Java Programming Lab Exam
Total: 40 Marks

January 2026

Instructions

- All solutions must be written in **Java**.
- Follow standard naming conventions and proper indentation.
- Demonstrate the required concepts clearly inside the `main()` method.
- Submit a single **PDF** file containing: source code + sample input/output for each task.

1 Task 1

Create a class `Movie` with:

- Private fields: `title` (String), `director` (String), `releaseYear` (int)
- Default constructor: sets "Untitled", "Unknown Director", 2000
- Parameterized constructor that accepts all three parameters
- Public getters for all fields
- Method `printDetails()` that prints:
  `Movie: [title] | Director: [director] | Year: [releaseYear]`

In `main()`, create two `Movie` objects (one default, one parameterized) and display their information.

2 Task 2

Create class `Student` containing:

- Instance fields: `name` (String), `rollNo` (String)
- Static field: `totalStudents` initialized to 0
- Parameterized constructor that receives name & roll number, increments `totalStudents`, uses `this`
- Method `display()` prints: `"Roll: [rollNo] [name]"`
- Static method `getStudentCount()` that returns the current count

In `main()`, instantiate at least 4 students, show each students details, and finally print the total number of students.

3 Task 3

Create base class `Gadget` with:

- Fields: `brand` (String), `price` (double)
- Constructor to set both values
- Method `describe()` `"[brand] gadget priced at [price]"`

Create derived class `Smartwatch` that:

- Adds field: `waterResistant` (boolean)
- Overrides `describe()` `"[brand] smartwatch Price: [price] Water resistant: [Yes/No]"`
- Adds method `showTime()` prints "Showing current time"

In `main()`, create one `Gadget` and one `Smartwatch`. Call `describe()` on both using a `Gadget` reference variable for the smartwatch (demonstrate overriding).

4 Task 4

Implement multilevel inheritance: `Product` → `ElectricItem` → `Laptop`

- `Product`: field `weightKg` (double), method `startUsing()` "Using product"
- `ElectricItem`: field `ratingOutOf5` (float), constructor calling super, method `displayRating()`
- `Laptop`: field `screenSizeInch` (int), overrides `startUsing()` calls super + adds "Laptop [screenSizeInch] is starting"

In `main()`, create one `Laptop` object and call `startUsing()` and `displayRating()`.

5 Task 5

Create abstract class `Figure` with:

- Abstract method `render()`
- Concrete method `getFigureType()` returns "Basic Figure"

Create three concrete subclasses:

- `Ellipse render()` prints "Rendering ellipse"
- `Square render()` prints "Rendering square"
- `Pentagon render()` prints "Rendering pentagon"

In `main()`:

- Create array of `Figure` type with size 5
- Fill with a mixture of Ellipse, Square, and Pentagon objects
- Iterate array and call `render()` on each element

6 Task 6

Create a **final** class named `ScientificValues` containing:

- `public static final double PI = 3.14159265359;`
- `public static final double PLANCK_CONSTANT = 6.62607015e-34;`
- Final method `displayValues()` that prints both constants
- In the comment, write what happens if someone tries to extend `ScientificValues`
- In `main()`, call `displayValues()` using class name

7 Task 7

Create two interfaces:

- `Flyable` with method `fly()`
- `Swimmable` with method `swim()`

Create abstract class `Creature` with:

- Field `species`
- Constructor
- Abstract method `consumeFood()`

Create two classes:

- `Penguin` extends `Creature` implements `Swimmable` (not Flyable)
- `Duck` extends `Creature` implements `Flyable`, `Swimmable`

Implement all required methods reasonably.
In `main()`, create one Penguin and one Duck, call `fly()`, `swim()`, `consumeFood()` where applicable (demonstrate interface usage).

8 Task 8

Create abstract class `Account` with:

- Protected fields: `accId` (String), `currentBalance` (double)
- Constructor
- Abstract method `withdraw(double amt)`
- Concrete method `credit(double amt)`
- Concrete method `showBalance()`

Create two subclasses:

- `FixedDepositAccount` cannot withdraw (print message "Cannot withdraw from FD")
- `SalaryAccount` can withdraw normally, but balance cannot go below 500

In `main()`:

- Create array of `Account` references (at least 3 accounts mix of both types)
- Perform several deposits and withdrawals
- Print final balance of each account