Lab 5 Theory Questions

Describe briefly the fundamental characteristics of creational design patterns.

Creational design patterns are all about how objects are created in a flexible and reusable way. Instead of creating objects directly using constructors, these patterns provide different techniques to manage object creation. Some key characteristics include:

- **Encapsulation of Object Creation** – These patterns hide the actual instantiation logic, making code easier to manage.
- **Flexibility** – They allow different representations of objects without changing the code that uses them.
- **Better Maintainability** – Since object creation is handled separately, it reduces dependencies and makes the codebase cleaner.
- **Efficient Resource Management** – Some creational patterns ensure that objects are reused efficiently to improve performance.

Examples of creational design patterns include:

- **Singleton** – Ensures only one instance of a class exists.
- **Factory Method** – Provides a way to create objects without specifying their exact type.
- **Abstract Factory** – Helps create families of related objects without specifying their concrete classes.
- **Builder** – Used to construct complex objects step by step.
- **Prototype** – Creates objects by cloning an existing one.

What is the difference between an Abstract and Concrete class?

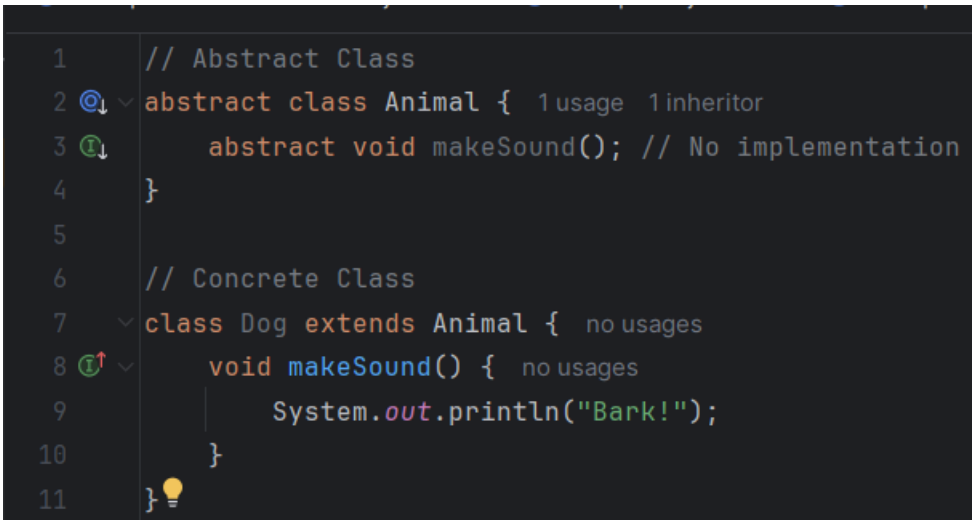
An abstract class is a class that cannot be instantiated on its own and is meant to be inherited by other classes. It can contain both abstract methods (methods without implementation) and regular methods. It serves as a blueprint for other classes.

A concrete class, on the other hand, is a fully implemented class that can be instantiated directly. It provides implementations for all methods and can be used to create objects.

Key Differences:

Feature	Abstract Class	Concrete Class
Definition	A class that acts as a template and may have unimplemented methods.	A complete class with full implementations.
Instantiation	Cannot be instantiated directly.	Can be instantiated normally.
Methods	Can have abstract (no implementation) and concrete (implemented) methods.	All methods must be fully implemented.
Purpose	Provides a common structure for subclasses.	Represents a fully functional object.

Example:



Here, Animal is abstract and cannot be used directly, but Dog is a concrete class that can be instantiated.

What is the intent of the builder design pattern?

The Builder Pattern is used when we need to create complex objects with many optional parameters in a structured way. Instead of using multiple constructors (which can get confusing), the builder pattern allows us to create objects step by step.

Why Use It?

- Avoids telescoping constructors – If a class has too many parameters, constructors can get messy.

- Improves code readability – The step-by-step approach makes object creation clearer.
- Encapsulates object creation – The builder class handles the construction logic, making the main class simpler.
- Ensures immutability – The object is only built once and can't be modified later.

Example:

```
public static void main(String[] args) {
    // Creating a Computer using the Builder Pattern
    Computer customComputer = new Computer.Builder( RAM: "16GB",  HDD: "1TB",  CPU: "Intel i9")
            .setGraphicsCardEnabled(true) // Enabling Graphics Card
            .setBluetoothEnabled(true) // Enabling Bluetooth
            .build();
```

Here, we first specify the required attributes (RAM, HDD, and CPU), then optionally enable the graphics card and Bluetooth before calling .build(). This approach makes object creation easier and more flexible.