing	Lightweight – Java-drawn, platform-independent appearance
AWT Package	java.awt.*
Swing Package	javax.swing.*
AWT Example	Button, TextField, Label
Swing Equivalent	JButton, JTextField, JLabel (all prefixed with J)

22.2 JOptionPane – Message Dialogs

```java
import javax.swing.*;

public class OptionPaneDemo {
    public static void main(String[] args) {
        // Input dialog
        String name = JOptionPane.showInputDialog("Enter your name:");

        // Different message dialog types
        JOptionPane.showMessageDialog(null,
            "Hello, " + name + "!",
            "Welcome",
            JOptionPane.INFORMATION_MESSAGE);

        JOptionPane.showMessageDialog(null,
            "Something went wrong!",
            "Error",
            JOptionPane.ERROR_MESSAGE);

        // Confirm dialog
        int choice = JOptionPane.showConfirmDialog(null,
            "Do you want to exit?",
            "Confirm",
            JOptionPane.YES_NO_OPTION);
        if (choice == JOptionPane.YES_OPTION) System.exit(0);
    }
}
```

JOptionPane Type	Icon Shown
INFORMATION_MESSAGE	Blue 'i' icon
WARNING_MESSAGE	Yellow '!' icon
ERROR_MESSAGE	Red 'X' icon
QUESTION_MESSAGE	Question mark icon

PLAIN_MESSAGE	No icon

22.3 JFrame

JFrame is the Swing equivalent of Frame. It has a built-in close behavior and a content pane for adding components.

```
import javax.swing.*;
import java.awt.*;

class MyJFrame extends JFrame {
    MyJFrame() {
        super("My JFrame");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null); // center on screen
        setBackground(Color.lightGray);
        setVisible(true);
    }
    public void paint(Graphics g) {
        super.paint(g);
        g.setFont(new Font("Arial", Font.BOLD, 20));
        g.setColor(Color.blue);
        g.drawString("Hello Swing JFrame!", 100, 150);
    }
}

public class JFrameDemo {
    public static void main(String[] args) {
        new MyJFrame();
    }
}
```

22.4 JPanel, JLabel, JButton

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class SwingBasics extends JFrame implements ActionListener {
    JPanel panel;
    JLabel label;
    JButton btnRed, btnBlue, btnGreen;

    SwingBasics() {
        super("Swing Basics");
        panel = new JPanel();

        label = new JLabel("Click a button to change background!");
        label.setFont(new Font("Arial", Font.BOLD, 14));

        btnRed   = new JButton("Red");
        btnBlue  = new JButton("Blue");
        btnGreen = new JButton("Green");

        // Tooltips
        btnRed.setToolTipText("Sets background to Red");
        btnBlue.setToolTipText("Sets background to Blue");
        btnGreen.setToolTipText("Sets background to Green");

        // Keyboard shortcuts (Alt+R, Alt+B, Alt+G)
        btnRed.setMnemonic('R');
        btnBlue.setMnemonic('B');
        btnGreen.setMnemonic('G');

        btnRed.addActionListener(this);
        btnBlue.addActionListener(this);
        btnGreen.addActionListener(this);

        panel.add(label);
        panel.add(btnRed); panel.add(btnBlue); panel.add(btnGreen);
        getContentPane().add(panel);

        setSize(500, 200);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent ae) {
        String cmd = ae.getActionCommand();
        if (cmd.equals("Red"))   panel.setBackground(Color.red);
        else if (cmd.equals("Blue"))  panel.setBackground(Color.blue);
        else panel.setBackground(Color.green);
    }

    public static void main(String[] args) { new SwingBasics(); }
}
```

Chapter 23: Swing Advanced Components

23.1 JTextField and JTextArea

```
import javax.swing.*; import java.awt.*; import java.awt.event.*;

public class TextDemo extends JFrame implements ActionListener {
    JTextField tfName, tfAge;
    JTextArea tArea;
    JButton submit;

    TextDemo() {
        super("JTextField & JTextArea Demo");
        JPanel p = new JPanel(new GridLayout(4, 2, 5, 5));
        p.add(new JLabel("Name:")); tfName = new JTextField(15); p.add(tfName);
        p.add(new JLabel("Age:"));  tfAge  = new JTextField(15); p.add(tfAge);
        submit = new JButton("Submit"); submit.addActionListener(this);
        p.add(submit);
        tArea = new JTextArea(5, 30);
        tArea.setEditable(false);
        JScrollPane scroll = new JScrollPane(tArea);
        getContentPane().setLayout(new BorderLayout());
        getContentPane().add(p, "North");
        getContentPane().add(scroll, "Center");
        setSize(400, 300);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent ae) {
        tArea.append("Name: " + tfName.getText() + "\n");
        tArea.append("Age: "  + tfAge.getText()  + "\n\n");
        tfName.setText(""); tfAge.setText("");
    }
    public static void main(String[] args) { new TextDemo(); }
}
```

23.2 JCheckBox and JRadioButton

```
import javax.swing.*; import java.awt.*; import java.awt.event.*;

public class CheckRadio extends JFrame implements ItemListener {
    JLabel result;
    JCheckBox java, python, cpp;
    JRadioButton male, female;

    CheckRadio() {
        super("JCheckBox & JRadioButton");
        JPanel panel = new JPanel(new GridLayout(6, 1));
        panel.add(new JLabel("Select Languages:"));
        java   = new JCheckBox("Java");
        python = new JCheckBox("Python");
        cpp    = new JCheckBox("C++");
        java.addItemListener(this); python.addItemListener(this);
cpp.addItemListener(this);
        panel.add(java); panel.add(python); panel.add(cpp);

        panel.add(new JLabel("Gender:"));
        ButtonGroup bg = new ButtonGroup();
        male   = new JRadioButton("Male", true);
        female = new JRadioButton("Female");
        bg.add(male); bg.add(female);
        JPanel gPanel = new JPanel();
        gPanel.add(male); gPanel.add(female);
        panel.add(gPanel);

        result = new JLabel("Selections will appear here");
        panel.add(result);
        getContentPane().add(panel);
        setSize(350, 280); setDefaultCloseOperation(EXIT_ON_CLOSE);
setVisible(true);
    }

    public void itemStateChanged(ItemEvent ie) {
        String sel = "";
        if (java.isSelected()) sel += "Java ";
        if (python.isSelected()) sel += "Python ";
        if (cpp.isSelected()) sel += "C++ ";
        result.setText("Selected: " + sel);
    }
    public static void main(String[] args) { new CheckRadio(); }
}
```

23.3 JComboBox (Drop-Down)

```
import javax.swing.*; import java.awt.*; import java.awt.event.*;

public class ComboDemo extends JFrame implements ActionListener {
    JComboBox<String> combo;
    JLabel msg;

    ComboDemo() {
        super("JComboBox Demo");
        String[] countries = {"India", "USA", "UK", "Japan", "Germany"};
        combo = new JComboBox<>(countries);
        combo.addActionListener(this);
        msg = new JLabel("Select a country");
        JPanel p = new JPanel();
        p.add(new JLabel("Country:")); p.add(combo); p.add(msg);
        getContentPane().add(p);
        setSize(400, 150); setDefaultCloseOperation(EXIT_ON_CLOSE);
setVisible(true);
    }
    public void actionPerformed(ActionEvent ae) {
        msg.setText("Selected: " + combo.getSelectedItem());
    }
    public static void main(String[] args) { new ComboDemo(); }
}
```

23.4 JList

```
import javax.swing.*; import javax.swing.event.*; import java.awt.*;

public class ListDemo extends JFrame implements ListSelectionListener {
    JList<String> list;
    JLabel msg;
    ListDemo() {
        super("JList Demo");
        String[] colors = {"Red", "Green", "Blue", "Yellow", "Pink", "Cyan"};
        list = new JList<>(colors);
        list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        list.addListSelectionListener(this);
        msg = new JLabel("Select a color");
        JPanel p = new JPanel();
        p.add(new JScrollPane(list));
        p.add(msg);
        getContentPane().add(p);
        setSize(300, 250); setDefaultCloseOperation(EXIT_ON_CLOSE);
setVisible(true);
    }
    public void valueChanged(ListSelectionEvent e) {
        String sel = list.getSelectedValue();
        msg.setText("Selected: " + sel);
    }
    public static void main(String[] args) { new ListDemo(); }
}
```

23.5 JSlider

```
import javax.swing.*; import javax.swing.event.*; import java.awt.*;

public class SliderDemo extends JFrame implements ChangeListener {
    JSlider red, green, blue;
    JPanel colorBox;

    SliderDemo() {
        super("RGB Slider Demo");
        red   = new JSlider(0, 255, 0);
        green = new JSlider(0, 255, 0);
        blue  = new JSlider(0, 255, 128);
        red.setMajorTickSpacing(50); red.setPaintTicks(true);
red.setPaintLabels(true);
        red.addChangeListener(this);
        green.addChangeListener(this);
        blue.addChangeListener(this);
        colorBox = new JPanel();
        colorBox.setPreferredSize(new Dimension(200, 100));
        JPanel controls = new JPanel(new GridLayout(4, 1));
        controls.add(new JLabel("Red:")); controls.add(red);
        controls.add(new JLabel("Green:")); controls.add(green);
        controls.add(new JLabel("Blue:")); controls.add(blue);
        getContentPane().setLayout(new BorderLayout());
        getContentPane().add(controls, "North");
        getContentPane().add(colorBox, "Center");
        setSize(400, 350); setDefaultCloseOperation(EXIT_ON_CLOSE);
setVisible(true);
        updateColor();
    }
    void updateColor() {
        colorBox.setBackground(new Color(red.getValue(), green.getValue(),
blue.getValue()));
    }
    public void stateChanged(ChangeEvent e) { updateColor(); }
    public static void main(String[] args) { new SliderDemo(); }
}
```

23.6 JMenuBar and JMenu

```
import javax.swing.*; import java.awt.*; import java.awt.event.*;

public class SwingMenu extends JFrame implements ActionListener {
    JPanel panel;

    SwingMenu() {
        super("Swing Menu Demo");
        panel = new JPanel();

        JMenuBar menuBar = new JMenuBar();

        // File Menu
        JMenu file = new JMenu("File");
        file.setMnemonic('F');
        JMenuItem newItem  = new JMenuItem("New",  'N');
        JMenuItem openItem = new JMenuItem("Open", 'O');
        JMenuItem saveItem = new JMenuItem("Save", 'S');
        JMenuItem exitItem = new JMenuItem("Exit", 'E');
        exitItem.addActionListener(this);
        file.add(newItem); file.add(openItem); file.addSeparator();
        file.add(saveItem); file.add(exitItem);

        // Color Menu
        JMenu colorMenu = new JMenu("Color");
        JMenuItem c1 = new JMenuItem("Red");
        JMenuItem c2 = new JMenuItem("Green");
        JMenuItem c3 = new JMenuItem("Blue");
        c1.addActionListener(this); c2.addActionListener(this);
c3.addActionListener(this);
        colorMenu.add(c1); colorMenu.add(c2); colorMenu.add(c3);

        menuBar.add(file); menuBar.add(colorMenu);
        setJMenuBar(menuBar);
        getContentPane().add(panel);
        setSize(400, 300); setDefaultCloseOperation(EXIT_ON_CLOSE);
setVisible(true);
    }

    public void actionPerformed(ActionEvent ae) {
        String cmd = ae.getActionCommand();
        switch(cmd) {
            case "Exit":  System.exit(0); break;
            case "Red":   panel.setBackground(Color.red);   break;
            case "Green": panel.setBackground(Color.green); break;
            case "Blue":  panel.setBackground(Color.blue);  break;
        }
    }
    public static void main(String[] args) { new SwingMenu(); }
}
```

23.7 JPopupMenu

```
import javax.swing.*; import java.awt.*; import java.awt.event.*;

public class SwingPopup extends JFrame implements ActionListener {
    JPanel panel;
    JPopupMenu popup;
    SwingPopup() {
        super("Right-Click Popup Menu");
        panel = new JPanel();
        popup = new JPopupMenu();
        JMenuItem r = new JMenuItem("Red");
        JMenuItem g = new JMenuItem("Green");
        JMenuItem b = new JMenuItem("Blue");
        r.addActionListener(this); g.addActionListener(this);
b.addActionListener(this);
        popup.add(r); popup.add(g); popup.addSeparator(); popup.add(b);

        panel.addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent e) {
                if (e.isPopupTrigger()) popup.show(e.getComponent(), e.getX(),
e.getY());
            }
            public void mouseReleased(MouseEvent e) {
                if (e.isPopupTrigger()) popup.show(e.getComponent(), e.getX(),
e.getY());
            }
        });
        getContentPane().add(panel);
        setSize(350, 250); setDefaultCloseOperation(EXIT_ON_CLOSE);
setVisible(true);
    }
    public void actionPerformed(ActionEvent ae) {
        switch(ae.getActionCommand()) {
            case "Red":   panel.setBackground(Color.red);   break;
            case "Green": panel.setBackground(Color.green); break;
            case "Blue":  panel.setBackground(Color.blue);  break;
        }
    }
    public static void main(String[] args) { new SwingPopup(); }
}
```

23.8 Swing Components Summary

Swing Component	Description & Use
JFrame	Top-level window; use setDefaultCloseOperation(EXIT_ON_CLOSE)
JPanel	Container for grouping components; add to content pane
JLabel	Display text or image; setFont(), setForeground()
JButton	Clickable button; setMnemonic(), setToolTipText()
JTextField	Single-line text input; getText(), setText()
JPasswordField	Text field that hides input with dots
JTextArea	Multi-line text; wrap in JScrollPane for scrolling
JCheckBox	Toggle checkbox; isSelected()
JRadioButton	Single selection; use ButtonGroup for mutual exclusion
JComboBox	Drop-down list; getSelectedItem()
JList	Scrollable list; getSelectedValue()

JSlider	Range selector; getValue(), addChangeListener()
JMenuBar	Menu bar attached to JFrame via setJMenuBar()
JMenu	Dropdown menu; add to JMenuBar
JMenuItem	Individual menu item; add to JMenu
JPopupMenu	Right-click context menu; show() at mouse position
JScrollPane	Adds scrollbars to any component
JOptionPane	Pre-built dialogs: message, input, confirm
JFileChooser	Swing file chooser dialog
JProgressBar	Visual progress indicator
JTable	Display data in rows and columns
JTree	Hierarchical tree display
JTabbedPane	Multiple tabs in one panel

Quick Reference Guide

Java Keywords

```
// Access: public  private  protected
// OOP:    class   extends  implements  interface  abstract  final
// Control:if  else  switch  case  break  continue  return
// Loops:  for  while  do
// Exception: try  catch  finally  throw  throws
// Object: new  this  super  instanceof  null
// Types:  int  double  float  long  char  boolean  byte  short  void
// Other:  static  import  package  synchronized  volatile
```

Java Compilation & Execution

Task	Command	Example
Compile	javac FileName.java	javac HelloWorld.java
Run	java ClassName	java HelloWorld
Run Applet	appletviewer File.html	appletviewer test.html
Jar create	jar cf myapp.jar *.class	packages compiled classes

Primitive Types Quick Reference

Type	Bits	Default Value
byte	8	0
short	16	0
int	32	0
long	64	0L
float	32	0.0f
double	64	0.0
char	16	'\u0000'
boolean	1	false

Wrapper Class Parsing

```
int    i = Integer.parseInt("42");
double d = Double.parseDouble("3.14");
float  f = Float.parseFloat("2.5");
long   l = Long.parseLong("99999");
boolean b = Boolean.parseBoolean("true");
```

Common Exception Types

Exception	Common Cause

ArithmeticException	int x = 5/0;
NullPointerException	String s = null; s.length();
ArrayIndexOutOfBoundsException	int[] a = {1,2}; a[5];
NumberFormatException	Integer.parseInt("abc");
ClassCastException	Object o = "hi"; Integer i = (Integer)o;
StackOverflowError	Infinite recursion

Swing vs AWT Component Names

AWT Component	Swing Equivalent	Notes
Frame	JFrame	setDefaultCloseOperation() available
Panel	JPanel	Add to getContentPane()
Button	JButton	Supports icons, mnemonics, tooltips
Label	JLabel	Supports HTML formatting
TextField	JTextField	getColumns(), getText()
TextArea	JTextArea	Wrap in JScrollPane
Checkbox	JCheckBox	isSelected() instead of getState()
Choice	JComboBox	getSelectedItem()
List	JList	getSelectedValue()
Scrollbar	JScrollBar	addAdjustmentListener()
Dialog	JDialog	setModal()
MenuBar	JMenuBar	setJMenuBar() on JFrame
Menu	JMenu	setMnemonic()
MenuItem	JMenuItem	setAccelerator() for shortcuts

Additional Programs & Exercises

A. Complete OOP Programs

A1. Library Book Management System

This program demonstrates a real-world use of classes, objects, constructors, and methods.

```
public class Book {
    private String title;
    private String author;
    private int year;
    private boolean available;

    // Constructor
    Book(String title, String author, int year) {
        this.title     = title;
        this.author    = author;
        this.year      = year;
        this.available = true;
    }

    void borrow() {
        if (available) {
            available = false;
            System.out.println("'" + title + "' borrowed successfully.");
        } else {
            System.out.println("'" + title + "' is not available.");
        }
    }

    void returnBook() {
        available = true;
        System.out.println("'" + title + "' returned. Thank you!");
    }

    void displayInfo() {
        System.out.println("Title:  " + title);
        System.out.println("Author: " + author);
        System.out.println("Year:   " + year);
        System.out.println("Status: " + (available ? "Available" : "Borrowed"));
        System.out.println("-----------------------------");
    }

    public static void main(String[] args) {
        Book b1 = new Book("Core Java", "Herbert Schildt", 2020);
        Book b2 = new Book("Head First Java", "Sierra & Bates", 2019);
        Book b3 = new Book("Effective Java", "Joshua Bloch", 2018);

        b1.displayInfo();
        b2.displayInfo();
        b3.displayInfo();

        b1.borrow();
        b1.borrow();   // try to borrow again
        b1.returnBook();
        b1.displayInfo();
    }
}
```

```
Output:
Title:  Core Java
Author: Herbert Schildt
Year:   2020
Status: Available
-----------------------------
(and more...)
Core Java borrowed successfully.
'Core Java' is not available.
'Core Java' returned. Thank you!
Status: Available
```

A2. Employee Payroll System using Inheritance

```
abstract class Employee {
    String name;
    int id;
    Employee(String name, int id) { this.name = name; this.id = id; }
    abstract double calculateSalary();
    void display() {
        System.out.println("ID: " + id + " | Name: " + name +
            " | Salary: Rs." + calculateSalary());
    }
}

class FullTimeEmployee extends Employee {
    double monthlySalary;
    FullTimeEmployee(String name, int id, double salary) {
        super(name, id);
        this.monthlySalary = salary;
    }
    double calculateSalary() { return monthlySalary; }
}

class PartTimeEmployee extends Employee {
    double hourlyRate;
    int hoursWorked;
    PartTimeEmployee(String name, int id, double rate, int hours) {
        super(name, id);
        this.hourlyRate  = rate;
        this.hoursWorked = hours;
    }
    double calculateSalary() { return hourlyRate * hoursWorked; }
}

class ContractEmployee extends Employee {
    double contractAmount;
    double taxRate;
    ContractEmployee(String name, int id, double amount, double tax) {
        super(name, id);
        this.contractAmount = amount;
        this.taxRate        = tax;
    }
    double calculateSalary() { return contractAmount * (1 - taxRate / 100); }
}

public class Payroll {
    public static void main(String[] args) {
        Employee[] staff = {
            new FullTimeEmployee("Alice",  101, 55000),
            new PartTimeEmployee("Bob",    102, 250, 80),
            new ContractEmployee("Carol",  103, 90000, 18.5),
            new FullTimeEmployee("David",  104, 72000),
        };

        System.out.println("===== PAYROLL REPORT =====");
        double total = 0;
        for (Employee e : staff) {
            e.display();
            total += e.calculateSalary();
        }
        System.out.println("Total Payout: Rs." + total);
    }
}
```

Output:

```
===== PAYROLL REPORT =====
ID: 101 | Name: Alice | Salary: Rs.55000.0
```

```
ID: 102 | Name: Bob | Salary: Rs.20000.0
ID: 103 | Name: Carol | Salary: Rs.73350.0
ID: 104 | Name: David | Salary: Rs.72000.0
Total Payout: Rs.220350.0
```

A3. Bank Account with Exception Handling

```
class InsufficientFundsException extends Exception {
    double amount;
    InsufficientFundsException(double amount) {
        this.amount = amount;
    }
    public String getMessage() {
        return "Insufficient funds! Short by: Rs." + amount;
    }
}

class BankAccount {
    private String owner;
    private double balance;

    BankAccount(String owner, double initialBalance) {
        this.owner   = owner;
        this.balance = initialBalance;
    }

    void deposit(double amount) {
        if (amount <= 0) {
            System.out.println("Invalid deposit amount!");
            return;
        }
        balance += amount;
        System.out.printf("Deposited Rs.%.2f | Balance: Rs.%.2f%n", amount,
balance);
    }

    void withdraw(double amount) throws InsufficientFundsException {
        if (amount > balance) {
            throw new InsufficientFundsException(amount - balance);
        }
        balance -= amount;
        System.out.printf("Withdrawn Rs.%.2f | Balance: Rs.%.2f%n", amount,
balance);
    }

    double getBalance() { return balance; }

    public static void main(String[] args) {
        BankAccount acc = new BankAccount("Alice", 10000);
        System.out.println("Account holder: " + acc.owner);

        acc.deposit(5000);
        try { acc.withdraw(3000); } catch (InsufficientFundsException e)
{ System.out.println(e.getMessage()); }
        try { acc.withdraw(20000); } catch (InsufficientFundsException e)
{ System.out.println(e.getMessage()); }
        System.out.println("Final Balance: Rs." + acc.getBalance());
    }
}
```

Output:

```
Account holder: Alice
Deposited Rs.5000.00 | Balance: Rs.15000.00
Withdrawn Rs.3000.00 | Balance: Rs.12000.00
Insufficient funds! Short by: Rs.8000.0
Final Balance: Rs.12000.0
```

B. String Programs

B1. Count Vowels, Consonants, Digits and Spaces

```
public class StringAnalysis {
    public static void main(String[] args) {
        String str = "Hello Java 2024 Programming!";
        int vowels = 0, consonants = 0, digits = 0, spaces = 0, others = 0;

        for (char c : str.toCharArray()) {
            if ("aeiouAEIOU".indexOf(c) >= 0) vowels++;
            else if (Character.isLetter(c))     consonants++;
            else if (Character.isDigit(c))       digits++;
            else if (c == ' ')                   spaces++;
            else                                 others++;
        }

        System.out.println("String:     " + str);
        System.out.println("Vowels:     " + vowels);
        System.out.println("Consonants: " + consonants);
        System.out.println("Digits:     " + digits);
        System.out.println("Spaces:     " + spaces);
        System.out.println("Others:     " + others);
    }
}
```

Output:

```
String:     Hello Java 2024 Programming!
Vowels:     8
Consonants: 18
Digits:     4
Spaces:     3
Others:     1
```

B2. Palindrome Check

```
public class Palindrome {
    static boolean isPalindrome(String s) {
        String rev = new StringBuilder(s).reverse().toString();
        return s.equalsIgnoreCase(rev);
    }

    public static void main(String[] args) {
        String[] words = {"RACECAR", "Hello", "MADAM", "Java", "LEVEL"};
        for (String w : words) {
            System.out.println(w + " -> " + (isPalindrome(w) ? "Palindrome" :
"Not Palindrome"));
        }
    }
}
```

Output:

```
RACECAR -> Palindrome
Hello -> Not Palindrome
MADAM -> Palindrome
Java -> Not Palindrome
LEVEL -> Palindrome
```

B3. Word Count and Frequency

```
import java.util.*;
```

```java
public class WordFrequency {
    public static void main(String[] args) {
        String text = "the cat sat on the mat the cat is fat";
        String[] words = text.split(" ");

        Map<String, Integer> freq = new LinkedHashMap<>();
        for (String word : words) {
            freq.put(word, freq.getOrDefault(word, 0) + 1);
        }

        System.out.println("Word Frequencies:");
        for (Map.Entry<String, Integer> e : freq.entrySet()) {
            System.out.printf("  %-10s -> %d%n", e.getKey(), e.getValue());
        }
        System.out.println("Total words: " + words.length);
        System.out.println("Unique words: " + freq.size());
    }
}
```

Output:

```
Word Frequencies:
  the        -> 3
  cat        -> 2
  sat        -> 1
  on         -> 1
  mat        -> 1
  is         -> 1
  fat        -> 1
Total words: 10
Unique words: 7
```

C. Collections Advanced Programs

C1. Student Grade Management using ArrayList

```
import java.util.*;

class Student {
    String name;
    int m1, m2, m3;
    Student(String name, int m1, int m2, int m3) {
        this.name = name; this.m1 = m1; this.m2 = m2; this.m3 = m3;
    }
    int total() { return m1 + m2 + m3; }
    double avg() { return total() / 3.0; }
    String grade() {
        double a = avg();
        if (a >= 75) return "Distinction";
        if (a >= 60) return "First Class";
        if (a >= 40) return "Second Class";
        return "Fail";
    }
    void print() {
        System.out.printf("%-12s %3d %3d %3d  Total:%-4d Avg:%.1f  %s%n",
            name, m1, m2, m3, total(), avg(), grade());
    }
}

public class GradeReport {
    public static void main(String[] args) {
        ArrayList<Student> students = new ArrayList<>();
        students.add(new Student("Alice",  85, 90, 78));
        students.add(new Student("Bob",    55, 60, 58));
        students.add(new Student("Carol",  30, 35, 28));
        students.add(new Student("David",  70, 65, 72));
        students.add(new Student("Eve",    95, 88, 92));

        System.out.println("=== STUDENT GRADE REPORT ===");
        System.out.printf("%-12s %3s %3s %3s  %-8s %-6s  %s%n",
            "Name", "M1", "M2", "M3", "Total", "Avg", "Grade");
        System.out.println("-".repeat(60));
        for (Student s : students) s.print();

        // Sort by average (highest first)
        students.sort((a, b) -> Double.compare(b.avg(), a.avg()));
        System.out.println("\nTop Student: " + students.get(0).name +
            " with avg: " + students.get(0).avg());
    }
}
```

Output:

```
=== STUDENT GRADE REPORT ===
Name         M1  M2  M3  Total    Avg    Grade
------------------------------------------------------------
Alice         85  90  78  Total:253 Avg:84.3  Distinction
Bob           55  60  58  Total:173 Avg:57.7  Second Class
Carol         30  35  28  Total:93  Avg:31.0  Fail
David         70  65  72  Total:207 Avg:69.0  First Class
Eve           95  88  92  Total:275 Avg:91.7  Distinction

Top Student: Eve with avg: 91.67
```

C2. Phone Book using HashMap

```
import java.util.*;

public class PhoneBook {
    public static void main(String[] args) {
        HashMap<String, String> book = new HashMap<>();

        // Add contacts
        book.put("Alice",   "9876543210");
        book.put("Bob",     "8765432109");
        book.put("Carol",   "7654321098");
        book.put("David",   "6543210987");

        // Display all
        System.out.println("=== Phone Book ===");
        for (Map.Entry<String, String> e : book.entrySet()) {
            System.out.printf("%-10s : %s%n", e.getKey(), e.getValue());
        }

        // Search
        String search = "Bob";
        if (book.containsKey(search)) {
            System.out.println("\n" + search + "'s number: " +
book.get(search));
        }

        // Delete
        book.remove("Carol");
        System.out.println("Contacts after deletion: " + book.size());

        // Update
        book.put("Alice", "1111111111");
        System.out.println("Updated Alice: " + book.get("Alice"));
    }
}
```

Output:

```
=== Phone Book ===
Alice      : 9876543210
Bob        : 8765432109
Carol      : 7654321098
David      : 6543210987

Bob's number: 8765432109
Contacts after deletion: 3
Updated Alice: 1111111111
```

D. Multithreading Programs

D1. Producer-Consumer Problem

```
// Demonstrates synchronized thread communication
class SharedBuffer {
    private int item = -1;
    private boolean hasItem = false;

    synchronized void produce(int val) throws InterruptedException {
        while (hasItem) wait();           // wait if buffer is full
        item    = val;
        hasItem = true;
        System.out.println("Produced: " + item);
        notifyAll();                       // wake up consumer
    }

    synchronized int consume() throws InterruptedException {
        while (!hasItem) wait();          // wait if buffer is empty
        hasItem = false;
        System.out.println("Consumed: " + item);
        notifyAll();                       // wake up producer
        return item;
    }
}

public class ProducerConsumer {
    public static void main(String[] args) {
        SharedBuffer buf = new SharedBuffer();
        Thread producer = new Thread(() -> {
            for (int i = 1; i <= 4; i++) {
                try { buf.produce(i * 10); Thread.sleep(200); }
                catch (InterruptedException e) {}
            }
        });
        Thread consumer = new Thread(() -> {
            for (int i = 0; i < 4; i++) {
                try { buf.consume(); Thread.sleep(300); }
                catch (InterruptedException e) {}
            }
        });
        producer.start();
        consumer.start();
    }
}
```

```
Output:
Produced: 10
Consumed: 10
Produced: 20
Consumed: 20
Produced: 30
Consumed: 30
Produced: 40
Consumed: 40
```

D2. Thread Priority

```
class PriorityThread extends Thread {
    PriorityThread(String name, int priority) {
        setName(name);
        setPriority(priority);
    }
    public void run() {
        for (int i = 0; i < 3; i++) {
            System.out.println(getName() + " (priority=" + getPriority() + ") - 
step " + (i+1));
        }
    }
}

public class PriorityDemo {
    public static void main(String[] args) {
        PriorityThread high = new PriorityThread("HIGH",  Thread.MAX_PRIORITY);
// 10
        PriorityThread mid  = new PriorityThread("MID",   Thread.NORM_PRIORITY);
// 5
        PriorityThread low  = new PriorityThread("LOW",   Thread.MIN_PRIORITY);
// 1
        low.start();
        mid.start();
        high.start();
    }
}
```

Output (HIGH thread usually runs first):

```
HIGH (priority=10) - step 1
HIGH (priority=10) - step 2
HIGH (priority=10) - step 3
MID (priority=5) - step 1
... (order not guaranteed but high priority preferred)
```

E. File I/O Programs

E1. Student Data File – Write and Read

```
import java.io.*;

public class StudentFile {
    public static void main(String[] args) {
        // Write student data
        String[] data = {
            "Alice,20,85.5",
            "Bob,21,72.0",
            "Carol,19,91.3",
            "David,22,65.8"
        };

        try (PrintWriter pw = new PrintWriter(new FileWriter("students.txt"))) {
            for (String line : data) {
                pw.println(line);
            }
            System.out.println("Data written to students.txt");
        } catch (IOException e) {
            System.out.println("Write error: " + e.getMessage());
        }

        // Read and process
        System.out.println("\n=== Reading Student Data ===");
        try (BufferedReader br = new BufferedReader(new
FileReader("students.txt"))) {
            String line;
            System.out.printf("%-10s %4s %6s%n", "Name", "Age", "Marks");
            System.out.println("-".repeat(24));
            while ((line = br.readLine()) != null) {
                String[] parts = line.split(",");
                System.out.printf("%-10s %4s %6s%n", parts[0], parts[1],
parts[2]);
            }
        } catch (IOException e) {
            System.out.println("Read error: " + e.getMessage());
        }
    }
}
```

Output:

```
Data written to students.txt

=== Reading Student Data ===
Name       Age  Marks
------------------------
Alice       20   85.5
Bob         21   72.0
Carol       19   91.3
David       22   65.8
```

E2. File Copy Program

```
import java.io.*;

public class FileCopy {
    public static void main(String[] args) {
        String source = "students.txt";
```

```java
        String dest   = "students_backup.txt";

        try (BufferedReader br = new BufferedReader(new FileReader(source));
             BufferedWriter bw = new BufferedWriter(new FileWriter(dest))) {
            String line;
            int count = 0;
            while ((line = br.readLine()) != null) {
                bw.write(line);
                bw.newLine();
                count++;
            }
            System.out.println("Copied " + count + " lines from '" + source
                + "' to '" + dest + "'");
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Output:

```
Copied 4 lines from 'students.txt' to 'students_backup.txt'
```

F. AWT Applications

F1. Complete Student Marks Calculator (AWT)

This program demonstrates a complete AWT application with GridLayout, multiple TextFields, Labels, and event handling to compute student results.

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class ResultSheet extends Applet implements ActionListener {
    TextField tName, tM1, tM2, tM3;
    Label lTotal, lAvg, lGrade, lResult;

    public void init() {
        setLayout(new GridLayout(6, 2, 5, 5));

        add(new Label("Student Name:")); tName = new TextField(15); add(tName);
        add(new Label("Mark 1:"));       tM1   = new TextField(10); add(tM1);
        add(new Label("Mark 2:"));       tM2   = new TextField(10); add(tM2);
        add(new Label("Mark 3:"));       tM3   = new TextField(10); add(tM3);

        Button calc = new Button("Calculate Result");
        calc.addActionListener(this);
        add(calc); add(new Label(""));

        add(new Label("Total:"));  lTotal  = new Label(""); add(lTotal);
        add(new Label("Average:")); lAvg    = new Label(""); add(lAvg);
        add(new Label("Grade:"));  lGrade  = new Label(""); add(lGrade);
        add(new Label("Result:"));  lResult = new Label(""); add(lResult);
    }

    public void actionPerformed(ActionEvent ae) {
        int m1 = Integer.parseInt(tM1.getText());
        int m2 = Integer.parseInt(tM2.getText());
        int m3 = Integer.parseInt(tM3.getText());
        int total = m1 + m2 + m3;
        double avg = total / 3.0;
        lTotal.setText(String.valueOf(total));
        lAvg.setText(String.format("%.1f", avg));
        if (m1 >= 40 && m2 >= 40 && m3 >= 40) {
            lResult.setText("Pass");
            if (avg >= 75) lGrade.setText("Distinction");
            else if (avg >= 60) lGrade.setText("First Class");
            else lGrade.setText("Second Class");
        } else {
            lResult.setText("Fail");
            lGrade.setText("No Grade");
        }
    }
}
```

F2. Simple Notepad Application (AWT)

A complete text editor built with AWT components – demonstrates Frame, MenuBar, TextArea, FileDialog, and clipboard operations.

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;

public class Notepad extends Frame implements ActionListener {
    TextArea textArea;
    String filename = "untitled";

    Notepad() {
        super("Notepad");
        textArea = new TextArea();
        add(textArea);

        MenuBar mb = new MenuBar();
        Menu file = new Menu("File");
        MenuItem newFile  = new MenuItem("New");
        MenuItem openFile = new MenuItem("Open");
        MenuItem saveFile = new MenuItem("Save");
        MenuItem exit     = new MenuItem("Exit");
        file.add(newFile); file.add(openFile); file.add(saveFile);
        file.addSeparator(); file.add(exit);
        mb.add(file);

        Menu edit = new Menu("Edit");
        edit.add(new MenuItem("Cut"));
        edit.add(new MenuItem("Copy"));
        edit.add(new MenuItem("Paste"));
        mb.add(edit);
        setMenuBar(mb);

        newFile.addActionListener(this);
        openFile.addActionListener(this);
        saveFile.addActionListener(this);
        exit.addActionListener(this);

        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) { System.exit(0); }
        });
        setSize(600, 400);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent ae) {
        String cmd = ae.getActionCommand();
        if (cmd.equals("New")) {
            textArea.setText(""); setTitle("untitled");
        } else if (cmd.equals("Open")) {
            FileDialog fd = new FileDialog(this, "Open", FileDialog.LOAD);
            fd.setVisible(true);
            if (fd.getFile() != null) {
                filename = fd.getDirectory() + fd.getFile();
                setTitle(filename);
                try (BufferedReader br = new BufferedReader(new
FileReader(filename))) {
                    StringBuilder sb = new StringBuilder();
                    String line;
                    while ((line = br.readLine()) != null)
sb.append(line).append("\n");
                    textArea.setText(sb.toString());
                } catch (Exception ex) { textArea.setText("Error reading file");
}
```

```
            }
        } else if (cmd.equals("Save")) {
            try (PrintWriter pw = new PrintWriter(new FileWriter(filename))) {
                pw.print(textArea.getText());
                System.out.println("Saved: " + filename);
            } catch (Exception ex) { System.out.println("Save failed"); }
        } else if (cmd.equals("Exit")) {
            System.exit(0);
        }
    }

    public static void main(String[] args) { new Notepad(); }
}
```

G. Swing Advanced Application

G1. Student Registration Form (Swing)

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class StudentForm extends JFrame implements ActionListener {
    JTextField tfName, tfAge, tfEmail;
    JComboBox<String> cbCourse;
    JRadioButton rbMale, rbFemale;
    JCheckBox cbJava, cbPython, cbSQL;
    JTextArea taOutput;
    JButton submit, clear;
    ButtonGroup genderGroup;

    StudentForm() {
        super("Student Registration Form");
        JPanel form = new JPanel(new GridLayout(7, 2, 10, 8));

        form.add(new JLabel("Name:"));
        tfName = new JTextField(20); form.add(tfName);

        form.add(new JLabel("Age:"));
        tfAge = new JTextField(5); form.add(tfAge);

        form.add(new JLabel("Email:"));
        tfEmail = new JTextField(20); form.add(tfEmail);

        form.add(new JLabel("Course:"));
        cbCourse = new JComboBox<>(new String[]{"BCA", "MCA", "BSc CS", "MSc
CS"});
        form.add(cbCourse);

        form.add(new JLabel("Gender:"));
        JPanel gPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        genderGroup = new ButtonGroup();
        rbMale   = new JRadioButton("Male", true);
        rbFemale = new JRadioButton("Female");
        genderGroup.add(rbMale); genderGroup.add(rbFemale);
        gPanel.add(rbMale); gPanel.add(rbFemale);
        form.add(gPanel);

        form.add(new JLabel("Skills:"));
        JPanel sPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        cbJava   = new JCheckBox("Java");
        cbPython = new JCheckBox("Python");
        cbSQL    = new JCheckBox("SQL");
        sPanel.add(cbJava); sPanel.add(cbPython); sPanel.add(cbSQL);
        form.add(sPanel);

        submit = new JButton("Submit");
        clear  = new JButton("Clear");
        submit.addActionListener(this);
        clear.addActionListener(this);
        JPanel btnPanel = new JPanel();
        btnPanel.add(submit); btnPanel.add(clear);
        form.add(btnPanel); form.add(new JLabel(""));
        taOutput = new JTextArea(8, 35);
        taOutput.setEditable(false);
        taOutput.setFont(new Font("Monospaced", Font.PLAIN, 13));
        JScrollPane scroll = new JScrollPane(taOutput);

        JPanel main = new JPanel(new BorderLayout(10, 10));
        main.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
        main.add(form, "North");
        main.add(scroll, "Center");
        getContentPane().add(main);

        setSize(500, 500);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent ae) {
        if (ae.getSource() == submit) {
            String skills = "";
            if (cbJava.isSelected())   skills += "Java ";
            if (cbPython.isSelected()) skills += "Python ";
            if (cbSQL.isSelected())    skills += "SQL ";
            taOutput.append("=== Registration Submitted ===\n");
            taOutput.append("Name:   " + tfName.getText()   + "\n");
            taOutput.append("Age:    " + tfAge.getText()    + "\n");
            taOutput.append("Email:  " + tfEmail.getText()  + "\n");
            taOutput.append("Course: " + cbCourse.getSelectedItem() + "\n");
            taOutput.append("Gender: " + (rbMale.isSelected() ? "Male" :
"Female") + "\n");
            taOutput.append("Skills: " + (skills.isEmpty() ? "None" : skills) +
"\n\n");
        } else {
            tfName.setText(""); tfAge.setText(""); tfEmail.setText("");
            rbMale.setSelected(true);
            cbJava.setSelected(false); cbPython.setSelected(false);
cbSQL.setSelected(false);
            taOutput.setText("");
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new StudentForm());
    }
}
```

Note: SwingUtilities.invokeLater() is the recommended way to start Swing applications – it ensures the GUI is created on the Event Dispatch Thread (EDT) for thread safety.

H. Pattern Programs

H1. Star Patterns

```
public class Patterns {
    public static void main(String[] args) {
        // Right triangle
        System.out.println("Right Triangle:");
        for (int i = 1; i <= 5; i++) {
            for (int j = 1; j <= i; j++) System.out.print("* ");
            System.out.println();
        }

        // Pyramid
        System.out.println("\nPyramid:");
        int n = 5;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n - i; j++) System.out.print(" ");
            for (int j = 1; j <= 2 * i - 1; j++) System.out.print("*");
            System.out.println();
        }

        // Diamond
        System.out.println("\nDiamond:");
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n - i; j++) System.out.print(" ");
            for (int j = 1; j <= 2 * i - 1; j++) System.out.print("*");
            System.out.println();
        }
        for (int i = n - 1; i >= 1; i--) {
            for (int j = 1; j <= n - i; j++) System.out.print(" ");
            for (int j = 1; j <= 2 * i - 1; j++) System.out.print("*");
            System.out.println();
        }
    }
}
```

```
Right Triangle:
*
* *
* * *
* * * *
* * * * *

Pyramid:
    *
   ***
  *****
 *******
*********

Diamond:
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
```

```
    *
```

H2. Number Programs

```
public class NumberPrograms {
    // Check prime
    static boolean isPrime(int n) {
        if (n < 2) return false;
        for (int i = 2; i <= Math.sqrt(n); i++) {
            if (n % i == 0) return false;
        }
        return true;
    }

    // Fibonacci series
    static void fibonacci(int count) {
        int a = 0, b = 1;
        System.out.print("Fibonacci: ");
        for (int i = 0; i < count; i++) {
            System.out.print(a + " ");
            int temp = a + b;
            a = b; b = temp;
        }
        System.out.println();
    }

    // Armstrong number check
    static boolean isArmstrong(int n) {
        int temp = n, sum = 0, digits = String.valueOf(n).length();
        while (temp != 0) {
            int d = temp % 10;
            sum  += (int) Math.pow(d, digits);
            temp /= 10;
        }
        return sum == n;
    }

    public static void main(String[] args) {
        // Prime numbers 1-50
        System.out.print("Primes (1-50): ");
        for (int i = 2; i <= 50; i++) if (isPrime(i)) System.out.print(i + " ");
        System.out.println();

        // Fibonacci
        fibonacci(10);

        // Armstrong numbers
        System.out.print("Armstrong (1-1000): ");
        for (int i = 1; i <= 1000; i++) if (isArmstrong(i)) System.out.print(i +
" ");
        System.out.println();
    }
}
```

Output:

```
Primes (1-50): 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
Fibonacci: 0 1 1 2 3 5 8 13 21 34
Armstrong (1-1000): 1 2 3 4 5 6 7 8 9 153 370 371 407
```

Important Java Concepts – Deep Dive

I. Java Memory Management & Garbage Collection

Java automatically manages memory using a Garbage Collector (GC). When no references point to an object, GC automatically frees that memory. This prevents memory leaks.

Memory Area	Description
Stack	Stores method calls and local variables; each thread has its own stack; LIFO
Heap	All Java objects are stored here; shared among all threads; GC runs here
Method Area	Stores class metadata, static fields, and method bytecodes
Program Counter	Stores address of current executing instruction for each thread

```
// Objects created with 'new' go to heap
String s = new String("Hello");  // s is on stack; "Hello" object on heap
int x = 5;                        // x is on stack (primitive, not object)

// When s goes out of scope, the String object becomes eligible for GC
// You can suggest GC runs (not guaranteed):
System.gc();

// finalize() called by GC before collecting (deprecated in Java 9+)
protected void finalize() {
    System.out.println("Object being collected by GC");
}
```

II. Java Generics

Generics enable type-safe collections and classes. They eliminate ClassCastException at runtime by catching type errors at compile time.

```
// Generic class – works with any type T
public class Pair<T, U> {
    T first;
    U second;

    Pair(T first, U second) {
        this.first  = first;
        this.second = second;
    }

    void display() {
        System.out.println("(" + first + ", " + second + ")");
    }

    public static void main(String[] args) {
        Pair<String, Integer> p1 = new Pair<>("Alice", 95);
        Pair<Integer, Double> p2 = new Pair<>(42, 3.14);
        Pair<String, String>  p3 = new Pair<>("Hello", "World");

        p1.display();  // (Alice, 95)
        p2.display();  // (42, 3.14)
        p3.display();  // (Hello, World)
    }
}
```

```
Output:
(Alice, 95)
(42, 3.14)
(Hello, World)
```

Generic Method

```
public class GenericMethods {
    // Generic method – T can be any type
    static <T extends Comparable<T>> T findMax(T a, T b) {
        return a.compareTo(b) > 0 ? a : b;
    }

    public static void main(String[] args) {
        System.out.println(findMax(10, 20));         // 20
        System.out.println(findMax("Apple", "Mango")); // Mango
        System.out.println(findMax(3.14, 2.71));    // 3.14
    }
}
```

```
Output:
20
Mango
3.14
```

III. Java Lambda Expressions (Functional Programming)

Lambda expressions (Java 8+) allow you to write inline implementations of functional interfaces (interfaces with exactly one abstract method). They make code shorter and more readable.

```
import java.util.*;

public class LambdaDemo {
    // Functional interface
    interface Greeting { void greet(String name); }
    interface MathOp  { int operate(int a, int b); }

    public static void main(String[] args) {
        // Lambda – replaces anonymous class
        Greeting g = (name) -> System.out.println("Hello, " + name + "!");
        g.greet("Java");

        // Lambda for math operations
        MathOp add = (a, b) -> a + b;
        MathOp mul = (a, b) -> a * b;
        System.out.println("Add: " + add.operate(5, 3));  // 8
        System.out.println("Mul: " + mul.operate(5, 3));  // 15

        // Sort list using lambda
        List<String> names = Arrays.asList("Charlie", "Alice", "Bob", "David");
        names.sort((s1, s2) -> s1.compareTo(s2));  // sort A-Z
        System.out.println("Sorted: " + names);

        // Filter using stream + lambda
        names.stream()
             .filter(n -> n.startsWith("A") || n.startsWith("B"))
             .forEach(n -> System.out.println("Match: " + n));
    }
}
```

```
Output:
Hello, Java!
Add: 8
Mul: 15
Sorted: [Alice, Bob, Charlie, David]
Match: Alice
Match: Bob
```

IV. String Formatting with printf and format()

```
public class FormatDemo {
    public static void main(String[] args) {
        // printf – formatted output
        System.out.printf("Name: %-15s Age: %3d Salary: %10.2f%n",
                          "Alice", 25, 55000.5);
        System.out.printf("Name: %-15s Age: %3d Salary: %10.2f%n",
                          "Bob", 30, 72500.75);

        // String.format – build formatted string
        String report = String.format("|%-10s|%5d|%8.2f|",
                                      "Carol", 28, 65000.0);
        System.out.println(report);

        // Format specifiers
        System.out.printf("%d in hex: %X%n",   255, 255);    // FF
        System.out.printf("%d in octal: %o%n", 8, 8);       // 10
        System.out.printf("PI = %.4f%n",        Math.PI);   // 3.1416
        System.out.printf("%10s | %-10s%n",    "Right", "Left"); // aligned
    }
}
```

```
Output:
Name: Alice           Age:  25 Salary:   55000.50
Name: Bob             Age:  30 Salary:   72500.75
|Carol     |   28| 65000.00|
255 in hex: FF
8 in octal: 10
PI = 3.1416
     Right | Left
```

V. Java Enumeration (enum)

Enums define a fixed set of named constants. They are safer and more readable than using integer constants.

```
public class EnumDemo {
    enum Day { MON, TUE, WED, THU, FRI, SAT, SUN }
    enum Planet {
        MERCURY(3.303e+23, 2.4397e6),
        VENUS  (4.869e+24, 6.0518e6),
        EARTH  (5.976e+24, 6.37814e6);

        final double mass;
        final double radius;
        Planet(double mass, double radius) {
            this.mass   = mass;
            this.radius = radius;
        }
        double surfaceGravity() { return 6.67300E-11 * mass / (radius * radius);
}
        double surfaceWeight(double otherMass) { return otherMass *
surfaceGravity(); }
    }

    public static void main(String[] args) {
        Day today = Day.WED;
        switch (today) {
            case MON: case TUE: case WED: case THU: case FRI:
                System.out.println(today + " is a weekday"); break;
            default:
                System.out.println(today + " is weekend");
        }

        double earthWeight = 75.0;
        double mass = earthWeight / Planet.EARTH.surfaceGravity();
        for (Planet p : Planet.values()) {
            System.out.printf("Weight on %-8s = %6.2f%n", p,
p.surfaceWeight(mass));
        }
    }
}
```

Output:

```
WED is a weekday
Weight on MERCURY =  28.33
Weight on VENUS   =  67.90
Weight on EARTH   =  75.00
```

VI. Nested Classes and Anonymous Classes

Inner (Nested) Class

```
public class Outer {
    private int x = 10;

    class Inner {
        void display() {
            System.out.println("Outer x = " + x); // can access outer's private
members
        }
    }

    static class StaticNested {
        void show() {
            System.out.println("Static nested class – no outer instance
needed");
        }
    }

    public static void main(String[] args) {
        Outer outer = new Outer();
        Outer.Inner inner = outer.new Inner();
        inner.display();

        Outer.StaticNested sn = new Outer.StaticNested();
        sn.show();
    }
}
```

Output:

```
Outer x = 10
Static nested class – no outer instance needed
```

Anonymous Class

```
interface Drawable { void draw(); }

public class AnonClass {
    public static void main(String[] args) {
        // Anonymous class implementing Drawable – no need to create a named
class
        Drawable circle = new Drawable() {
            public void draw() {
                System.out.println("Drawing a circle!");
            }
        };

        Drawable square = new Drawable() {
            public void draw() {
                System.out.println("Drawing a square!");
            }
        };

        circle.draw();
        square.draw();
    }
}
```

Output:

```
Drawing a circle!
Drawing a square!
```

VII. Java Date and Time (java.util.Date)

```
import java.util.*;
import java.text.SimpleDateFormat;

public class DateDemo {
    public static void main(String[] args) {
        Date now = new Date();
        System.out.println("Raw date:    " + now);

        // Format date
        SimpleDateFormat fmt1 = new SimpleDateFormat("dd/MM/yyyy");
        SimpleDateFormat fmt2 = new SimpleDateFormat("EEEE, MMMM dd, yyyy
HH:mm:ss");

        System.out.println("Formatted:   " + fmt1.format(now));
        System.out.println("Full date:   " + fmt2.format(now));

        // Calendar
        Calendar cal = Calendar.getInstance();
        System.out.println("Year:  " + cal.get(Calendar.YEAR));
        System.out.println("Month: " + (cal.get(Calendar.MONTH) + 1)); // 0-
indexed
        System.out.println("Day:   " + cal.get(Calendar.DAY_OF_MONTH));
        System.out.println("Hour:  " + cal.get(Calendar.HOUR_OF_DAY));
    }
}
```

Output:

```
Raw date:    Sat Feb 28 11:30:00 IST 2026
Formatted:   28/02/2026
Full date:   Saturday, February 28, 2026 11:30:00
Year:  2026
Month: 2
Day:   28
Hour:  11
```

VIII. Sorting Algorithms in Java

Bubble Sort

```
public class BubbleSort {
    static void bubbleSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    // swap
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }
    public static void main(String[] args) {
        int[] nums = {64, 34, 25, 12, 22, 11, 90};
        System.out.print("Before: ");
        for (int n : nums) System.out.print(n + " ");
        bubbleSort(nums);
        System.out.print("\nAfter:  ");
        for (int n : nums) System.out.print(n + " ");
        System.out.println();
    }
}
```

```
Output:
Before: 64 34 25 12 22 11 90
After:  11 12 22 25 34 64 90
```

Selection Sort

```
public class SelectionSort {
    static void selectionSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            int minIdx = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIdx]) minIdx = j;
            }
            // Swap minimum with current position
            int temp = arr[minIdx]; arr[minIdx] = arr[i]; arr[i] = temp;
        }
    }
    public static void main(String[] args) {
        int[] arr = {29, 10, 14, 37, 13};
        selectionSort(arr);
        System.out.print("Sorted: ");
        for (int n : arr) System.out.print(n + " ");
    }
}
```

```
Output:
Sorted: 10 13 14 29 37
```

Binary Search

```
public class BinarySearch {
    static int binarySearch(int[] arr, int target) {
        int left = 0, right = arr.length - 1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (arr[mid] == target) return mid;
            if (arr[mid] < target) left  = mid + 1;
            else                   right = mid - 1;
        }
        return -1;  // not found
    }
    public static void main(String[] args) {
        int[] sorted = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};
        System.out.println("Array: 2 5 8 12 16 23 38 56 72 91");
        int pos = binarySearch(sorted, 23);
        System.out.println("Search 23: found at index " + pos);
        pos = binarySearch(sorted, 99);
        System.out.println("Search 99: " + (pos == -1 ? "Not found" : "At " +
pos));
    }
}
```

```
Output:
Array: 2 5 8 12 16 23 38 56 72 91
Search 23: found at index 5
Search 99: Not found
```

IX. Java Best Practices

Best Practice	Why It Matters
Use meaningful variable names	Code readability: int studentAge instead of int a
Use constants for magic numbers	final int MAX_STUDENTS = 30; instead of if(count > 30)
Follow naming conventions	Classes: PascalCase, variables/methods: camelCase, constants: UPPER_SNAKE
Handle exceptions properly	Never use empty catch blocks – at least log the error
Close resources in finally or try-with-resources	Prevents resource leaks for files, connections
Use StringBuilder for string concatenation in loops	String + in a loop creates many objects; SB is mutable
Prefer interfaces over concrete classes	Loose coupling: List<String> instead of ArrayList<String>
Use @Override annotation	Compiler checks you're actually overriding a method
Keep methods short and focused	A method should do ONE thing well
Write comments for complex logic	// Why, not what – code shows what, comments show why

X. Java Compilation Errors vs Runtime Errors

Error Type	Description & Example
Syntax Error (Compile-time)	Missing semicolon, wrong brackets: int x = 5 // error
Semantic Error (Compile-time)	Type mismatch: int x = "hello"; // error
Logic Error (Runtime / no error)	Wrong algorithm: a + b instead of a - b
Runtime Exception	Correct syntax but fails at runtime: 5/0, null.length()
Error	Serious JVM issues: StackOverflowError, OutOfMemoryError

XI. Java Naming Conventions

Element	Convention	Example
Class/Interface	PascalCase – start each word with uppercase	StudentRecord, BankAccount
Method	camelCase – start lowercase, capitalize next words	calculateSalary(), getStudentName()
Variable	camelCase	firstName, totalAmount, isActive
Constant	UPPER_SNAKE_CASE – all caps with underscores	MAX_SIZE, PI, DEFAULT_TIMEOUT

Package	all lowercase – usually reverse domain	com.company.project.util
Parameter	camelCase (same as variable)	void setName(String firstName)

XII. Common Java Programs Summary

Program Type	Key Concepts Used
Calculator	switch/if-else, Scanner input, basic arithmetic
Student Grade	if-else chains, arithmetic, Strings
Fibonacci	loops, arrays or recursion
Factorial	recursion or loop, long for large values
Palindrome	String manipulation, StringBuilder reverse
Prime Check	for loop, Math.sqrt, modulo operator
Sort Array	nested loops (bubble/selection) or Arrays.sort()
File Read/Write	FileReader/Writer, BufferedReader, IOException
Thread Demo	Thread/Runnable, start(), sleep(), join()
AWT GUI	Frame/Applet, Button, TextField, ActionListener
Swing Form	JFrame, JPanel, JTextField, JComboBox, ActionListener
Collections	ArrayList, HashMap, Iterator, for-each loop