1) Bridge And Command Pattern

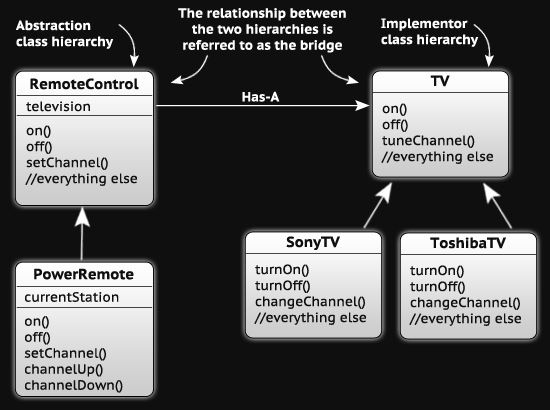
Bridge pattern – The main idea of this pattern, is to decouple the client interface from the implementation so that each of them can change independently.

When the abstract class have several implementations we use inheritance . But, when we use Inheritance it binds an implementation to the abstraction permanently, which makes it difficult to modify, extend, and reuse abstractions and implementations independently. The bridge pattern addresses these problems by putting the abstraction and implementations in separate class hierarchies .

Command pattern -The command pattern is used to decouple the operations of an object class so that their execution can be arranged dynamically which will help understand and maintain the access implementation .By delegating a request to a command object ,the code size of the delegating class is reduced .

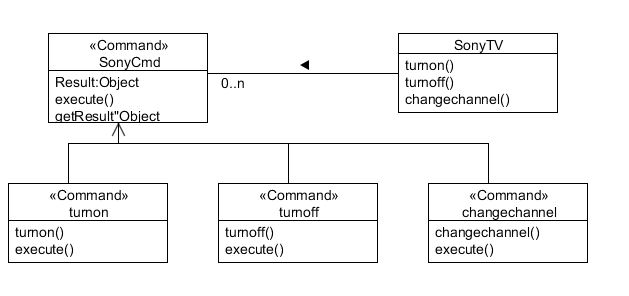
The main difference between command and bridge pattern is that command pattern the concrete command differ in functionality whereas in bridge pattern the concrete implementations differ in implementations not functionality .In bridge ,the client does not create or invoke the Concrete implementation objects but in command ,the client creates and invokes the concrete command objects .

Example of bridge pattern : We take an example of 2 television's which has to be controlled by the remote control .The basic methods or operations are on/off and change channel. The bridge pattern can be efficiently applied to control both the television's as the remote control has a common interface .Here is the pictorial representation



The command pattern cannot be applied as there are not many operations associated with the concreter implementation's SonyTV and ToshibaTV .

Example of command pattern : We consider the example above .If the concrete implementation say SonyTV had many operations the implementation would have many lines of code and would be hard to understand .In such a case applying command patterns would delegate the request to the command objects .The following figure shows the application .



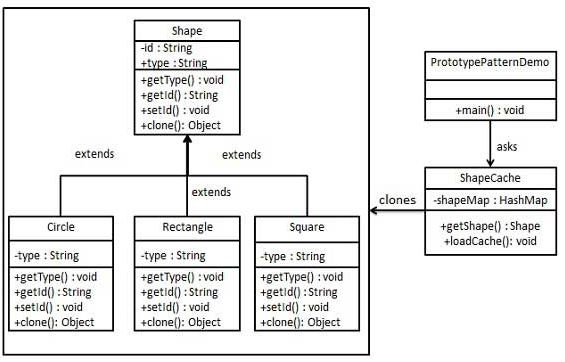
2) Flyweight and prototype pattern

The prototype pattern reduces the number of classes that share similar behavior and relationships .It defines a class to replace all the classes that share similar behavior and relationships .

The flyweight pattern reduces memory requirement due to sharing . It reduces the number of instances that need to be created by sharing the same instances .

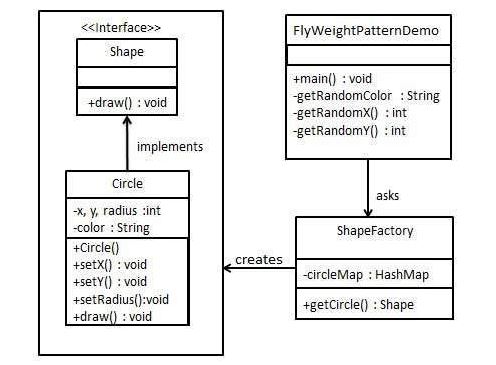
The difference between prototype and flyweight pattern is that flyweight reduces the number of instances of a class to be created ,whereas prototype reduces the number of classes.

Example of prototype pattern : The following is an example of prototype pattern .

The concrete implementations Circle ,Rectangle and Square have similar behavior i.e they have similar methods getType(),getid() and setid() .Hence ,applying prototype patterns we can reduce the number of classes by creating a prototype of the class .

The flyweight pattern is not used in this example as this implementation does not share any common objects .

Example of Flyweight pattern :



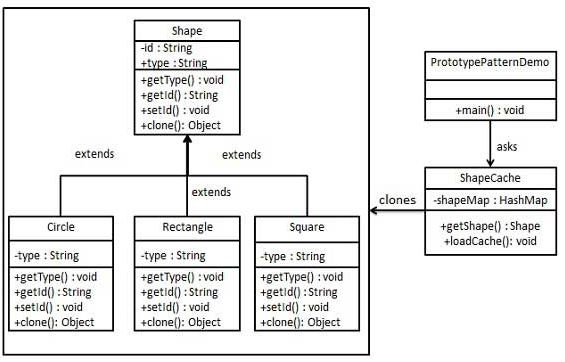
The example given above draws 20 circles at different locations but creates only 5 objects .5 colors are available ,so color property is used to check already existing Circle objects. ShapeFactory has a HashMap of Circle having key as color of the Circle object. Whenever a request comes to create a circle of particular color to ShapeFactory, it checks the circle object in its HashMap, if object of Circle is found, that object is returned otherwise a new object is created, stored in hashmap for future use

3) Singleton and prototype pattern :The prototype pattern reduces the number of classes that share similar behavior and relationships .It defines a class to replace all the classes that share similar behavior and relationships .

The singleton pattern limits the number of globally accessible instances by making the constructor of the class private and defining a public static method to control the creation of the instances .

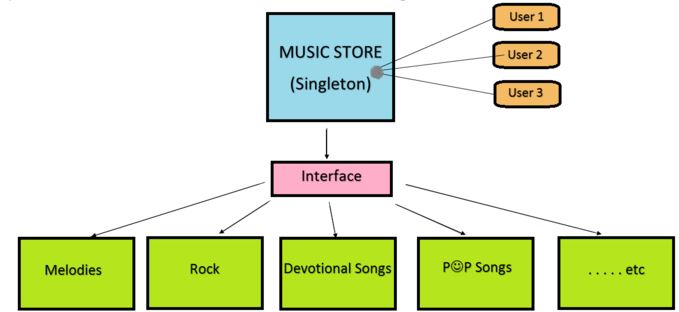
The major difference between singleton and prototype pattern is that singleton limits the number of instances of a class while the prototype pattern reduces the number of classes having similar behavior .

Example of prototype pattern : The following is an example of prototype pattern .

The concrete implementations Circle ,Rectangle and Square have similar behavior i.e they have similar methods getType(),getid() and setid() .Hence ,applying prototype patterns we can reduce the number of classes by creating a prototype of the class .

The singleton pattern cannot be applied as we want to reduce number of classes of Shape ,not limiting the number of instances of the class .

Example of singleton pattern: In the below specified example we want to keep only one instance of music store where all user can access the music via this implementation .Hence this supports global access to instances .



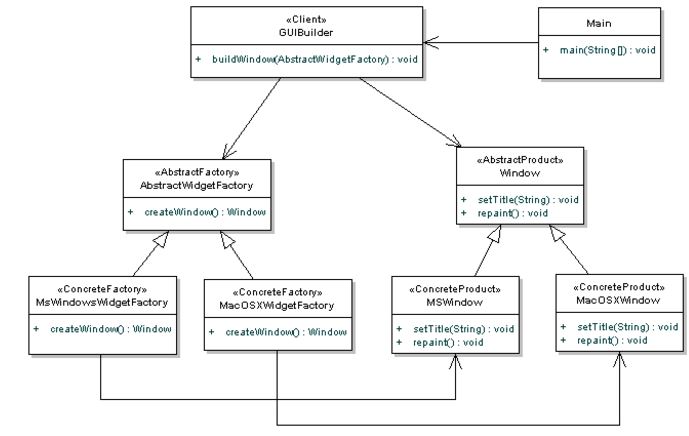
Applying prototype pattern is not useful because we don't want to reduce the number of classes but provide global access to one instance .

4)Abstract factory and factory pattern

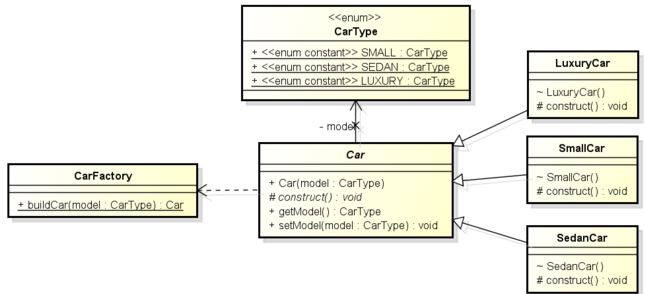
Factory Method : The Factory Method pattern is used when there is a need to decouple a client from a particular product that it uses. The Factory Method to relieves a client of responsibility for creating and configuring instances of a product.

Abstract factory : The abstract factory pattern creates objects of different families and hides from the client which family of objects is created .

The methods of an Abstract Factory are implemented as Factory Methods. Both the Abstract Factory Pattern and the Factory Method Pattern decouples the client system from the actual implementation classes through the abstract types and factories. The Factory Method creates objects through inheritance where the Abstract Factory creates objects through composition.

Example of abstract factory :We consider an example of an Abstract Factory used in UI toolkits. Widgets are provided for all windows platforms . The implementation of these widgets vary across platforms. We can write a platform independent client using the Abstract Factory implementation.  

Example of factory method : Below diagram depicts a common scenario using example of car factory which is able to build 3 types of cars i.e. small, sedan and luxury. Hence factory pattern introduces loose coupling between classes .



5) Adapter and proxy patterns :

Adapter pattern :The adapter pattern converts one interface to another .It defines an interface to match the one expected and let the implementing subclass adapt it to the existing interface .

Proxy pattern : A proxy exposes the exact same behavior as the object it hides. A proxy is typically used to contact a remote object without having to know how to contact it.

The main difference between adapter and proxy pattern is that the primary purpose of the adapter pattern is to change the interface of class/library A to the expectations of client B. The typical implementation is a wrapper class or set of classes. The purpose is to facilitate current interface incompatibilities. The proxy pattern also uses wrapper classes, but for a different purpose. The purpose of the proxy pattern is to create a stand-in for a real resource .

Example of proxy pattern:

https://dzone.com/articles/design-patterns-abstract-factory

http://www.tutorialspoint.com/design\_pattern/abstract\_factory\_pattern.htm

http://howtodoinjava.com/2012/10/23/implementing-factory-design-pattern-in-java/

http://programmers.stackexchange.com/questions/201397/difference-between-the-adapter-pattern-and-the-proxy-pattern

http://stackoverflow.com/questions/3766764/differences-between-proxy-pattern-and-adapter-pattern