**Creational Design Pattern:**

**Factory pattern:** It is one of the most used design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.

**Abstract Factory:** Abstract Factory patterns work around a super-factory which creates other factories. This factory is also called as factory of factories. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

In Abstract Factory pattern an interface is responsible for creating a factory of related objects without explicitly specifying their classes. Each generated factory can give the objects as per the Factory pattern.

**Singleton:** This pattern is one of the simplest design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

This pattern involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.

**Prototype:** Prototype pattern refers to creating duplicate object while keeping performance in mind. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

**Structural Design Pattern:**

**Facade pattern:** It hides the complexities of the system and provides an interface to the client using which the client can access the system. This type of design pattern comes under structural pattern as this pattern adds an interface to existing system to hide its complexities.

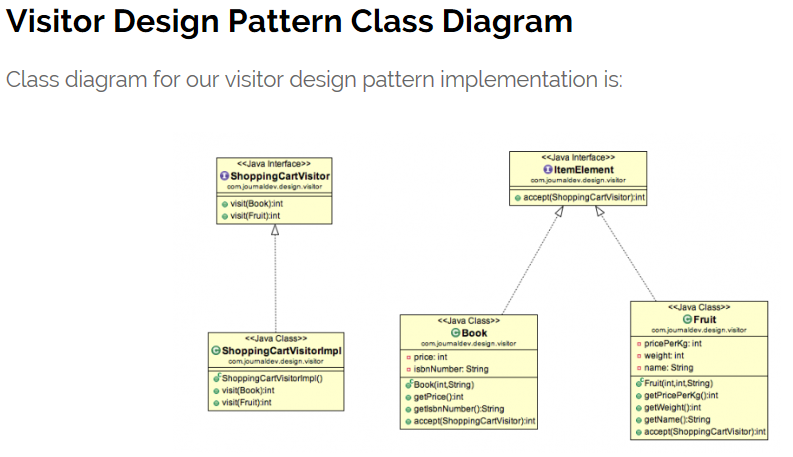
This pattern involves a single class which provides simplified methods required by client and delegates calls to methods of existing system classes.

**Flyweight Design Pattern:** Flyweight pattern is primarily used to reduce the number of objects created and to decrease memory footprint and increase performance. This type of design pattern comes under structural pattern as this pattern provides ways to decrease object count thus improving the object structure of application.

**Behavioral Design Pattern:**

**Observer Design Pattern:** Observer pattern is used when there is one-to-many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically. Observer pattern falls under behavioral pattern category.

**Visitor Design Pattern:** In Visitor pattern, we use a visitor class which changes the executing algorithm of an element class. By this way, execution algorithm of element can vary as and when visitor varies. This pattern comes under behavior pattern category. As per the pattern, element object has to accept the visitor object so that visitor object handles the operation on the element object.



**Template Method Pattern:** Template Method is a behavioral design pattern and it’s used to create a method stub and deferring some of the steps of implementation to the subclasses. Template method defines the steps to execute an algorithm and it can provide default implementation that might be common for all or some of the subclasses.

SOLID Design Principle:

[**Single-responsibility Principle**](https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design#toc-single-responsibility-principle)**:** A class should have one and only one reason to change, meaning that a class should have only one job.

**Open/Closed principle:** Open/Closed principle says that a class should be open for extension but closed for modification. Which means that you can add new features through inheritance but should not change the existing classes (other than bug fixes).

A great example of this in real life is sitting in your pocket in the form of a smartphone. All such phones have app stores and these app stores let you extend the base functionality of the phone. Sure, it ships with the basics: camera operation, actual calls, text messages, etc.

**L is for Liskov Substitution Principle:** Liskovs Substitution Principle states that any method that takes class X as a parameter must be able to work with any subclasses of X.

**I is for Interface Segregation Principle:** The Interface Segregation Principle (ISP) says that you should favor many, smaller, client-specific interfaces over one larger, more monolithic interface.

**D is for Dependency Inversion:** Entities must depend on abstractions not on concretions. It states that the high level module must not depend on the low level module, but they should depend on abstractions.

**Can you share some example of ribbon ?**