JAVA

Tag-28 03.05.2018

***Import sales.\**** yazildiginda sadece sales altindaki klaslari kapsar. Alt daldaki package leri kapsamaz

Wir haben swing kennen gelernt.

**Fabrik design Pattern:**

Bu dersimizde, yazdığımız kodların daha iyi olmasını sağlayacak bir tasarım desenini işleyeceğiz. Adı “Fabrika tasarım deseni”. Fabrika denmesinin sebebi, bir sınıfın, kendi içlerinde benzer kalıplarda olan java sınıflarından üretiyor olmasıdır.

Örneğin bir araba fabrikamız olsun fakat farklı modellerde araçları üretsin. Reno, Bmw ve Mercedes gibi. Bu araçların hepsinin bir “MARKASI” vardır, bir “FİYATI” vardır öyle değil mi? Bu iki temel özellikle, şablon haline getirilip, üretilen araçların kullanılmasını kolaylaştıracak bir desen oluşturacağız birlikte.

Önce araçların ortak kalıbı olan şablonu yani arayüzü oluşturalım. Adı “Araba” olacak, aşağıdaki gibi.

2. package com.umitsamimi.fabrikadeseni;
4. public interface Araba {
5. public String getMarka();
6. public String getFiyat();
7. }

Arayüz sınıfımızda gördüğünüz gibi, iki method(yordam) bulunmakta. Bu iki yordam, kendisinden implemente edilen tüm sınıflarda olmak zorundadır ve o sınıfların, Araba arayüzüne göre şekillenmesini sağlayacaktır.

Şimdi Araba Arayüz’ünden implemente edilen sınıflara geçelim. İlk,

2. package com.umitsamimi.fabrikadeseni;
4. public class Renault implements Araba {
6. @Override
7. public String getMarka() {
8. return "Bu aracin markasi Reno";
9. }
11. @Override
12. public String getFiyat() {
13. return "X lira";
14. }
16. }

Gördüğünüz gibi Renault araci, bir markaya ve bir fiyata sahip. Bu sebeple Araba Arayüz’ünden implemente edildi. Aynı şekilde

2. package com.umitsamimi.fabrikadeseni;
4. public class Bmw implements Araba {
6. @Override
7. public String getMarka() {
8. return "Bu aracin markasi BMW";
9. }
11. @Override
12. public String getFiyat() {
13. return "Y lira";
14. }
16. }

Bmw aracı da Araba Arayüz’ünden implemente edildi. Son olarak,

2. package com.umitsamimi.fabrikadeseni;
4. public class Mercedes implements Araba {
6. @Override
7. public String getMarka() {
8. return "Bu aracin markasi Mercedes";
9. }
11. @Override
12. public String getFiyat() {
13. return "Z lira";
14. }
16. }

Tüm araçların tanıtımını yaptık, şimdi sırada bu araçların üretildiği fabrikayı oluşturmakta.

2. package com.umitsamimi.fabrikadeseni;
4. public class ArabaFabrikasi {
6. public static Araba arabaGetir(String arabaMarkasi)
7. {
8. if("Renault".equals(arabaMarkasi))
9. {
10. return new Renault();
11. }
12. else if("Bmw".equals(arabaMarkasi))
13. {
14. return new Bmw();
15. }
16. else if("Mercedes".equals(arabaMarkasi))
17. {
18. return new Mercedes();
19. }
20. return null;
21. }
22. }

Özellikle arabaGetir yordamını static tanımladık. Böylelikle, her defasında ArabaFabrikasi sınıfını oluşturmanıza gerek olmadan, direk yordam ile bir arac üretebileceksiniz. Bu yordam gördüğünüz gibi geriye “Araba” sınıfını dönüyor. Modelini vermeniz yeterli.

Şimdi de sırada ArabaFabrikasının çağrılıp, üretilen aracı kullanacağımız ana sınıfı tanımlayalım..

1. package com.umitsamimi.fabrikadeseni;
3. public class Istemci {
5. public static void main(String[] args) {
6. Araba reno = ArabaFabrikasi.arabaGetir("Renault");
7. System.out.println("Reno arac fiyati " + reno.getFiyat());
9. Araba bmw = ArabaFabrikasi.arabaGetir("Bmw");
10. System.out.println("Bmw arac fiyati "+ bmw.getFiyat());
11. }
12. }

|  |
| --- |
| package Factory\_Design\_Pattern\_4;    2. public class Araba { 3. private String marka; 4. private String model; 5. private double fiyat; 6. public Araba(String marka, String model, double fiyat){ 7. this.marka=marka; 8. this.model=model; 9. this.fiyat=fiyat; 10. } 12. public String getMarka() { 13. return marka; 14. } 16. public void setMarka(String marka) { 17. this.marka = marka; 18. } 20. public String getModel() { 21. return model; 22. } 24. public void setModel(String model) { 25. this.model = model; 26. } 28. public double getFiyat() { 29. return fiyat; 30. } 32. public void setFiyat(double fiyat) { 33. this.fiyat = fiyat; 34. } 36. } |

Yukarıda ilk olarak Araba sınıfımızı oluşturarak içinde yer alacak bazı özellikleri ve metotları belirledik.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | package Factory\_Design\_Pattern\_4;    public class BmwZ4 extends Araba {    public BmwZ4(double fiyat) {  super("BMW", "Z4", fiyat);  }    } |

Yukarıda ki sınıfta ise Araba sınıfı BmwZ4 sınıfını türetti ve yapıcı metodunda sadece fiyat parametresini aldı.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | package Factory\_Design\_Pattern\_4;    public class Nissan350z extends Araba {    public Nissan350z( double fiyat) {  super("Nissan", "350z", fiyat);  }    } |

BmwZ4 için yapılan işlemlerin aynısını Nissan350z sınıfı içinde yapıldı.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | package Factory\_Design\_Pattern\_4;    import java.util.ArrayList;  import java.util.List;    public abstract class ArabaFactory {  public List<Araba> arabaListesi=new ArrayList<Araba>();  public ArabaFactory(){  createAraba();  }  public abstract void createAraba();  public List<Araba> getArabaListesi() {  return arabaListesi;  }    public void setArabaListesi(List<Araba> arabaListesi) {  this.arabaListesi = arabaListesi;  }    } |

ArabaFactory sınıfımızı abstract class olarak belirledik. Böylece bu sınıfı kalıtım olarak alan diğer sınıflar bu sınıfın hazır metotlarını kullanma ve createAraba metotunu kendilerine uyarlayarak yazma imkanına kavuşacaklar. arabaListesi değişkeni ile üretilen arabaların listesini tutmayı sağladık. Sınıfın yapıcı metotunda createAraba() fonksiyonunu kullanıdk ki bu sınıfa ait veya alt sınıfına ait bir nesne üretildiğinde otomatik olarak creteAraba() metotu devreye girsin.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | package Factory\_Design\_Pattern\_4;    public class BmwFactory extends ArabaFactory {    @Override  public void createAraba() {  getArabaListesi().add(new BmwZ4(120000));  }    } |

BmwFactory sınıfı ArabaFactory sınıfını kalıtım olarak alarak createArabasa metodunu ezerek kendisine uygun biçimde kodları yazdı. Burada dikkat edin createAraba metodu içersinde araba listesine BmwZ4 sınıfı üretilerek listeye eklendi. Yeni modeller oluştuğunda bu listeye eklememiz yeterli olacak.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | package Factory\_Design\_Pattern\_4;    public class NissanFactory extends ArabaFactory {    @Override  public void createAraba() {  getArabaListesi().add(new Nissan350z(140000));  }    } |

BmwFactory için geçerli olan kodlar NissanFactory içinde geçerli. Burada da Nissan350z sınıfı listeye eklendi.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | package Factory\_Design\_Pattern\_4;    public class Test {    public static void main(String[] args) {  ArabaFactory bmwArabalar=new BmwFactory();  ArabaFactory nissanArabalar=new NissanFactory();  for(Araba  araba: bmwArabalar.getArabaListesi()){  System.out.println(araba.getMarka() + " - " + araba.getModel() + " - " + araba.getFiyat());  }  for(Araba  araba: nissanArabalar.getArabaListesi()){  System.out.println(araba.getMarka() + " - " + araba.getModel() + " - " + araba.getFiyat());  }    }    } |

ArabaFactory sınıfı alt sınıfları olan BmwFactory ve NissanFactory alt sınıflarının nesnesi üretildi. Üretilen bu nesneler ile daha önce fabrikalarında üretilmesi için tanımlanmış olan araba modelleri for ile tek tek ekrana aşağıda ki gibi yazdırıldı.

factory-design-pattern-araba