Java Design Patterns Explained with Examples

Singleton Pattern

Definition: Ensures a class has only one instance and provides a global point of access to it.

```
Example:
class Singleton {
    private static Singleton instance;
    private Singleton() {}
    public static Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
}
```

Interview Tip: Explain how Singleton is used for logging, configuration, or thread pools.

Factory Pattern

Definition: Defines an interface for creating an object, but lets subclasses alter the type of objects that will be created.

```
Example:
interface Shape {
    void draw();
}
class Circle implements Shape {
    public void draw() { System.out.println("Circle"); }
}
class ShapeFactory {
    public Shape getShape(String type) {
        if (type.equals("CIRCLE")) return new Circle();
        return null;
    }
}
```

Interview Tip: Mention it promotes loose coupling and is used when object creation logic is complex.

Observer Pattern

Definition: Defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified.

```
Example:
```

```
interface Observer {
    void update();
}
class Subject {
    List<Observer> observers = new ArrayList<>();
    void attach(Observer o) { observers.add(o); }
    void notifyAllObservers() {
        for (Observer o : observers) o.update();
    }
}
```

Interview Tip: Useful in event-driven systems, like GUI or messaging systems.