**Decorator Pattern (Bezeme Şablonu)**

**Tür:** *Yapısal Şablon*

**Diğer Adı:** *Sarmalayıcı (Wrapper)*

Adında anlaşılacağı gibi bezeme şablonu ilişkilendirildiği (bu manada Ingilizce '*attach*' fiili kullanılır) sınıfı dinamik olarak dekore eder. Bezeme şablonuna ayrıca sarmalıyıcı (*Wrapper*) da denir. Bu isimle daha çok karşılaşmak mümkündür çünkü Java ve C# dilinin kendi içinde doğal türlerin (*int, float, double* gibi) birer *Wrapper* sınıfları bulunur. Buradaki konsept her ne kadar farklı da olsa gerçek hayatta bir şablonun zaten tek ve nihayi bir hali yoktur.

Bezeme şablonuna alternatif olarak sınıf türetme düşünülebilir. Fakat bu durumda bir sınıfın alacağı durum önceden belirlenmiş (statik olarak) ve sınırlandırılmış olur. Bu genellikle karşılaşılan bir durumdur fakat bazı durumlar vardır ki dinamik olarak bezemek daha çok işimize gelebilir. Örneğin bir sınıfımıza ekleyebileceğimiz n tane yeteneğimiz (*capability*) olsun. Bu yeteneklerin bir yada bir kaçını destekleyen alt sınıflar elde etmek istersek n'in bütün kombinasyonlarını toplamamız gerekir. Fakat bu işi bezeme sınıfları ile yaparsak sadece n tane bezeme sınıfını türetmek yeterli olur.

Bezeme şablonu oldukça sık karşılaşılan bir şablondur. Son zamanlarda moda olan AOP (*Aspect Oriented Programming*) özellikle bezeme şablonu üzerine bina olmuştur. AOP'ta Interceptor denilen nesneler kendisi için özel olarak bezenmiş sınıflara sonradan raporlama gibi iş hayatında önemli özellikler katabilir.

Bezeme şablonu en çok katmanlı yapılarda kullanılır. Her sınıf bir katman teşkil eder. Katmanlı yapılara da en iyi örnek görsel uygulamalardır (*GUI App - Graphic User Interface Application*). İlk önce boş ekranı temsil eden bir sınıf olur. Bu sınıf bile tek başına görüntü oluşturabilir. Fakat çeşitli kütüphanelerde (*.NET, Java Swift, Qt* gibi) farklı olmakla birlikte çok benzer katmanlı görsel sınıflar vardır. Bu sınıflar hepsi aynı sınıftan türerler. Hepsinin kendini çizdirme metodları vardır. İstenilen çizdirme özelliği için var olan görsel nesne bu yeteneği temsil eden sınıf tarafından sarmalanır.



GOF Decorator Kalıbı

**Örnek**

Ev eşyaları üzerine uzmanlaşmış bir firma, bir fabrikadan hammade halinde ayna satın alıp, bunları çeşitli ürünler haline getirdikten sonra pazarlamaktadır. Ayna sınıfı üzerinde bir değişiklik yapılması istenmemektedir. Daha ziyade, değişik aynalı ürünleri modellemek için yeni sınıfların oluşturulması istenmektedir. Pazarlama firması ilk etapta değişik çerçevelere sahip aynaları piyasaya sürmeyi planladığından, hazırlayacağı program buna cevap vermelidir.



|  |
| --- |
| public interface EvEsyalari  {  /\*\*  \* Üretimi gerceklestirmek icin  \* kullanilan metod.  \*/  public void produce();  } |

|  |
| --- |
| public class Ayna implements EvEsyalari  {  public void produce()  {  System.out.println("Ayna imal edildi.");  }  } |

|  |
| --- |
| public abstract class AynaDecorator implements EvEsyalari  {  /\*  \* Bünyesinde mevcut bir ayna nesnesi  \* bulundurur ve degisik metodlar kullanarak  \* bu ayna nesnesini dekore eder.  \*/  private EvEsyalari ayna = new Ayna();  public EvEsyalari getAyna()  {  return ayna;  }  public void setAyna(EvEsyalari ayna)  {  this.ayna = ayna; |

|  |
| --- |
| public class CerceveliAyna extends AynaDecorator  {  /\*\*  \* Üretim icin kullanilan sinif.  \* addBorder metodu ile  \* aynaya cerceve ekler.  \*/  public void produce()  {  getAyna().produce();  addBorder();  }    /\*\*  \* Cerceve ekleme islemini gerceklestirmek  \* icin kullanilan metod.  \*  \*/  public void addBorder()  {  System.out.println("Aynaya cerceve eklendi.");  }  } |

|  |
| --- |
| public class Test  {  public static void main(String[] args)  {  EvEsyalari ayna = new CerceveliAyna();  ayna.produce();  }  } |

|  |
| --- |
| **package** cream;  **public** **interface** Icecream {  **public** String makeIcecream();  } |

|  |
| --- |
| **package** cream;  **abstract** **class** IcecreamDecorator **implements** Icecream {  **protected** Icecream specialIcecream;  **public** IcecreamDecorator(Icecream specialIcecream) {  **this**.specialIcecream = specialIcecream;  }  **public** String makeIcecream() {  **return** specialIcecream.makeIcecream();  }  } |

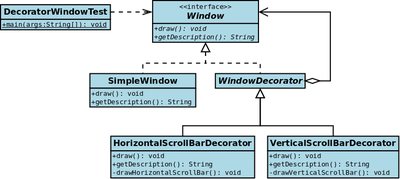
|  |
| --- |
| **package** cream;  **public** **class** SimpleIcecream **implements** Icecream {  @Override  **public** String makeIcecream() {  **return** "Base Icecream";  }  } |

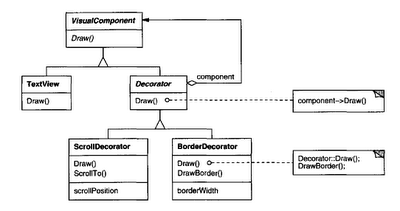
|  |
| --- |
| **package** cream;  **public** **class** HoneyDecorator **extends** IcecreamDecorator {  **public** HoneyDecorator(Icecream specialIcecream) {  **super**(specialIcecream);  }  **public** String makeIcecream() {  **return** specialIcecream.makeIcecream() + addHoney();  }  **private** String addHoney() {  **return** " + sweet honey";  }  } |

|  |
| --- |
| **package** cream;  **public** **class** NuttyDecorator **extends** IcecreamDecorator {  **public** NuttyDecorator(Icecream specialIcecream) {  **super**(specialIcecream);  }  **public** String makeIcecream() {  **return** specialIcecream.makeIcecream() + addNuts();  }  **private** String addNuts() {  **return** " + cruncy nuts";  }  } |

|  |
| --- |
| **package** cream;  **public** **class** NuttyNuttyDecorator **extends** IcecreamDecorator {  **public** NuttyNuttyDecorator(Icecream specialIcecream) {  **super**(specialIcecream);  }  **public** String makeIcecream() {  **return** specialIcecream.makeIcecream() + addNutts();  }  **private** String addNutts() {  **return** " + cruncy nuts nuts";  }  } |

|  |
| --- |
| **import** cream.\*;  **public** **class** TestIcecream {  **public** **static** **void** main(String args[]) {  Icecream icecream = **new** HoneyDecorator(**new** NuttyDecorator(**new** NuttyNuttyDecorator(  **new** SimpleIcecream())));  System.*out*.println(icecream.makeIcecream());  }  } |



[](http://4.bp.blogspot.com/-4HH4zu7jxQQ/Tm-We_UteRI/AAAAAAAAADg/gDvHaqmyyAg/s1600/GoF_decorator.png)

Burada VisualComponent ana sınıfımızdır. Genellikle soyut sınıf (abstract class) olur. TextView sınıfı sadece verilen yazıyı göstermekle yükümlüdür. Diğer Decorator'dan türeyen ScrollDecorator ve BorderDecorator sınıfları TextView sınıfını isteğe bağlı olarak bezer. İster biri, ikisi yada hiçbiri kullanılmayabilir.

Çünkü tüm sınıflar bir birinden bağımsızdırlar.

Ayrıca Java'daki dosya işleme sınıfları tamamen bezeme sınıfları üzerine kurulmuştur.

|  |
| --- |
| **package** iodecor;  **import** java.io.FilterInputStream;  **import** java.io.IOException;  **import** java.io.InputStream;  **public** **class** LowerCaseInputStream **extends** FilterInputStream {  **public** LowerCaseInputStream(InputStream in) {  **super**(in);  }  **public** **int** read() **throws** IOException {  **int** c = **super**.read();  **return** (c == -1 ? c : Character.*toLowerCase*((**char**) c));  }  **public** **int** read(**byte**[] b, **int** offset, **int** len) **throws** IOException {  **int** result = **super**.read(b, offset, len);  **for** (**int** i = offset; i < offset + result; i++) {  b[i] = (**byte**) Character.*toLowerCase*((**char**) b[i]);  }  **return** result;  }  } |

|  |
| --- |
| **package** iodecor;  **import** java.io.BufferedInputStream;  **import** java.io.FileInputStream;  **import** java.io.FileNotFoundException;  **import** java.io.IOException;  **public** **class** Test {  /\*\*  \* **@param** args  \*/  **public** **static** **void** main(String[] args) {  **try** {  FileInputStream fi=**new** FileInputStream("oku.txt");  LowerCaseInputStream buf = **new** LowerCaseInputStream(**new** BufferedInputStream(fi));  **int** c;  **while**((c=buf.read())>0)  System.*out*.print((**char**)c);  } **catch** (FileNotFoundException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }    }  } |

Merhaba Dünya --o merhaba dünya

|  |
| --- |
| Scanner sc = **new** Scanner(buf);  **while** (sc.hasNextLine())  System.*out*.println(sc.nextLine()); |

Bu örnekte InputStream sinifi Component, FileInputStream sinifi ConcreteComponent,

FilterInputStream sinifi Decorator, LowerCaseInputSTream ve BufferedInputSTrean siniflari

ConcreteDecorator sinifi oldular.

Sonrada eklenen Scanner veya DataInputSTream ile okumada ise Strategy tasarim kalibi vardir.