OOP Advanced Assignment (Abstraction, Polymorphism & Static Members)

Task 1: Abstraction in Action

Create an **abstract class** `Device` with:

- `brand` (String)
- An **abstract method** `turnOn()`

A **non-abstract method** `showBrand()` that prints:
Device brand: {brand}

Then, create **two subclasses**:

1. **`Laptop`** (extra field: `ramSize`)

Implement `turnOn()` to print:
Laptop is turning on with {ramSize}GB RAM.

2. **`Smartphone`** (extra field: `screenSize`)

Implement `turnOn()` to print:
Smartphone is turning on with {screenSize}-inch display.

✅ **Create objects of `Laptop` & `Smartphone`, call `showBrand()` and `turnOn()`.**

Task 2: Polymorphism - Shape Drawing System

Create an **abstract class** `Shape` with:

- An **abstract method** `calculateArea()`

Then, create two **subclasses**:

1. **`Circle`** (extra field: `radius`)
   - Implement `calculateArea()` to return **π * radius²**
2. **`Rectangle`** (extra fields: `length`, `width`)
   - Implement `calculateArea()` to return **length × width**

✅ **Create a list of `Shape` objects and call `calculateArea()` for each one.**

Task 3: Real-World Polymorphism - Payment System

Create an **abstract class** `Payment` with:

- `amount`
- An **abstract method** `processPayment()`

Then, create **two subclasses**:

1. **`CashPayment`** (no extra field)

`processPayment()` should print:
Cash payment of {amount} received.

2. **`CardPayment`** (extra field: `cardNumber`)

`processPayment()` should print:
Card payment of {amount} processed with card {cardNumber}. ✅ **Create**

payment objects, store them in a list, and call `processPayment()`.

Task 4: Static Variable & Method - User Counter

Create a `User` class with:

- `username` (String)
- A **static variable** `userCount` to track the total users
- A **constructor** that increments `userCount` when a new user is created

A **static method** `showTotalUsers()` that prints:
Total registered users: {userCount}

-

✅ **Create multiple users and test `showTotalUsers()`.**

Task 5: Managing Employees using Abstraction & Static Methods
Create an **abstract class** `Employee` with:

- `name`, `salary`
- An **abstract method** `calculateBonus()`

Then, create **two subclasses**:

1. **`Developer`**
    - Implement `calculateBonus()` as **salary * 0.1**
2. **`Manager`**
    - Implement `calculateBonus()` as **salary * 0.2**

✅ **Create employee objects, store them in a list, and calculate bonuses for all.**