Design Patterns

Introduction

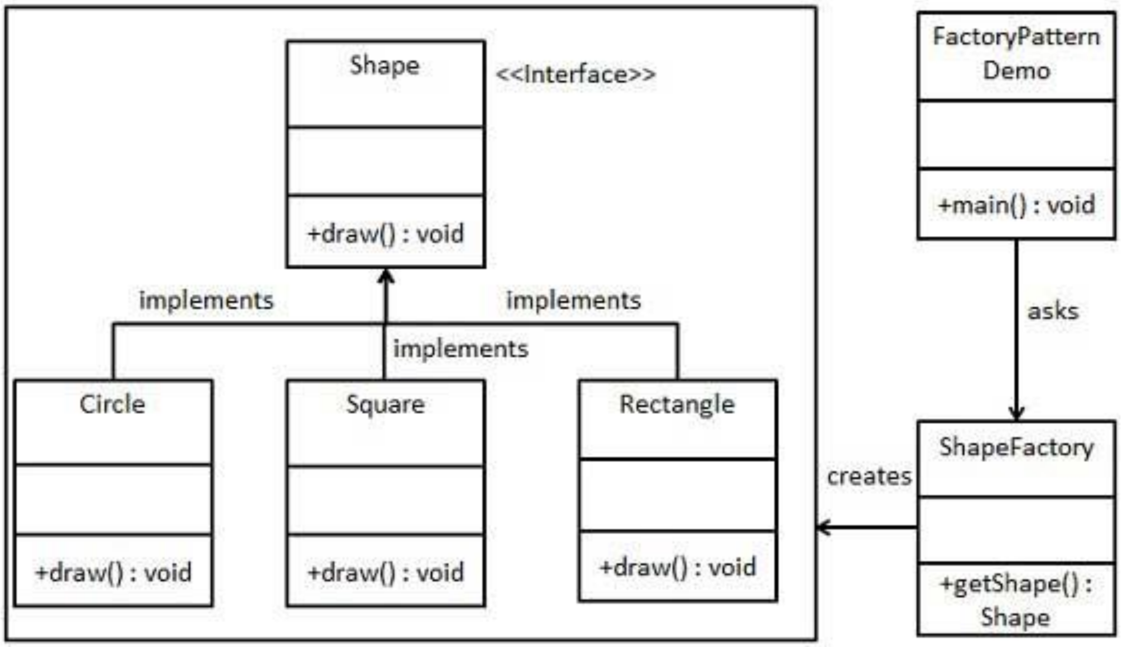
- Design patterns are programming language independent strategies for solving the common object-oriented design problems.
- By using the design patterns you can make your code more flexible, reusable and maintainable.
- It is the most important part because java internally follows design patterns.
- They provide clarity to the system architecture and the possibility of building a better system.
- Categories of Design Patterns
    - Creational
    - Behavioural
    - Structural

1. Creational Design Pattern

- Creational design patterns are concerned with *the way of creating objects*.
- These design patterns provide a way to create objects while hiding the creation logic, rather than instantiating objects directly using new operator.
- This gives program more flexibility in deciding which objects need to be created for a given use case.
- Creational Design Pattern can be further categorized into Class-creational patterns and object-creational patterns.
- While class-creation patterns use inheritance effectively in the instantiation process, object-creation patterns use delegation effectively to get the job done.
- Types of creational design patterns :
    - Factory Method Pattern
    - Abstract Factory Pattern
    - Singleton Pattern
    - Prototype Pattern
    - Builder Pattern
    - Object Pool Pattern

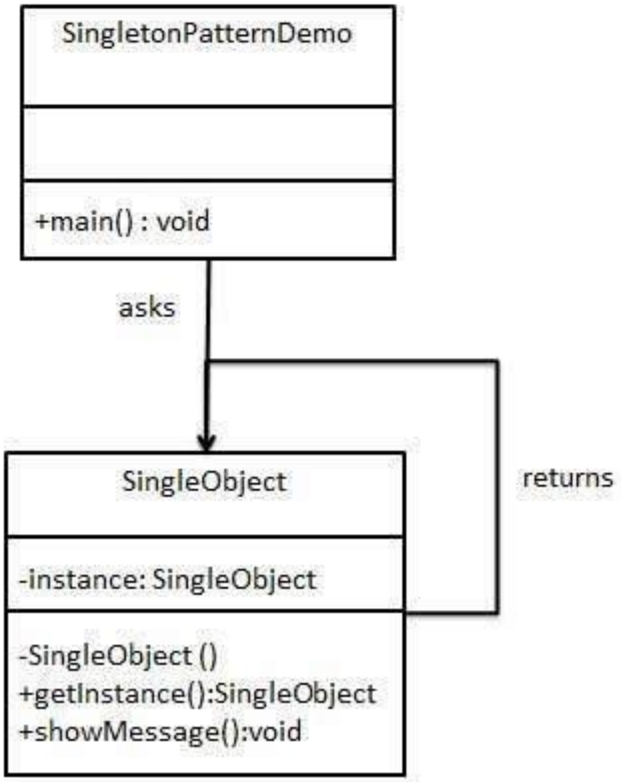
1.1. Factory Method Pattern

- It defines an interface or abstract class for creating an object but lets the subclasses decide which class to instantiate.
- This pattern takes out the responsibility of instantiation of a class from client program to the factory class.
- Super class in factory design pattern can be an interface, abstract class or a normal java class.
- Factory pattern provides abstraction between implementation and client classes through inheritance.
- Eg.



1.2. Singleton Design Pattern

- Singleton pattern restricts the instantiation of a class and ensures that only one instance of the class exists in the java virtual machine.
- The singleton class must provide a global access point to get the instance of the class.
- It uses private non-parameterized constructor to avoid client applications to use constructor.



- Approaches of singleton design pattern

1.2.1. Eager Instantiation:

- The instance of Singleton Class is created at the time of class loading.
- If your singleton class is not using a lot of resources, this is the approach to use.
- But in most of the scenarios, Singleton classes are created for resources such as File System, Database connections etc and we should avoid the instantiation until unless client calls the getInstance method.

1.2.2. Lazy Instantiation:

- The creation of instance is done when required.
- Lazy initialization method to implement Singleton pattern creates the instance in the global access method.
- Lazy initialization works fine in a single threaded environment but when it comes to multithreaded systems, it can cause issues. If multiple threads are inside the if loop at the same time, it will destroy the singleton pattern and both threads will get the different instances of singleton class.

1.2.3. Static Block Initialization:

- Static block initialization implementation is similar to eager initialization, except that instance of class is created in the static block that provides option for exception handling.
- Both eager initialization and static block initialization creates the instance even before it's being used and that is not the best practice to use.

1.2.4. Thread Safe Singleton

- The easier way to create a thread-safe singleton class is to make the global access method synchronized, so that only one thread can execute this method at a time.
- Above implementation works fine and provides thread-safety but it reduces the performance because of cost associated with the synchronized method, although we need it only for the first few threads who might create the separate instances.
- To avoid this extra overhead every time, **double checked locking** principle is used. In this approach, the synchronized block is used inside the if condition with an additional check to ensure that only one instance of singleton class is created.

1.2.5. Bill Pugh Singleton Implementation

- Prior to Java 5, java memory model had a lot of issues and above approaches used to fail in certain scenarios where too many threads try to get the instance of the Singleton class simultaneously even when using synchronization.
- So Bill Pugh came up with a different approach to create the Singleton class using a inner static helper class.

1.2.6. Enum Singleton

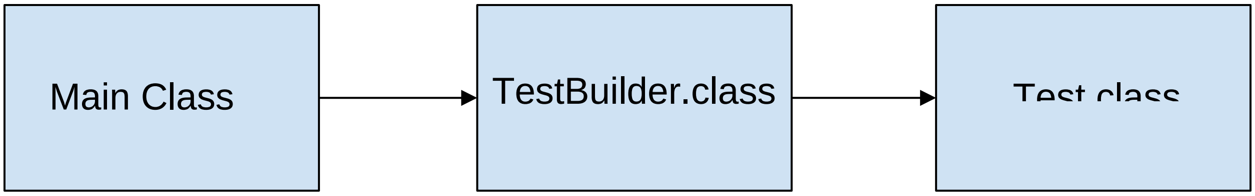
- To overcome the situation with Reflection, Enum is used to implement Singleton design pattern as Java ensures that any enum value is instantiated only once in a Java program.
- Only drawback is that it does not allow lazy initialization.

1.2.7. Serialized Singleton

- Sometimes in distributed systems, we need to implement Serializable interface in Singleton class so that we can store its state in the file system and retrieve it at a later point of time.
- The problem with serialized singleton class is that whenever we deserialize it, it will create a new instance of the class.
- So it destroys the singleton pattern, to overcome this scenario all we need to do provide the implementation of `readResolve()` method.

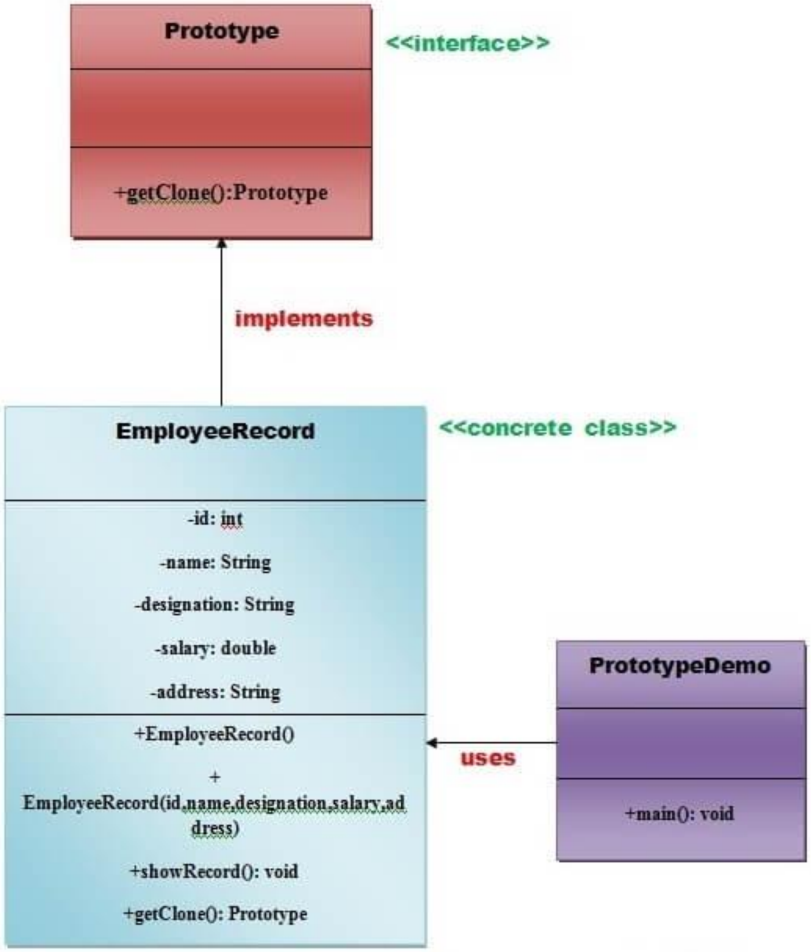
1.3. Builder Design Pattern

- This pattern was introduced to solve some of the problems with Factory and Abstract Factory design patterns when the Object contains a lot of attributes.
- Builder pattern solves the issue with large number of optional parameters and inconsistent state by providing a way to build the object step-by-step and provide a method that will actually return the final Object.



1.4. Prototype Design Pattern

- Prototype Pattern says that cloning of an existing object instead of creating new one and can also be customized as per the requirement.
- This pattern should be followed, if the cost of creating a new object is expensive and resource intensive.



2. Structural Design Pattern

- These design patterns are about organizing different classes and objects to form larger structures and provide new functionality.
- The structural design patterns simplifies the structure by identifying the relationships.
- These patterns focus on, how the classes inherit from each other and how they are composed from other classes.
- Types of structural design patterns :
    - Adapter Pattern
    - Bridge Pattern
    - Composite Pattern
    - Decorator Pattern
    - Facade Pattern
    - Flyweight Pattern
    - Proxy Pattern

2.1. Adapter Pattern

- It’s used so that two unrelated interfaces can work together.
- The object that joins these unrelated interface is called an Adapter.
-