1906003172019

Tasarım Desenleri

Dr. Öğr. Üy. Önder EYECİOĞLU
Bilgisayar Mühendisliği

Giriş

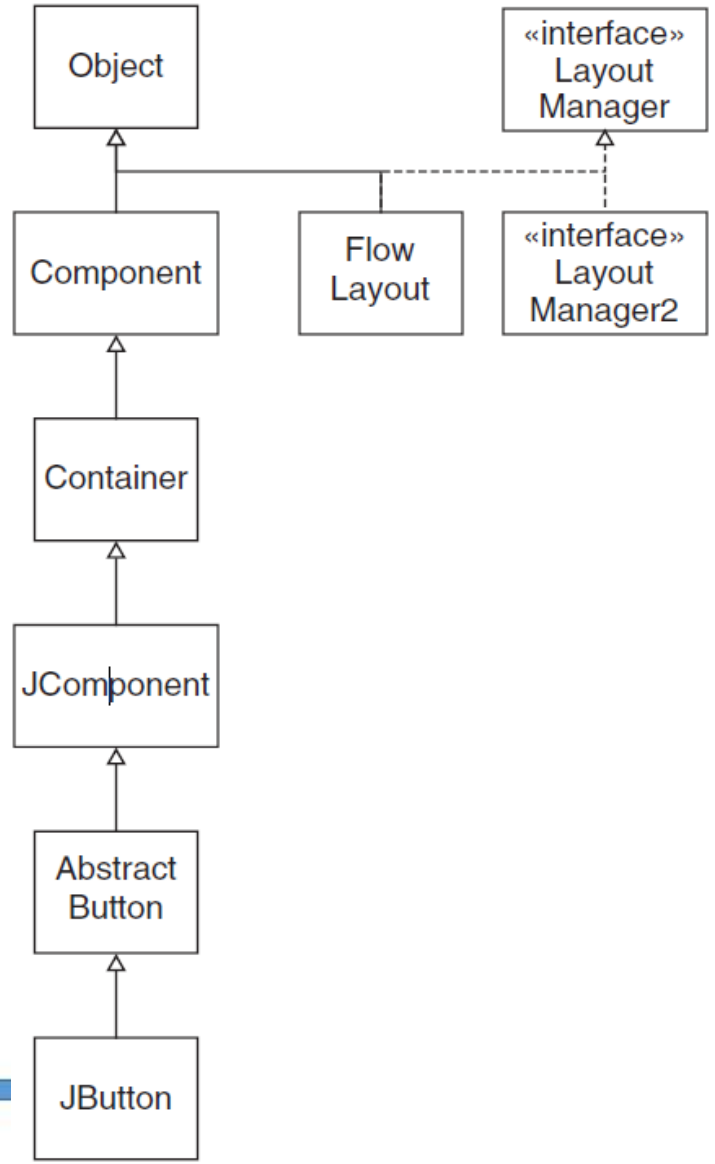
Nesne yönelimli tasarımın beş önemli kavramı.

- Java tip sistemini ve alt tip ilişkisi.
- **Tüm Java sınıflarının ortak üst sınıfı olan Object sınıfı ve sağladığı hizmetler.**
- Bir programın kendi nesnelerini ve sınıflarını analiz etmesine ve Java dilinin tür parametreleriyle sınıfları ve yöntemleri uygulamaya yönelik yeni bir özelliği olan genel programlamayı incelemesine izin veren yansıma kavramı.
- Bileşenler kavramı.
- Varlıklar kavramı.

Giriş

Java'daki her tür aşağıdakilerden biridir:

1. A primitive type (int, short, long, byte, char, float, double, boolean)
2. A class type
3. An interface type
4. An array type
5. The null type



Java Object sınıfı

- Object sınıfı , varsayılan olarak Java'daki tüm sınıfların üst sınıfıdır. Başka bir deyişle, java'nın en üst sınıfıdır.
- java.lang
- Her sınıfın Objectbir üst sınıfı vardır. Diziler dahil tüm nesneler bu sınıfın yöntemlerini uygular.

```
Object obj=getObject();//we don't know what object will be returned from this method
```

- Object sınıfı, nesnenin karşılaştırılabilmesi, nesnenin klonlanabilmesi, nesnenin bildirilebilmesi vb. gibi tüm nesneler için bazı ortak davranışlar sağlar.

Java Object sınıfı

yöntemler

Değiştirici ve Tip	Yöntem ve Açıklama
`protected Object`	`clone()` Bu nesnenin bir kopyasını oluşturur ve döndürür.
`boolean`	`equals(Object obj)` Başka bir nesnenin buna "eşit" olup olmadığını gösterir.
`protected void`	`finalize()` Çöp toplama, nesneye daha fazla başvuru olmadığını belirlediğinde, bir nesnede çöp toplayıcı tarafından çağrılır.
`Class<?>`	`getClass()` Bunun çalışma zamanı sınıfını döndürür `Object`.
`int`	`hashCode()` Nesne için bir Hash kod değeri döndürür.
`void`	`notify()` Bu nesnenin monitöründe bekleyen tek bir iş parçacığını uyandırır.
`void`	`notifyAll()` Bu nesnenin monitöründe bekleyen tüm iş parçacıklarını uyandırır.

Java Object sınıfı

yöntemler

Değiştirici ve Tip	Yöntem ve Açıklama
`String`	`toString()` Nesnenin dize temsilini döndürür.
`void`	`wait()` Geçerli iş parçacığının, başka bir iş parçacığı bu nesne için `notify()`yöntemi veya yöntemi çağırana kadar beklemesine neden olur .`notifyAll()`
`void`	`wait(long timeout)` Geçerli iş parçacığının, başka bir iş parçacığı bu nesne için `notify()`yöntemi veya yöntemi çağırana veya belirli bir süre geçene kadar beklemesine neden olur.`notifyAll()`
`void`	`wait(long timeout, int nanos)` `notify()`Geçerli iş parçacığının, başka bir iş parçacığı bu nesne için yöntemi veya yöntemi çağırana kadar veya başka bir iş parçacığı geçerli iş parçacığını kesene kadar `notifyAll()`veya belirli bir miktarda gerçek zaman geçene kadar beklemesine neden olur.

Java Object getClass()

Java Object getClass() yöntemi, nesnenin sınıf adını döndürür.

```
object.getClass()
```

getClass() Parametreleri

Yöntem getClass()herhangi bir parametre almaz.

getClass() Dönüş Değerleri

•yöntemi çağıran nesnenin sınıfını döndürür

Java Object equals()

Yöntem equals(), iki nesnenin eşit olup olmadığını kontrol eder.

```
object.equals(Object obj)
```

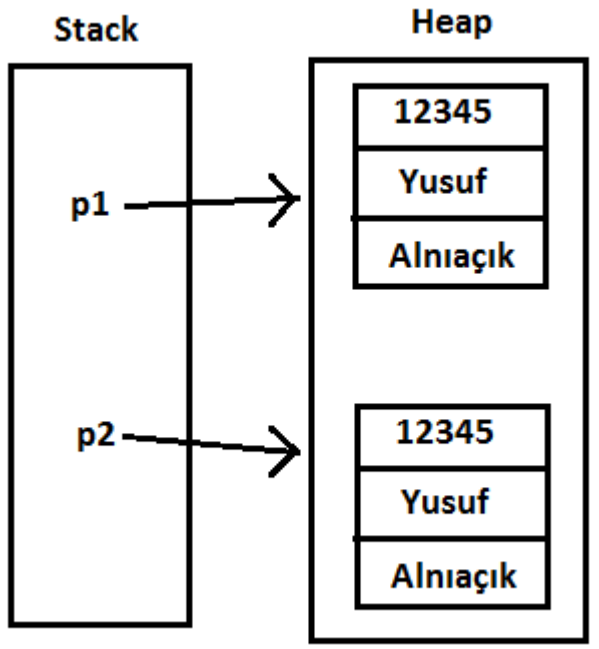
equals() Parametreleri

Yöntem equals()tek bir parametre alır.

obj - mevcut nesneyle karşılaştırılacak nesne

equals() Dönüş Değerleri (boolen)

- returns **true** if two objects are equal
- returns **false** if two objects are not equal



Java Object toString()

Yöntem toString(), nesneyi bir dizgeye dönüştürür ve onu döndürür.

```
object.toString()
```

toString() Parametreleri

Yöntem toString()herhangi bir parametre almaz.

toString() Dönüş Değerleri

•nesnenin metinsel temsilini döndürür

Java Object hashCode()

Nesne için bir Hash kod değeri döndürür.

Hash algoritması tarafından oluşturulan bir tamsayı veya 4 baytlık bir değer döndürür.

Hashing, hashing algoritmaları kavramını kullanarak verileri temsili bir tamsayı değerine eşleme işlemidir. Java'da hash kod, her nesneyle bağlantılı bir tamsayı değeridir. Hashing, veri yapısı uygulamasını HashTables ve HashMaps'te bulur.

Java kitaplığının HashSet ve HashMap sınıfları, öğeleri hızlı bir şekilde bulmak için Hash tabloları kullanır.

toString() Parametreleri

Yöntem toString()herhangi bir parametre almaz.

toString() Dönüş Değerleri

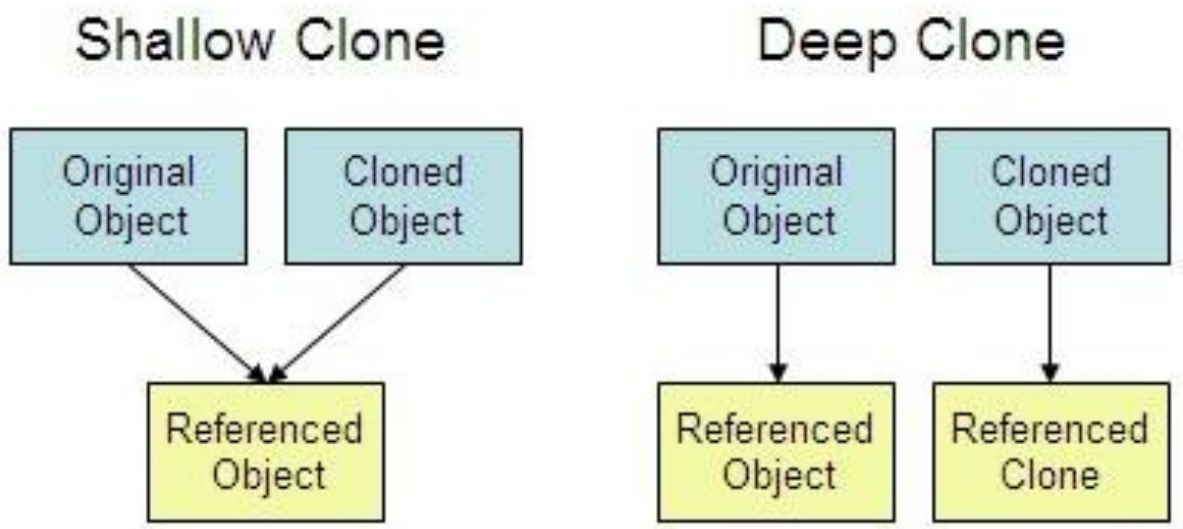
•nesnenin hash kodu dğerini döndürür

Sığ (Shallow) ve Derin (Deep) Kopyalama

- Klonlama , bellekte var olan bir nesnenin tam bir kopyasını oluşturma işlemidir. Java'da klonlama işlemi için Java.lang.Object sınıfının **clone**() yöntemi kullanılmaktadır. Bu yöntem, alandan alan atama yoluyla çağrıldığı bir nesnenin tam bir kopyasını oluşturur ve o nesnenin referansını döndürür.
- Java'daki tüm nesneler klonlama işlemi için uygun değildir. **Cloneable** arabirimini uygulayan nesneler yalnızca klonlama işlemi için uygundur. Klonlanabilir arayüz, işaretleyiciyi klonlama işlemine sağlamak için kullanılan bir işaretleyici arayüzüdür .
- Hem sığ kopya hem de derin kopya bu klonlama işlemiyle ilgilidir. **clone()** yönteminin varsayılan sürümü, bir nesnenin sığ kopyasını oluşturur. Bir nesnenin derin kopyasını oluşturmak için **clone()** yöntemini geçersiz kılmanız gerekir.

Sığ (Shallow) ve Derin (Deep) Kopyalama

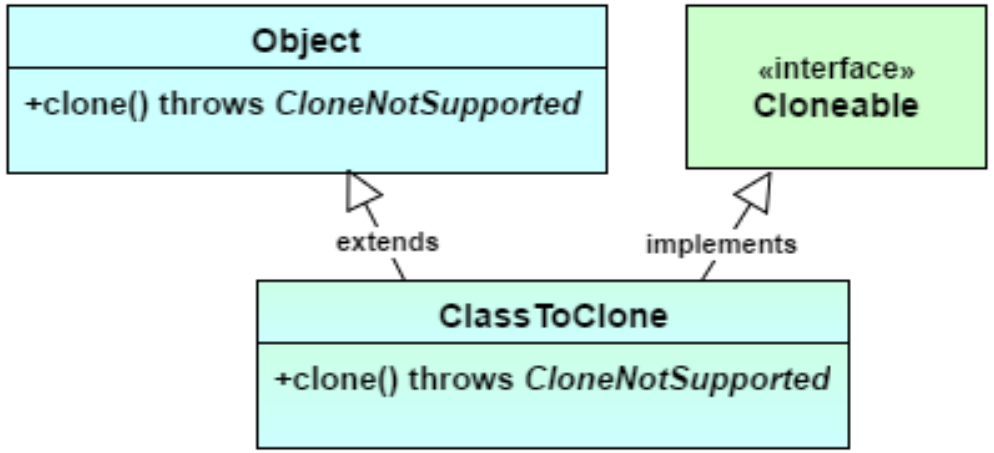
- Bir nesneyi klonlamak, nesnenin bir kopyasını oluşturmak anlamına gelir. Bir nesne verildiğinde, o nesnenin 'tam' kopyaları olan birden çok nesne oluşturmak istiyorsanız, ancak yeni bir nesne örneği tanımlama sürecinden geçmek istemiyorsanız, klonlama yapılır.
- **Nesneleri her zaman sıfırdan başlatmak ve doldurmak yerine nesnelerin klonlarını oluşturmak bir tasarım kararıdır. (Prototip Tasarım Modeli)**

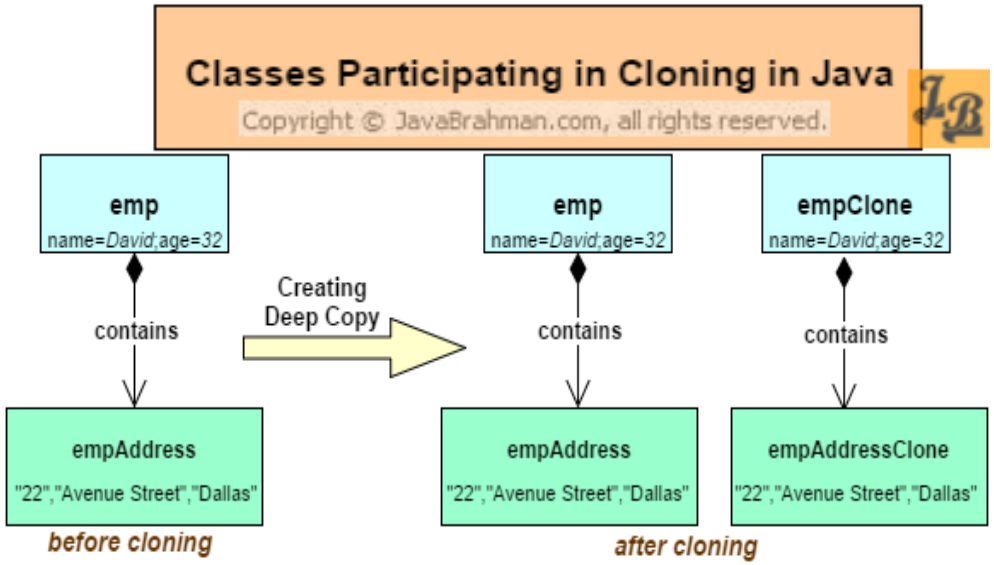


Sığ (Shallow) ve Derin (Deep) Kopyalama

- **Cloneable.java: java.lang.Cloneable** bir işaretleyici arayüzüdür (Nesnelerinin klonlanma kabiliyetini sağlamayı amaçlayan herhangi bir sınıfın **Cloneable** arabirimini uygulaması gerekir.

- (Bir işaretleyici arabirimi boş bir arabirimdir, yani herhangi bir yöntem içermez. Amacı, Java çalışma zamanına, onu uygulayan herhangi bir sınıf tarafından oynanacak belirli bir rol hakkında bilgi vermektir. Yaygın olarak kullanılan işaretleyici arabirimlerinin örnekleri arasında Serializable, Cloneablevb.)



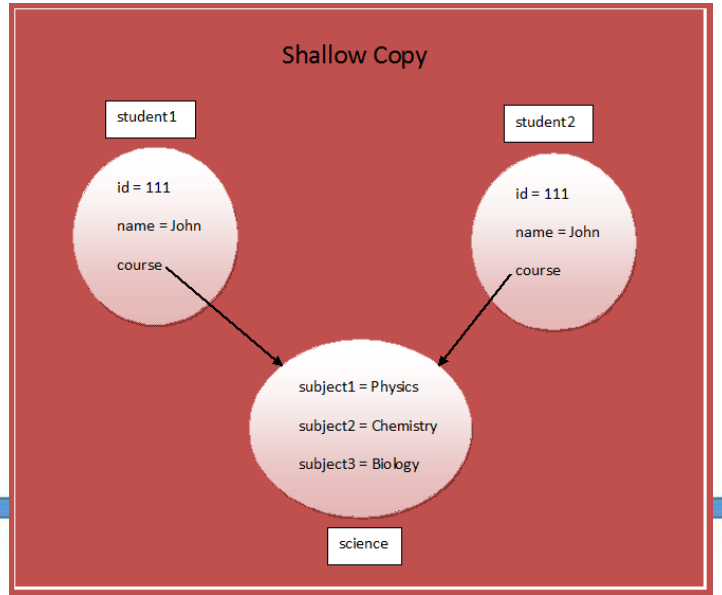


Sığ (Shallow) ve Derin (Deep) Kopyalama

SIĞ KOPYALAMA

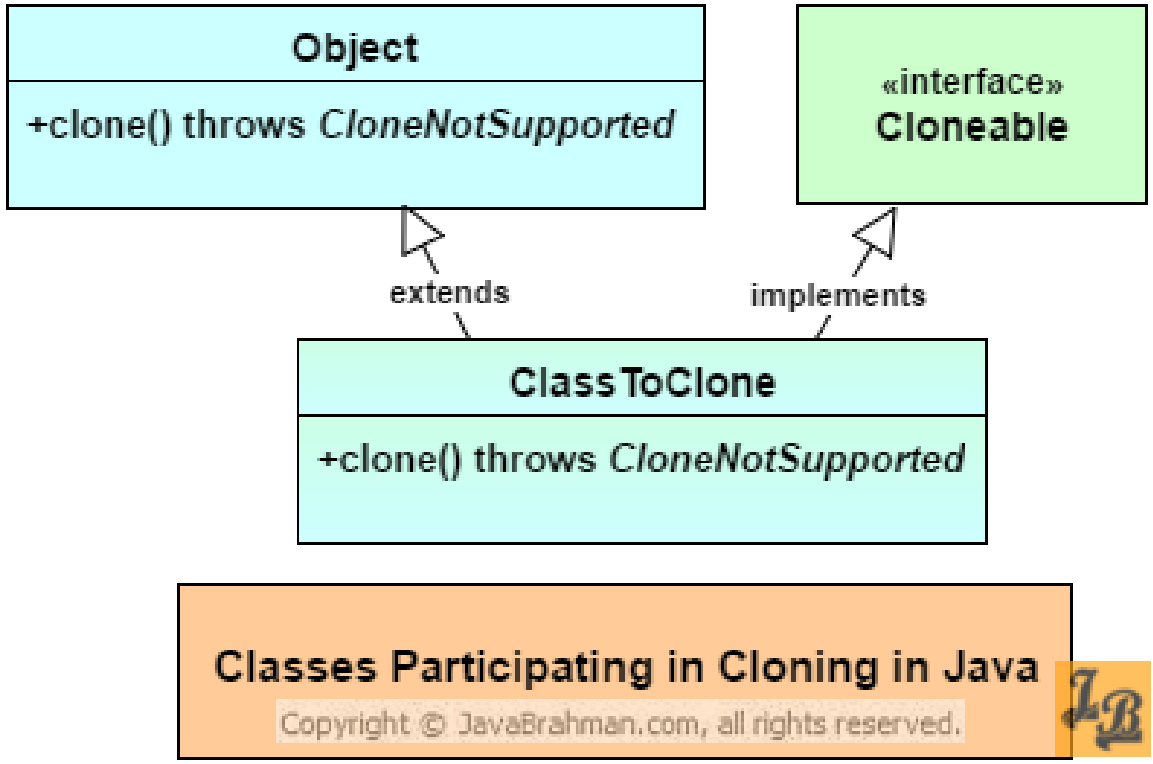
- Sığ kopylamada, orijinal veya klon nesnenin herhangi bir referansı değiştirildiğinde, diğerininde refe

```
//Default version of clone() method. It creates shallow copy of an object.
protected Object clone() throws CloneNotSupportedException
{
    return super.clone();
}
```



Sığ (Shallow) ve Derin (Deep) Kopyalama

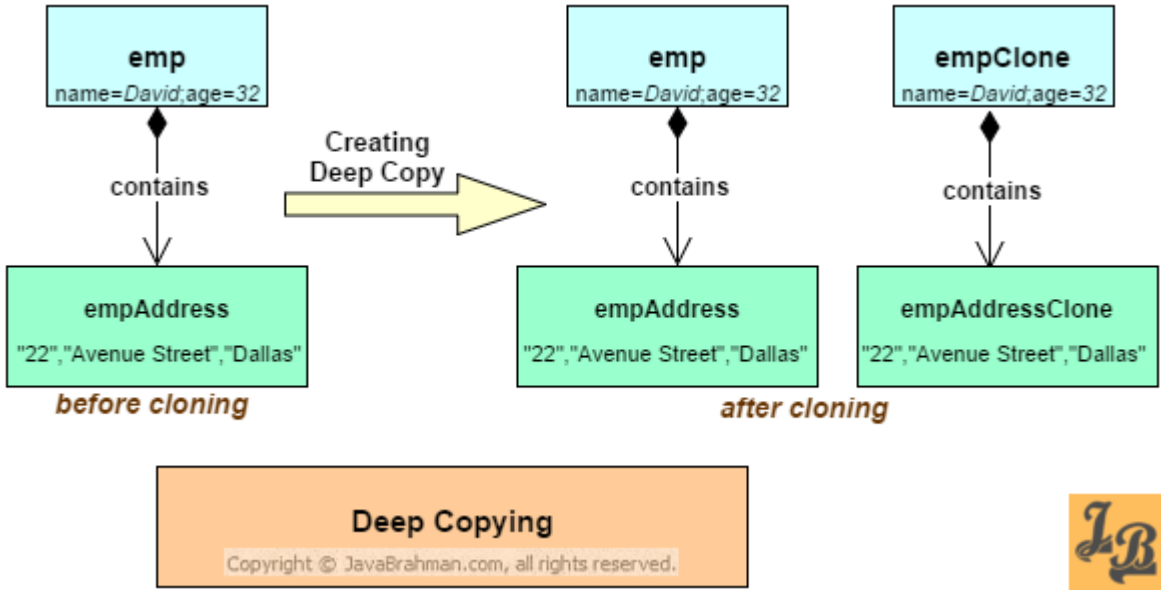
```
public class Employee implements Cloneable
{
   public Employee clone()
   {
      try
      {
         return (Employee) super.clone();
      }
      catch (CloneNotSupportedException e)
      {
         return null; // Won't happen
      }
   }
   . . .
}
```



Sığ (Shallow) ve Derin (Deep) Kopyalama

DERİN KOPYALAMA

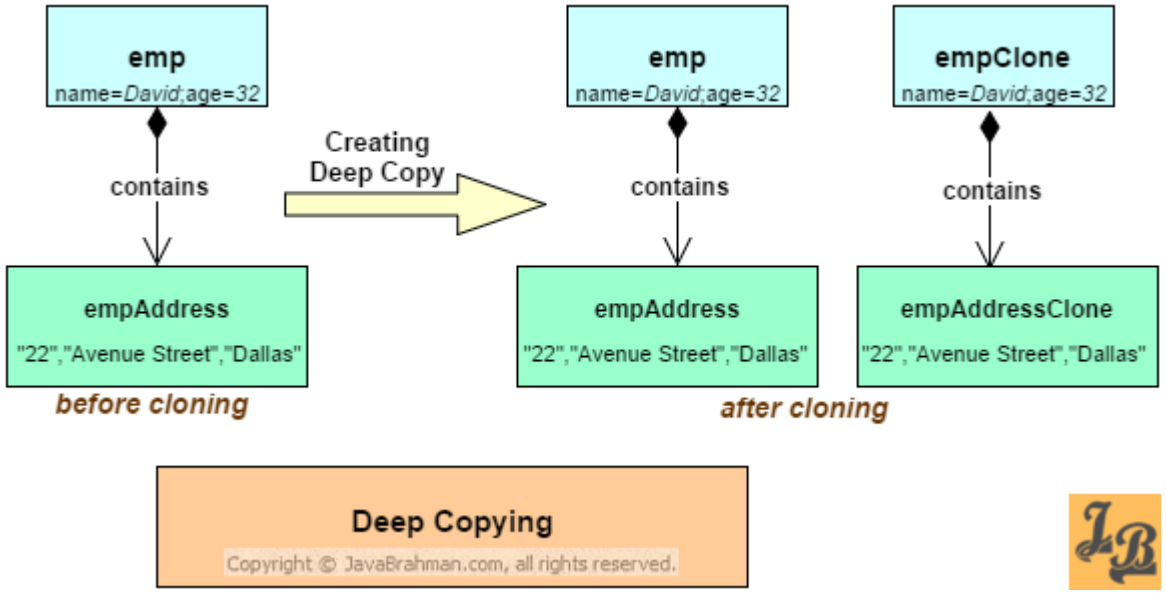
- Tamamen bağımsız/ayrı nesne klonlarına sahip olmak için Derin Kopyalama yapılmalıdır. Derin kopyalama, yalnızca "sığ" ilkel değerleri klonlamakla kalmaz, aynı zamanda klonlanan nesnenin içindeki iç içe nesne kiralama düzeninin kopyalarını da oluşturur. Bir nesnenin derin kopyasını oluşturmak için **clone()** yöntemini geçersiz kılmanız gerekir.



Sığ (Shallow) ve Derin (Deep) Kopyalama

DERİN KOPYALAMA

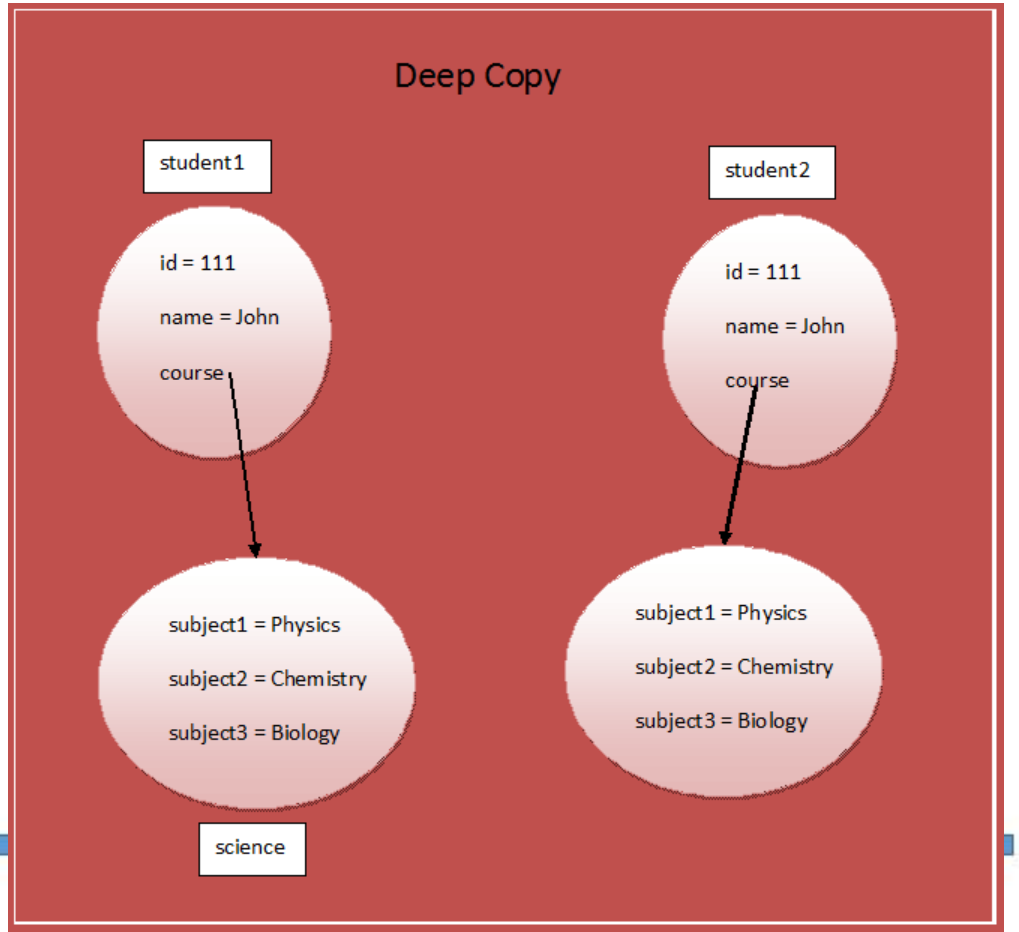
- Bir nesnenin derin kopyası, tıpkı sığ kopya gibi, orijinal nesnenin tüm alanlarının tam kopyasına sahip olacaktır. Ancak ek olarak, orijinal nesnenin alan olarak başka nesnelere referansları varsa, o zaman bu nesnelerin kopyaları da üzerlerinde clone() yöntemi çağrılarak oluşturulur. Bu, klon nesnesi ve orijinal nesnenin %100 ayrık olacağı anlamına gelir. Birbirlerinden %100 bağımsız olacaklardır. Nesneyi klonlamak için yapılan herhangi bir değişiklik orijinal nesneye yansıtılmayacaktır veya bunun tersi de geçerlidir.



Sığ (Shallow) ve Derin (Deep) Kopyalama

DERİN KOPYALAMA

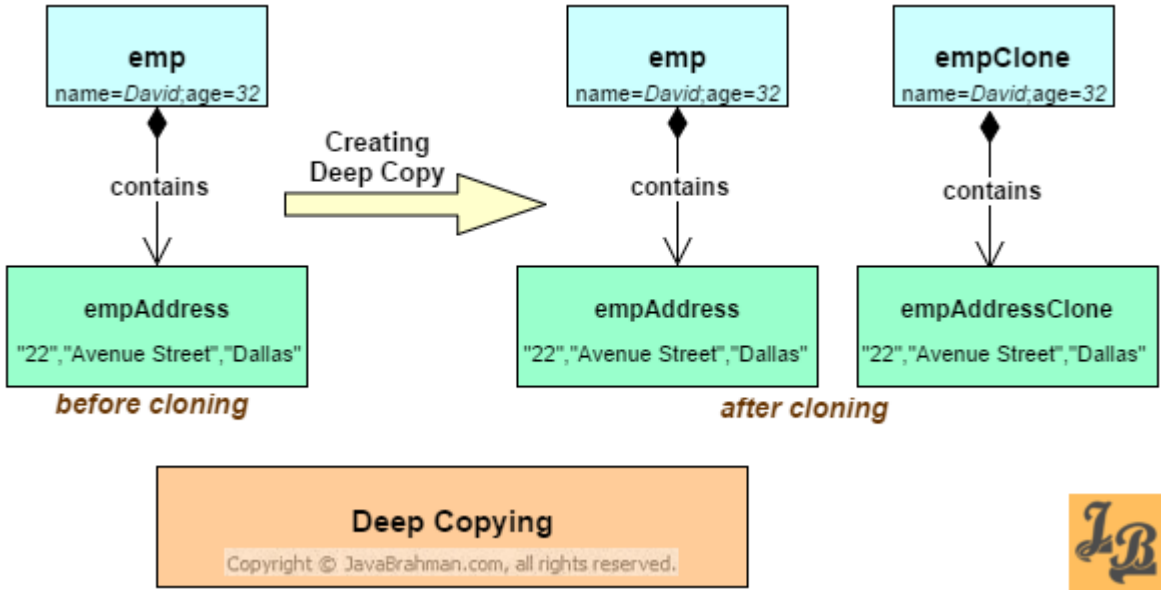
```
//Overriding clone() method to create a deep copy of an object.
protected Object clone() throws CloneNotSupportedException
{
    Student student = (Student) super.clone();
    student.course = (Course) course.clone();
    return student;
}
```



Sığ (Shallow) ve Derin (Deep) Kopyalama

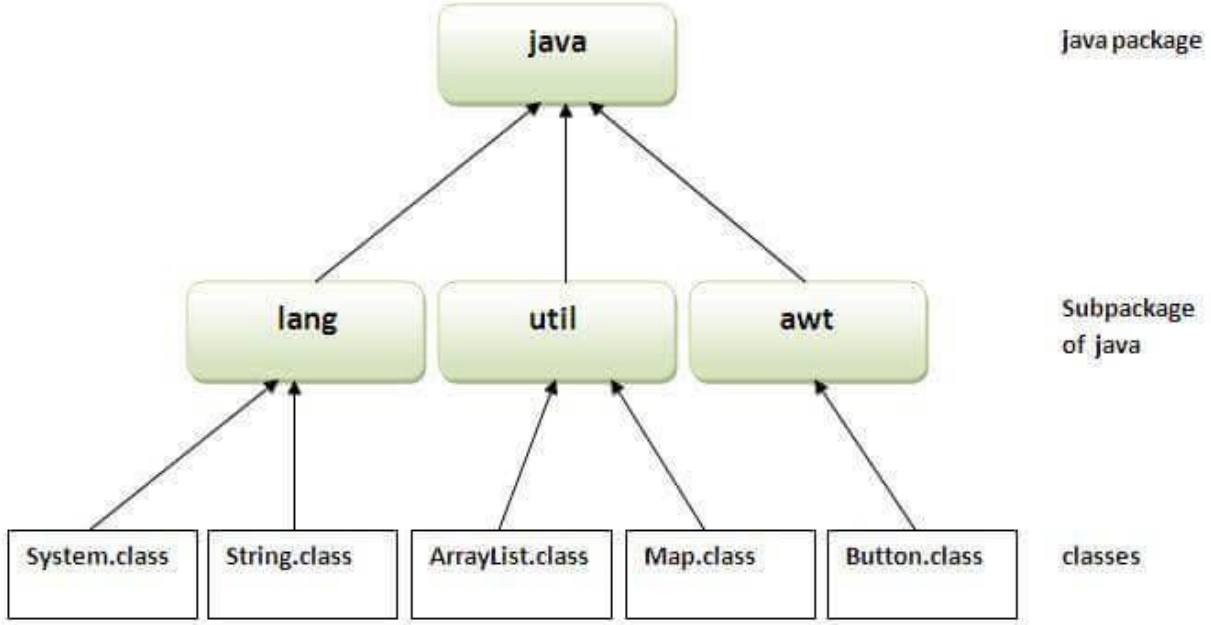
DERİN KOPYALAMA

- Bir nesnenin derin kopyası, tıpkı sığ kopya gibi, orijinal nesnenin tüm alanlarının tam kopyasına sahip olacaktır. Ancak ek olarak, orijinal nesnenin alan olarak başka nesnelere referansları varsa, o zaman bu nesnelerin kopyaları da üzerlerinde clone() yöntemi çağrılarak oluşturulur. Bu, klon nesnesi ve orijinal nesnenin %100 ayrık olacağı anlamına gelir. Birbirlerinden %100 bağımsız olacaklardır. Nesneyi klonlamak için yapılan herhangi bir değişiklik orijinal nesneye yansıtılmayacaktır veya bunun tersi de geçerlidir.



Packages

- Adlandırma çakışmalarını önlemek, erişimi kontrol etmek, sınıfların, arayüzlerin, numaralandırmaların ve ek açıklamaların aranmasını / bulunmasını ve kullanımını kolaylaştırmak için Java'da paketler kullanılır.
- Bir Paket, erişim koruması ve ad alanı yönetimi sağlayan ilgili türlerin (sınıflar, arabirimler, numaralandırmalar ve ek açıklamalar) bir gruplaması olarak tanımlanabilir.



Built in Packages

Java API, Java Geliştirme Ortamına dahil olan, kullanımı ücretsiz olan önceden yazılmış sınıflardan oluşan bir kitaplıktır.

Kitaplık, girişi yönetmek, veritabanı programlamak ve çok daha fazlası için bileşenler içerir. Tam liste Oracles web sitesinde bulunabilir: https://docs.oracle.com/javase/8/docs/api/.

Kitaplık, paketlere ve sınıflara ayrılmıştır. Yani, tek bir sınıfı (yöntemleri ve nitelikleriyle birlikte) veya belirtilen pakete ait tüm sınıfları içeren bütün bir paketi içe aktarabilirsiniz.

Built in Packages

```java
import java.util.Scanner;

class MyClass {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);
    System.out.println("Enter username");

    String userName = myObj.nextLine();
    System.out.println("Username is: " + userName);
  }
}
```

Packages

Programcılar, sınıflar / arabirimler vb. Gruplarını bir araya getirmek için kendi paketlerini tanımlayabilir.

Paket yeni bir ad alanı oluşturduğundan, diğer paketlerdeki adlarla herhangi bir ad çatışması olmayacaktır. Paketleri kullanarak erişim kontrolü sağlamak daha kolaydır ve ilgili sınıfları bulmak da daha kolaydır.

Interfaces-Arayüzler

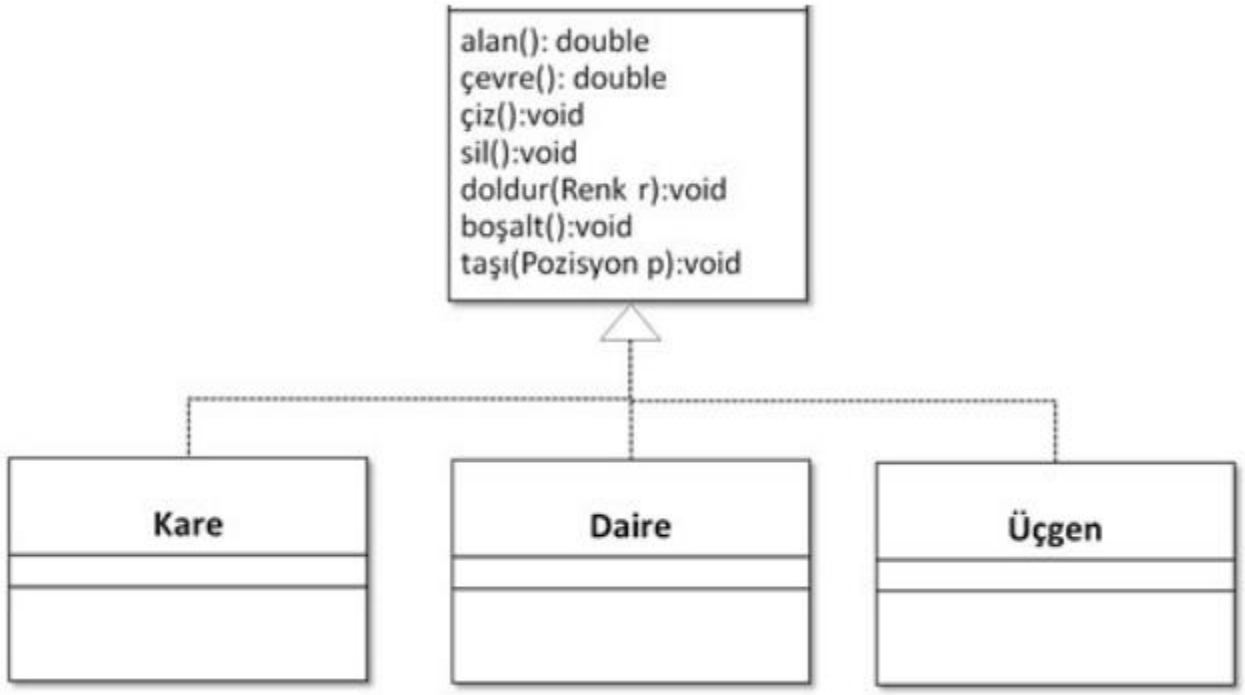
- Arayüz, gövdesi olmayan bir grup yöntem içeren tamamen soyut bir sınıftır.
- Arayüz, Java'da bir referans türüdür. Sınıfa benzer. Soyut yöntemler koleksiyonudur. Bir sınıf bir arabirim uygular, dolayısıyla arabirimin soyut yöntemlerini miras alır.
- Soyut yöntemlerin yanı sıra, bir arabirim ayrıca sabitler, varsayılan yöntemler, statik yöntemler ve iç içe türler içerebilir. Yöntem gövdeleri yalnızca varsayılan yöntemler ve statik yöntemler için mevcuttur.
- Arayüz yazmak, sınıf yazmaya benzer. Ancak bir sınıf, bir nesnenin niteliklerini ve davranışlarını tanımlar. Ve bir arayüz, bir sınıfın uyguladığı davranışları içerir.
- Arabirimi uygulayan sınıf soyut olmadığı sürece, arabirimin tüm yöntemlerinin sınıfta tanımlanması gerekir.

Interfaces-Arayüzler

Avantajları:

- Soyut sınıflara benzer şekilde, arayüzler Java'da soyutlama elde etmemize yardımcı olur .
- Arabirimler , bir sınıfın (onu uygulayan) izlemesi gereken özellikleri sağlar .
- Arayüzler ayrıca Java'da çoklu kalıtım elde etmek için kullanılır.

Interfaces-Arayüzler



Interfaces-Arayüzler

```
interface <interface_name>{

   // declare constant fields
   // declare methods that abstract
   // by default.
}
```

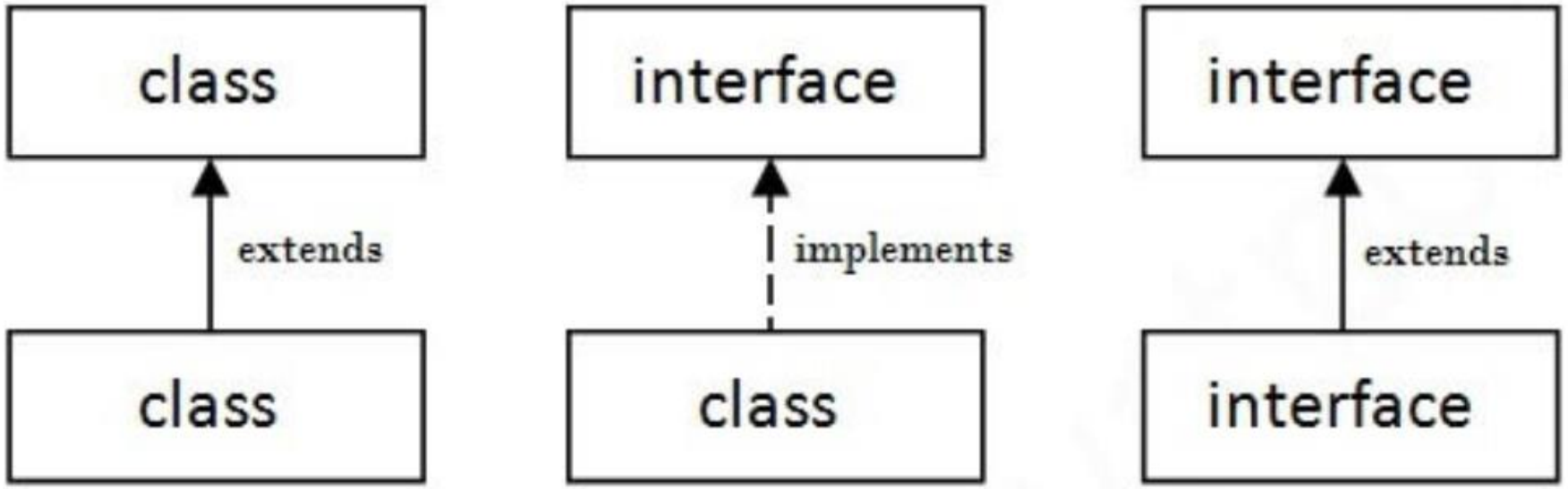
- Bir arayüz, herhangi bir sayıda yöntem içerebilir.
- Bir arayüz örtük olarak soyuttur. Bir arayüz bildirirken abstract anahtar sözcüğünü kullanmanıza gerek yoktur.
- Bir arayüzdeki her yöntem de dolaylı olarak soyuttur, bu nedenle soyut anahtar sözcüğe gerek yoktur.
- Bir arayüzdeki yöntemler örtük olarak geneldir.

Interfaces-Arayüzler

Bir arayüz, bir sınıftan birkaç yönden farklıdır:

- Bir arabirimi örnekleyemezsiniz.
- Bir arayüz herhangi bir kurucu içermez.
- Bir arayüzdeki tüm yöntemler soyuttur.
- Bir arayüz, örnek alanları içeremez. Bir arabirimde görünebilecek alanların hem statik hem de son olarak bildirilmesi gerekir.
- Bir arayüz bir sınıf tarafından genişletilmez; bir sınıf tarafından uygulanmaktadır.
- Bir arayüz birden çok arayüzü genişletebilir.

Interfaces-Arayüzler



Interfaces-Çoklu Arayüzler

```
interface A {
  // members of A
}

interface B {
  // members of B
}

class C implements A, B {
  // abstract members of A
  // abstract members of B
}
```

Interfaces-Çoklu Arayüzler

```
interface A {
    ...
}
interface B {
    ...
}

interface C extends A, B {
    ...
}
```

Interfaces-Arayüzler- default motodlar

```
public default void getSides() {
   // body of getSides()
}
```

```
interface Drawable{
void draw();
default void msg(){System.out.println("default method");}
}
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}
class TestInterfaceDefault{
public static void main(String args[]){
Drawable d=new Rectangle();
d.draw();
d.msg();
}}
```

Interfaces-Arayüzler- Static motodlar

```
interface Drawable{
void draw();
static int cube(int x){return x*x*x;}
}
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}

class TestInterfaceStatic{
public static void main(String args[]){
Drawable d=new Rectangle();
d.draw();
System.out.println(Drawable.cube(3));
}}
```

Interfaces-Arayüzler- Nested Interface

```
interface printable{
 void print();
 interface MessagePrintable{
   void msg();
 }
}
```

Java’da nesneler gibi new sözcüğü ile oluşturulurlar.

```
Kişi k = new Kişi();
```

Bu işlemle yeni bir Kişi nesnesi oluşturulur ve k bu nesneye bir referanstır. Kişi() bir metot çağrısıdır. Buna kurucu metot (constructor) denir. Kurucu metotun adı sınıf adı ile aynıdır. Döndürdüğü bir değer olmaz (void). Şayet sınıf içinde böyle bir kurucu metot varsa, ilgili kurucu metot çalıştırılır. Böyle bit metot yoksa Java otomatik olarak varsayılan bir kurucu metot ekler, varsayılan kurucu metotun parametresi yoktur. Birden fazla kurucu metot olabilir, fakat parametre listesi farklı olmalıdır. Aşağıda örnek kurucu metotlar verilmiştir.

```
public class Kişi {
    private String ad;
    private String soyad;
    public void Kişi() { // varsayılan kurucu metot
        ad = “”;
        soyad = “”;
    }
    public void Kişi(String ad, String soyad) { // ikinci kurucu metot
        this.ad = ad;
        this.soyad = soyad;
    }
    // …
}
```

Şayet kurucu metotlar tanımlanmışsa ve bunlar arasında varsayılan kurucu metot (boş parametreli) yoksa, o zaman varsayılan kurucu metot otomatik olarak eklenmez, bu durumda kurucu metotun geçerli olmadığı kabul edilir ve şayet çağırılısa Java derleme hatası verir.

Alt sınıflardan nesneler oluşturulduğunda önce alt sınıfların uygun olan kurucu metotları, sonra yukarı doğru üst sınıfların varsayılan kurucu metotları otomatik olarak çağrılır. Üst sınıfın varsayılan kurucu metodunun çağrılması için super() komutu otomatik çağrılır. Şayet üst sınıflardan birinin varsayılan kurucu metodu yoksa derleyici hata verir. Bu durumda kurucu metot içinde bir üst sınıfın uygun kurucu metodunu çağırmak için uygun çağrı yine super(…) çağrısı ile kurucu metotun ilk komutu olarak yapılmalıdır.

```
public class Öğrenci extends Person {
   private String öğrenciNo;
   public void Öğrenci() {
     öğrenciNo = “”; // önce otomatik super() çağrılır, sonra bu işlem
   }
   public void Öğrenci(String ad, String soyad) {
      super(ad, soyad); // super() yerine bu metotla uygun kurucu çağrılır
      this(); // varsayılan kurucu çağrılır
   }
   public void Öğrenci(String ad, String soyad, String no) {
      super(ad, soyad); // super() yerine bu metotla uygun kurucu çağrılır
      öğrenciNo = no;
   }
}
```

Java gibi modern nesnesel dillerde sınıfların silinmesi (destructor) işlemi açık olarak yapılamaz. Bir nesneye herhangi bir referans kalmadığında ilgili nesneye erişilemeyeceği için, yani ancak referans sayısı sıfır olduğunda, ilgili nesne otomatik olarak bellekten silinir. Buna çöp toplama (garbage collection) denir ve oldukça etkin bir şekilde çalışmaktadır. Ancak bir kısım daha önceden geliştirilmiş nesnesel programlama dillerinde, örneğin C++'da, silme işleminin açık bir şekilde kodlama ile yapılması ve nesneler için silici metotların (destructor) yazılması beklenir.

```
public class Öğrenci extends Person {
   private String öğrenciNo;
   public void Öğrenci() {
     öğrenciNo = “”; // önce otomatik super() çağrılır, sonra bu işlem
   }
   public void Öğrenci(String ad, String soyad) {
      super(ad, soyad); // super() yerine bu metotla uygun kurucu çağrılır
      this(); // varsayılan kurucu çağrılır
   }
   public void Öğrenci(String ad, String soyad, String no) {
      super(ad, soyad); // super() yerine bu metotla uygun kurucu çağrılır
      öğrenciNo = no;
   }
}
```

Java gibi modern nesnesel dillerde sınıfların silinmesi (destructor) işlemi açık olarak yapılamaz. Bir nesneye herhangi bir referans kalmadığında ilgili nesneye erişilemeyeceği için, yani ancak referans sayısı sıfır olduğunda, ilgili nesne otomatik olarak bellekten silinir. Buna çöp toplama (garbage collection) denir ve oldukça etkin bir şekilde çalışmaktadır. Ancak bir kısım daha önceden geliştirilmiş nesnesel programlama dillerinde, örneğin C++'da, silme işleminin açık bir şekilde kodlama ile yapılması ve nesneler için silici metotların (destructor) yazılması beklenir.

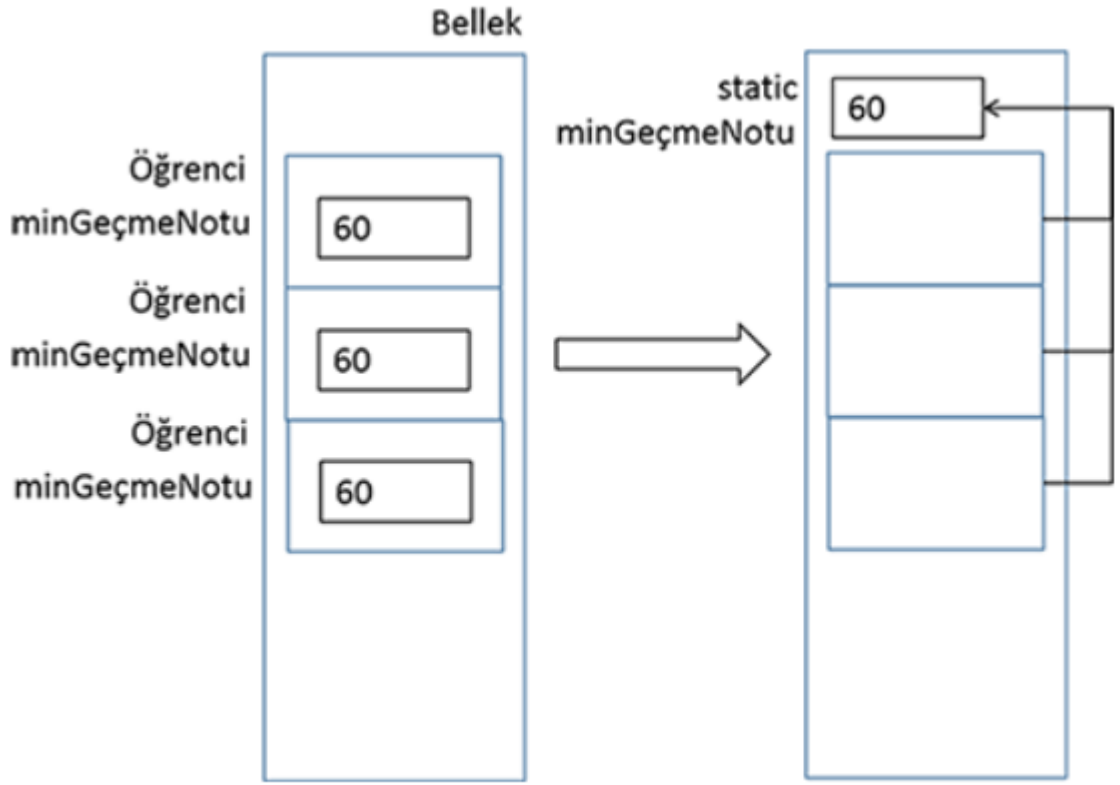
static anahtar sözcüğü

Java'daki static anahtar sözcüğü esas olarak bellek yönetimi için kullanılır. Static anahtar kelimesini değişkenler, yöntemler, bloklar ve iç içe sınıflarla uygulayabiliriz. Static anahtar sözcüğü, sınıfın bir örneğinden çok sınıfa aittir.

Static şunlar olabilir:

- Değişken (sınıf değişkeni olarak da bilinir)
- Yöntem (sınıf yöntemi olarak da bilinir)
- Blok
- İç içe geçmiş sınıf.

static anahtar sözcüğü



(Şekil). "static" değişkenler için bellek atama

Tasarım Deseni Nasıl Seçilir?

1. Tasarım kalıplarının tasarım problemlerini nasıl çözdüğünü düşünün.
2. Niyet bölümlerini araştırın.
3. Modellerin birbiriyle nasıl ilişkili olduğunu inceleyin.
4. Benzer amaca yönelik kalıpları inceleyin.
5. Yeniden tasarımın bir nedenini inceleyin.
6. Tasarımınızda neyin değişken olması gerektiğini düşünün.

Observer Patterns

GoF Tanımı: Nesneler arasında birden çoğa bağımlılık tanımlayın, böylece bir nesne durum değiştirdiğinde, tüm bağımlıları otomatik olarak bildirilir ve güncellenir.

Konsept

Bu modelde, belirli bir özneyi (nesneyi) gözlemleyen birçok gözlemci (nesne) vardır. Gözlemciler temelde ilgilenirler ve o konuda bir değişiklik yapıldığında haberdar olmak isterler. Böylece kendilerini o konuya kaydettirirler. Konuya olan ilgilerini kaybettiklerinde, konunun kaydını silerler. Bazen bu modele Yayıncı-Abone modeli de denir.

Observer Patterns

Bilgisayar Dünyası Örneği

Bilgisayar bilimi dünyasında, bu kullanıcı arabiriminin bir veritabanıyla (veya iş mantığıyla) bağlantılı olduğu, kullanıcı arabirimi tabanlı basit bir örnek düşünün. Bir kullanıcı, bu UI aracılığıyla bazı sorgular yürütebilir ve veritabanını aradıktan sonra, sonuç UI'ye geri yansıtılır. Çoğu durumda, kullanıcı arayüzünü veritabanıyla ayırırız. Veritabanında bir değişiklik olursa, değişikliğe göre görüntüsünü güncelleyebilmesi için UI'ye bildirilmelidir.

Observer Patterns

İllüstrasyon

Şimdi doğrudan basit örneğimize girelim. Burada bir **«observer»** (daha fazlasını yaratabilirsiniz) ve bir **«subject»** oluşturalım. **«subject»** , tüm **«observer»** için bir liste tutar (ancak burada basitlik için sadece bir tane var). Buradaki **«observer», «subject» ile** ilgili bayrak değeri değiştiğinde bilgilendirilmek istiyor. Çıktı ile, bayrak değeri 5 veya 25 olarak değiştirildiğinde gözlemcinin bildirimleri aldığını keşfedeceksiniz. Ancak bayrak değeri 50'ye değiştiğinde herhangi bir bildirim yok çünkü bu zamana kadar gözlemci kendisini konudan kaydını silmiş durumda olacak.

Observer Patterns