Java Programming Laboratory 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 class `Character` with:

- Private fields: `name` (String), `level` (int), `classType` (String)
- No-arg constructor → "Newbie", 1, "Adventurer"
- All-args constructor
- Public getters for all fields
- Method `showCharacterCard()` that prints:
  `Name: [name] Lv.[level] Class: [classType]`

In `main()`, create one character using default constructor and one using parameterized constructor. Display both cards.

2 Task 2

Create class `Course` with:

- Instance fields: `courseCode` (String), `title` (String)
- Static field: `enrolledCount` = 0
- Parameterized constructor (code, title) that increases `enrolledCount`
- Method `printInfo()` → `"[courseCode]  [title]"`
- Static method `getEnrollmentTotal()` returning current count

In `main()`, create 5 different courses, show info for each, and finally display total enrolled students.

3 Task 3

Base class `Transport` with:

- Fields: `model` (String), `capacity` (int)
- Constructor
- Method `announce()` → `"[model] can carry up to [capacity] people"`

Derived class `ElectricBus` that:

- Adds field: `batteryRangeKm` (int)
- Overrides `announce()` → `"Electric bus [model]  capacity [capacity], range [batteryRangeKm] km"`
- New method `chargeStatus()` → prints "Charging battery"

In `main()`, create one `Transport` and one `ElectricBus`.
Call `announce()` on both objects using a `Transport` reference for the bus.

4 Task 4

Hierarchy: `Appliance` → `HomeAppliance` → `AirConditioner`

- `Appliance`: field `voltage` (int), method `powerOn()` "Appliance powered on"
- `HomeAppliance`: field `energyClass` (char), create constructor with super, method `showEfficiency()`
- `AirConditioner`: field `coolingPowerBTU` (int), overrides `powerOn()` → super call + "A/C starting cooling at [coolingPowerBTU] BTU"

In `main()`, instantiate one `AirConditioner` and demonstrate both methods.

5 Task 5

Create abstract class `Artwork` with:

- Abstract method `exhibit()`
- Concrete method `category()` → prints `"Artwork"`

Create three subclasses:

- `Painting` → `exhibit()` prints "Hanging painting on wall"
- `Sculpture` → `exhibit()` prints "Placing sculpture on pedestal"
- `DigitalArt` → `exhibit()` prints "Displaying digital artwork on screen"

In `main()`:

- Array of `Artwork` (size 6)
- Fill with at least one of each type (repeat allowed)
- Loop and call `exhibit()` on every element

6 Task 6

Create **final** class `UnitConverter` with:

- `public static final double INCH_TO_CM = 2.54;`
- `public static final double FEET_TO_CM = 30.48;`
- Final method `printConversionTable()` that shows both constants
  - Add comment explaining why someone cannot create:
    `class MetricConverter extends UnitConverter {}` // compile error
  - In `main()`, call the table-printing method via class name

7 Task 7

Two interfaces:

- `Trackable` with method `getLocation()`
- `Notifiable` with method `sendNotification()`

Abstract class `DeliveryItem` with:

- Field `trackingId` (String)
- Constructor
- Abstract method `deliver()`

Two concrete classes:

- `FoodOrder` extends `DeliveryItem` implements `Trackable`, `Notifiable`
- `Parcel` extends `DeliveryItem` implements `Trackable` (no notification)

Implement reasonable behavior for all methods.
In `main()`, create one food order and one parcel. Call all applicable interface + abstract methods (show polymorphism).

8 Task 8

Abstract class `Payment` with:

- Protected fields: `transactionId` (String), `amount` (double)
- Constructor
- Abstract method `processPayment()`
- Concrete method `addTip(double tip)`
- Concrete method `getFinalAmount()`

Two concrete subclasses:

- `CashPayment` process prints "Cash received", tip not allowed
- `CreditCardPayment` process prints "Card charged", allows tip (adds to amount)

In `main()`:

- Array of `Payment` references (at least 4 mix of both types)
- Perform payments, add tips where possible
- Show final amount for each transaction

9 Task 9

Create a class `Complex` to represent complex numbers with the following requirements:

- Private fields: `real` (double), `img` (double)
- Constructor: `Complex(double real, double img)`
- Methods:
  - `Complex add(Complex other)` returns a new Complex object representing the sum
  - `Complex multiply(Complex other)` returns a new Complex object representing the product
  - `String toString()` returns a properly formatted string:

  - `3.0 + 4.0i`
  - `-2.5 - 1.1i`
  - `7.0` (when imaginary part is 0)
  - `-5.2i` (when real part is 0)
  - `0.0` (for zero)
- `void display()` prints the complex number using the format from `toString()`

In `main()`:

- Create at least two `Complex` objects (e.g., $3 + 4i$ and $1 - 2i$)
- Compute their sum and product using the `add()` and `multiply()` methods
- Display all three numbers (the two originals + sum + product) with clear labels
- Example output format:

```
Complex number 1: 3.0 + 4.0i
Complex number 2: 1.0 - 2.0i
Sum             : 4.0 + 2.0i
Product         : 11.0 + 2.0i
```