**Creational**

Creational patterns deal with the creation of objects.

**Structural**

* Structural patterns deal with the composition of objects.
* What does a class contain?
* What are the relationships of a class with other classes? Is it inheritance or composition?

**Behavioral**

Behavioral patterns focus more on the behavior of objects, or more precisely, interactions between objects.

**Creational**

**Builder Pattern**

The Builder Pattern separates object construction from its representation.

**Singleton Pattern**

* It allows only one instance of a class per JVM at any point in time.
* The constructor needs to be private, to prevent the possibility of other objects creating instances of your class.
* In Java, build a Singleton using an Enum.
* JEE 7 has a built-in annotation named @Singleton, along with other related annotations.
* Provide the implementation of readResolve() method.

public class A {

private A(){}

private static class SingletonHelper{

private static final A INSTANCE = new A();

}

public static A getInstance(){

return SingletonHelper.INSTANCE;

}

}

**Structural**

**Adapter Pattern**

The object that joins these unrelated interfaces is called an Adapter.

**Decorator Pattern**

Is used to modify the functionality of an object at runtime. At the same time other instances of the same class will not be affected by this.

**Proxy Pattern**

Proxy pattern is used when we want to provide controlled access of a functionality.

**Facade Pattern**

Is used to help client applications to easily interact with the system. Suppose we have an application with a set of interfaces to use MySql/Oracle database and to generate different types of reports, such as HTML report, PDF report, etc.

**Behavioral**

**Template Pattern**

* When we have pre-defined steps or common to achieve some algorithm.
* When we want to avoid duplicating code, moving the common implementation and steps in the base class.

**Mediator Pattern**

The mediator design pattern is used to provide a centralized communication medium between different objects in a system. The mediator design pattern is very helpful in an enterprise application where multiple objects are interacting with each other.

**Strategy pattern**

Is used when we have multiple algorithms for a specific task and the client decides the actual implementation be used at runtime.

**SOLID principles**

* **Single Responsibility** - A class should only have one responsibility.
* **Open/Closed** - Software components should be open for extension, but closed for modification.
* **Liskov Substitution** - Derived types must be completely substitutable for their base types.
* **Interface Segregation** - Clients should not be forced to implement unnecessary methods which they will not use.
* **Dependency Inversion** - Decoupling of software modules.