Creational - Prototype Design Pattern - Tüm Örnekleri incele Ağır bir database nesnesini çağırıp bir kopyasını deep copy olarak çıkardıktan sonra üzerinde çalışmak maliyeti düşürecek ve performans sağlayacaktır. Prototype design pattern bu problemi çözmek için vardır,initialize masraflarını minimize eder.Bir diğer örnekte excelde aylık bir rapor hazırlandığını düşünelim, her ay raporu baştanmı yaratıyorsunuz yoksa diğer ay kullandığınız excel i copy paste edip verilerimi değiştiriyorsunuz.

Örnek 1;

Cloneable interface'inden türetilen bir abstract class create ediyoruz,içerisinde colorName diye bir property ve fillColor adında bir method barındırıyor.Clone methodu ise gelen objenin clone'unu oluşturuyor;

```
abstract class ColorPrototype implements Cloneable{
   protected String colorName;
   abstract void fillColor();

   @Override
   protected Object clone() {
      Object clone = null;
      try{
         clone = super.clone();
      }catch (CloneNotSupportedException e){
         e.printStackTrace();
      }
      return clone;
   }
}
```

YellowColor adında bir class ColorPrototype'ımızı inherit ediyor,aynı şekilde değişik renklerde eklenebilir;

```
class YellowColor extends ColorPrototype{
   public YellowColor() {
      this.colorName = "Yellow";
   }
   @Override
   void fillColor() {
      System.out.println("filling yellow color");
   }
}
```

Prototype class'ımızı yaratacak bir factory create ediyoruz.Bu class maliyeti yüksek olan class'larımızı newleme görevini üstlenen class'ımız;

```
class ColorPrototypeFactory{
   private static HashMap<String,ColorPrototype> colorMap = new HashMap<>();
   static{
      colorMap.put("Yellow",new YellowColor());
      colorMap.put("Red",new RedColor());
   }
   public static ColorPrototype getColor(String colorName){
      return colorMap.get(colorName);
   }
}
```

Main içerisinde yellow'u create ettiğimiz anda Red clone olarak hiç uğraşılmadan direk create edilmiş olacak

```
class main{
   public static void main(String[] args) {
      ColorPrototypeFactory.getColor("Yellow").fillColor();
      ColorPrototypeFactory.getColor("Red").fillColor();
   }
}
```

Creational - Prototype Design Pattern - Örnek 2

```
abstract class Prototype implements Cloneable{
   @Override
   protected Object clone() throws CloneNotSupportedException {
      Object clone = null;
      try{
         clone = super.clone();
      }catch (CloneNotSupportedException exception){
         exception.printStackTrace();
      }
      return clone;
   }
}
```

Class ;

```
class Movie extends Prototype{
   private String name = null;
   @Override
   protected Object clone() throws CloneNotSupportedException {
      System.out.println("Cloning Movie ...");
      return super.clone();
   }
}


class Album extends Prototype{
   private String name;
   @Override
   protected Object clone() throws CloneNotSupportedException {
      System.out.println("Cloning Album ...");
      return super.clone();
   }
}
```

Factory;

```
class PrototypeFactory{
   public static class ModelType{
      public static final String MOVIE = "Movie";
      public static final String ALBUM = "Album";
   }
   private static HashMap<String,Prototype> type = new HashMap<>();
   static{
      type.put(ModelType.MOVIE,new Movie());
      type.put(ModelType.ALBUM,new Album());
   }

   public static Prototype getInstance(final String s) throws CloneNotSupportedException {
      return (Prototype) type.get(s).clone();
   }
}
```

Main;

```
class Test{
   public static void main(String[] args) throws CloneNotSupportedException {
      String test = PrototypeFactory.getInstance("Movie").toString();
      String test2 = PrototypeFactory.getInstance("Album").toString();
      System.out.println(test);
      System.out.println(test2);
   }
}
```

Creational - Prototype Design Pattern Örnek 3;

```
interface Animal extends Cloneable{
   public Animal clone();
}
```

Concrete Class

```
class Sheep implements Animal{
   public Sheep(){
      System.out.println("Sheep is made");
   }
   @Override
   public Animal clone() {
      Sheep object = null;
      try{
         object = (Sheep) super.clone();
      } catch (CloneNotSupportedException e) {
         e.printStackTrace();
      }
      return object;
   }
}
```

Factory

```
class CloneFactory{
   private static HashMap<String,Animal> animalHashMap = new HashMap<>();
   static{
      animalHashMap.put("Sheep",new Sheep());
   }
   public static Animal getInstance(final String s){
      return animalHashMap.get(s);
   }
}
```

Main

```
class main{
   public static void main(String[] args) {
      Sheep test = (Sheep) CloneFactory.getInstance("Sheep");
      System.out.println(test.getClass());
   }
}
```