Behavioral - Template Method Design Pattern (2. Örneğe mutlaka bakılmalı)

Strategy design patterni davranışın tamamen değiştiği durumlarda, Template design pattern ise bir kısmı değiştiği durumlarda tercih etmekteyiz.

Örnekte **HotDrink** parent class'ı içerisinde bir **doIt**() adlı template metodu hazırlanmış, **Tea** ve **nescafe** içecekleri için gerekli yerlerde değişiklik yaparak kullanılmıştır. Bu template içinde tanımlanmış **heatWater** ve **fillTheGlass** fonksiyonlarının tüm içecekler için aynı olması nedeniyle bu fonksiyonlar geçersiz kılınması isteğe bağlı olacak şekilde şekilde tanımlanmıştır. **Contents** methodu ise **Tea** ve **nescafe** için farklılık göstereceğinden bu fonksiyon tanımlanması zorunlu olacak şekilde **abstract** olarak tanımlanmıştır. **Tea** ve **Nescafe** alt sınıflarında **Content** farklı olarak tanımlanmış ve daha sonra **doIt**() metodu çağırılarak program çıktıları elde edilmiştir. Bu durumda programa yeni bir içecek eklenmesi gerektiğinde bütün kodları kopyalamak yerine yeni içecek sınıfını **HotDrink** parent classından türetip **Contents** metodunu tanımlamak yeterli olur. Böylece hem kod tekrarının önüne geçilmiş hem de sade ve anlaşılır bir kod yazılmış olur.

```
abstract class HotDrink{
   void heatWater(){
      System.out.println("Su isitildi");
   }
   //Primitive operation
   abstract void contents(); //içerik

   void fillTheGlass(){
      System.out.println("Icecek bardaga dolduruldu");
   }
   //Template method
   final void doIt(){
      heatWater();
      contents();
      fillTheGlass();
      System.out.println("Icecek hazırlandı");
   }
}
```

Concrete class'lar

```
class Tea extends HotDrink{
   @Override
   void contents() {
      System.out.println("Cay hazirlandi");
   }
}

class Nescafe extends HotDrink{
   @Override
   void contents() {
      System.out.println("Nescafe hazirlandi");
   }
}
```

Main

```
public static void main(String[] args) {
   HotDrink tea = new Tea();
   tea.doIt();
   HotDrink nescafe = new Nescafe();
   nescafe.doIt();
}
```

Behavioral - Template Method Design Pattern (2.Örnek)

Template method design pattern'i bir algoritmayı yürütmek için adımları tanımlar ve alt sınıfların tümü veya bazıları için ortak olabilecek varsayılan implementasyonları sağlayabilir. Bu deseni bir örnekle anlayalım, bir ev inşa etmek için bir algoritma sağlamak istediğimizi varsayalım. Bir ev inşa etmek için adımlar atılmalıdır - bina temeli, bina direkleri, bina duvarları ve pencereler. Önemli olan nokta, yürütme sırasını değiştirememiz çünkü temeli inşa etmeden önce pencereleri inşa edemiyoruz. Yani bu durumda biz ev inşa etmek için farklı yöntemler kullanacak bir template method yöntemi oluşturabiliriz.Alt class'ların template methodunu geçersiz kılmamaları için mutlaka method imzasını **final** olarak işaretlemeliyiz

```
abstract class HouseTemplate{
   //template method
   public final void buildHouse(){
      buildFoundation(); //temel
      buildPillars(); //sütunlar
      buildWalls(); //duvarlar
      buildWindows(); //pencereler
      System.out.println("House is built");
   }
   //default implementation
   private void buildWindows(){
      System.out.println("Building glass windows");
   }
   private void buildFoundation(){
      //Çimento,demir çubuklar ve kum ile temel oluşturma
      System.out.println("Building foundation with cement,iron rods and sand");
   }
   //methods to be implemented by subclasses
   public abstract void buildWalls();
   public abstract void buildPillars();
}
```

Concrete Classes;

```
class WoodenHouse extends HouseTemplate{
   @Override
   public void buildWalls() {
      System.out.println("Building wooden walls");
   }
   @Override
   public void buildPillars() {
      System.out.println("Building Pillars with Wood coating");
   }
}
class GlassHouse extends HouseTemplate{
   @Override
   public void buildWalls() {
      System.out.println("Building glass walls");
   }
   @Override
   public void buildPillars() {
      System.out.println("Building Pillars with glass coating");
   }
}
```

Main;

```
public static void main(String[] args) {
   HouseTemplate woodenHouse = new WoodenHouse();
   woodenHouse.buildHouse();
}
```