Creational - Factory Design Pattern

Oluşturduğumuz bir interface ya da abstract sınıftan türeterek başka bir sınıf oluşturma işlemine verilen addır Factory Pattern. Araba'nın özelliklerini bildirecek bir yazılım yapacağız. Kullanıcı araba seçecek ancak seçilen arabanın hatchback mi yoksa sedan özellikli olacağını bilmiyoruz ve bu değişkenlik gösterebilir. Kullanıcının seçimine bırakılan bu işlemde kullanıcının her seçiminde kod değişikliği yapmamak için Factory Pattern'e başvurulur. Yada şöyle bir şey düşünelim. Loglama yapan bir interface'imiz ve altında concrete class'ları var. Hangi loglama mekanizmasını seçersek fabrika onu üretecek.

//Logic'e Sahip class'ların inherit edileceği interface

```
interface IServer{
   //belli bir sunucu için ağ ile ilgili sorunları giderir
   public void resolve();
}
```

Concrete Class'lar

```
class MailServer implements IServer{
   @Override
   public void resolve() {
      System.out.println("Mail server üzerinde ki problem giderildi");
   }
}

class FtpServer implements IServer{
   @Override
   public void resolve() {
      System.out.println("Ftp Server üzerinde ki problem giderildi");
   }
}
```

Gelen string değere göre dynamicly üretim yapan fabrikamız

```
class ServerFactory{
   public static IServer getServer(String serverType) throws Exception {
      switch(serverType){
         case "mail" : return new MailServer();
         case "ftp" : return new FtpServer();
         default: throw new Exception("Invalid server type");
      }
   }
}
```

Main

```
public static void main(String[] args) throws Exception {
   Scanner input = new Scanner(System.in);
   System.out.println("Which server do you want to resolve");
   String serverResult = input.nextLine();
   IServer server = ServerFactory.getServer(serverResult);
   server.resolve();
}
```

Problemli olan server'ı kullanıcığın gireceği string değere göre gidip resetleyecek basit bir kod örneği