Assignment: Creational Design Patterns in Java

Objective

To understand and implement **five major Creational Design Patterns** in Java:

1. **Singleton**
2. **Factory Method**
3. **Builder**
4. **Prototype**
5. **Abstract Factory**

Each task is small and focused — total time ~**1.5 hours**.

Instructions

- Use **Java 17 or higher**.
- Each pattern should be implemented as a **separate mini-program**.
- Include **short comments** explaining the logic.
- Name your main class as <PatternName>Demo (e.g., SingletonDemo.java).
- Keep the code simple — no complex business logic needed.

Question 1 – Singleton Pattern (15 min)

Implement a **Logger** class using the **Singleton Pattern**.

- Only one instance of Logger should exist.
- It should have a method log(String message) that prints a timestamped message.
- Demonstrate that multiple calls return the same instance.

Hint: Use a private static Logger instance and getInstance().

Question 2 – Factory Method Pattern (15 min)

Create a **Shape Factory** using the **Factory Method Pattern**.

- Define an interface Shape with draw().
- Implement classes: Circle, Rectangle, and Square.
- Create a ShapeFactory that returns the correct Shape object based on input.
- In Main, create and draw different shapes.

Hint: The client should not use new directly for shape creation.

Question 3 – Builder Pattern (20 min)

Create a **Builder** for a Computer class.

- Fields: CPU, RAM, storage, graphicsCard.
- Use a static nested Builder class.
- Demonstrate building two different configurations using method chaining.

Hint: Example:

```
Computer gamingPC = new Computer.Builder()
  .setCPU("i9")
  .setRAM(32)
  .setStorage(1000)
  .setGraphicsCard("RTX 4080")
  .build();
```

Question 4 – Prototype Pattern (20 min)

Implement the **Prototype Pattern** for a Document class.

- The class should have fields like title and content.
- Implement a clone() method that creates a deep copy.
- Show that modifying the clone doesn't affect the original.

Hint: Implement Cloneable and override clone() properly.

Question 5 – Abstract Factory Pattern (25 min)

Create an **Abstract Factory** for a **UI theme system**.

- Define interfaces:
    - Button with paint()
    - Checkbox with check()
- Implement two families:
    - **LightThemeFactory** (creates LightButton, LightCheckbox)
    - **DarkThemeFactory** (creates DarkButton, DarkCheckbox)
- In Main, choose a theme (e.g., "light" or "dark") and create UI elements from that factory.

Hint: Demonstrates how one factory can create related object families.

Bonus Challenge (Optional – 10 min)

Add a **lazy initialization** Singleton or use **enum-based Singleton** for better thread safety.

Deliverables

- Five .java files (one per pattern)
- Optional README.md describing:
    - What each pattern does
    - When it should be used