PART 17

17. Kalıtım

Kalıtım; bir nesnenin, başka bir nesnenin özelliklerinin edinmesi işlemi olarak tanımlanabilir. Kalıtımın kullanımı ile, bilgi hiyerarşik sırayla yönetilebilir.

Kalıtımdan bahsettiğimiz zaman, genelde en çok kullanılar anahtar kelimeler **extends** ve **implements** olmaktadır. Bu kelimeler, bir nesnenin, diğer bir nesnenin IS-A türünde olup olmadığını belirleyecektir. Bu anahtar kelimeleri kullanarak, bir nesnenin başka bir nesnenin özelliklerini edinmesini sağlayabiliriz.

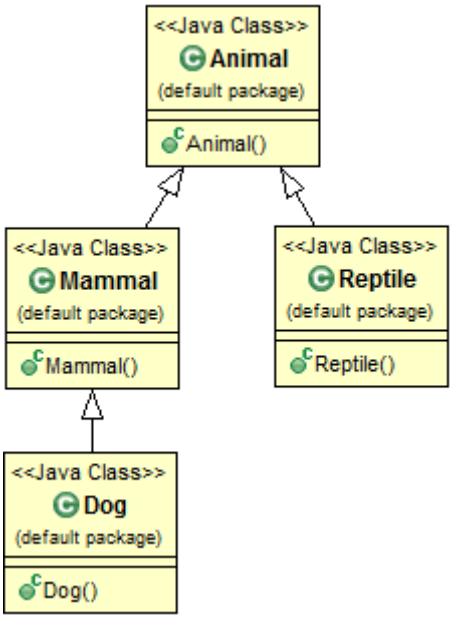
17.1 IS-A İlişkisi

IS-A, bu nesne o nesnenin bir türüdür, demenin bir yoludur. O zaman **extends** anahtar kelimesinin, kalıtımı gerçekleştirmek için nasıl kullanıldığına bir bakalım.

```
public class Animal{
}

public class Mammal extends Animal{
}

public class Reptile extends Animal{
}

public class Dog extends Mammal{
}
```

Yukardaki örneğe dayanarak, aşağıdakiler tanımlar Object Oriented açısından doğrudur:

- Animal, Mammal sınıfının üstsınıfıdır.
- Animal, Reptile sınıfının üstsınıfıdır.
- Mammal ve Reptile, Animal sınıfının altsınıflarıdır.
- Dog, hem Mammal hemde Animal sınıfının altsınıfıdır.



Eğer IS-A ilişkisini dikkate alırsa, bunları söyleyebiliriz:

- Mammal IS-A Animal (Mammal bir Animal'dır)
- Reptile IS-A Animal(Reptile bir Animal'dır)
- Dog IS-A Mammal(Dog bir Mammal'dır)
- **Bu yüzden : Dog IS-A Animal(Dog da bir Animal'dır)**

Extends anahtar kelimesinin kullanımı ile; altsınıflar, üstsınıfların private özellikleri hariç bütün özelliklerini kalıt alabilecektir.

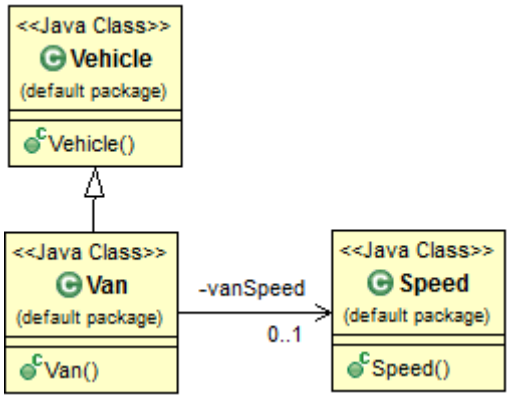
Instance operatör kullanarak, Mammal sınıfının aslında bir Animal sınıfı olduğuna emin olabiliriz.

17.2 HAS-A İlişkisi

Bu ilişkiler başlıca kullanıma dayanırlar. Bir sınıfın bir şeyle HAS-A(sahip) ilişkisi olduğunu belirler. Bu ilişki, kod tekrarının azaltılmasına yardım etmekle birlikte bug'ların da azaltılmasına yardımcı olur.

Örnek içinde bir bakalım:

```
public class Vehicle{}
public class Speed{}
public class Van extends Vehicle{
        private Speed vanSpeed;
}
```



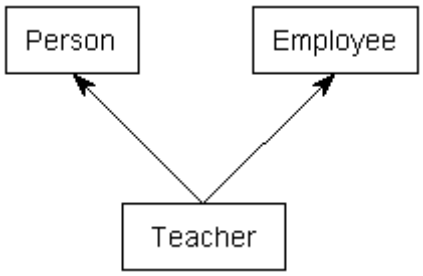
Bu diyagram, Van sınıfının Speed sınıfına sahip olduğunu göstermektedir.(HAS-A) Speed için ayrı bir sınıfa sahip olursak, Van sınıfı içinde Speed'e ait bütün bir kodu koymak zorunda değiliz, bu da Speed sınıfını birden çok uygulama içinde tekrar kullanılmasını mümkün kılar.

17.3 Tekil/Çoğul Kalıtım

Bir önemli konuda, Java'nın sadece tekil kalıtımı desteklediğidir. Bu da , bir sınıfın birden çok sınıfa genişletilemeyeceği anlamına gelmektedir. Sonuç olarak, aşağıdaki kural dışıdır:

```
public class Dog extends Animal, Mammal{}
```

Bununla birlikte, C++ çoklu kalıtım yeteneği sağlar. Çoklu kalıtım, türetilmiş sınıfın birden çok parent sınıfının üyelerini kalıt almasına olanak sağlar.




```
#include <string>
class Person {
};

class Employee {
};

// Teacher publicly inherits Person and Employee
class Teacher: public Person, public Employee {
```

```
};
```

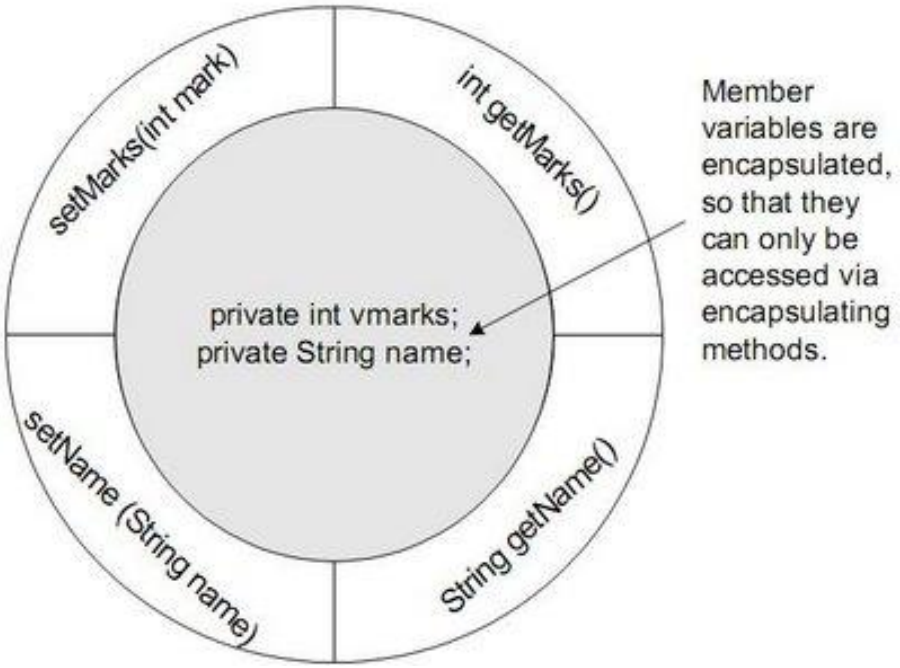
Java, Çoklu Kalıtım fonksiyonelliğini sağlamak için Interface'leri kullanır. Bir sınıf , bir veya birden çok interface uygulayabilir. Bu özellik, Java'nın çoklu kalıtım imkansızlığından kurtulmasını sağlar.

implements anahtar kelimesi, sınıfların interface'lerden kalıt alması için kullanılır. Interface'ler sınıflar tarafından asla genişletilemezler.

PART 18

18. 1 Kapsülleme(Encapsulation)

Kapsülleme, temel bir nesne yönelimi programlama konseptidir. Kapsülleme, bir sınıf içindekileri alanları(field) private yapıp , bu alanlara public metotlarla erişebilme tekniğidir. Eğer alan private olarak deklare edilmişse, ona sınıf dışından erişilemez, böylece sınıf içindeki alanları gizlemiş oluruz. Bu sebepten ötürü, kapsülleme veri gizleme olarak da adlandırılır.



Kapsülleme, sınıfın dışında tanımlanmış bir kodun, sınıfın içindeki veri ve koda rastgele erişimini engelleyen bir koruyucu bariyer olarak tanımlanabilir. Veri ve koda erişim, getter ve setter'lar aracılığıyla sıkı bir şekilde kontrol edilir.

Örnek

Kapsüllemeyi anlatan örneğe bir bakalım:

```
/* File name : EncapTest.java */
public class EncapTest{

   private String name;
   private String idNum;
   private int age;

   public int getAge(){
      return age;
   }

   public String getName(){
      return name;
   }

   public String getIdNum(){
```

```
      return idNum;
   }

   public void setAge( int newAge){
      age = newAge;
   }

   public void setName(String newName){
      name = newName;
   }

   public void setIdNum( String newId){
      idNum = newId;
   }
}
```

Public metotlar, bu sınıfın alanlarına java dünyası dışından erişim noktalarıdır. Bu nedene, değişkenlere erişmek isteyen herhangi bir sınıf ,onlara getter ve setter'lar üzerinden erişmelidir.

EncapTest sınıfının değişkenlerine aşağıdaki gibi erişilebilir:

```
/* File name : RunEncap.java */
public class RunEncap{

   public static void main(String args[]){

      EncapTest encap = new EncapTest();
      encap.setName("James");
      encap.setAge(20);
      encap.setIdNum("12343ms");

      System.out.print("Name : " + encap.getName()+ " Age : "+ encap.getAge());
    }
}
```

Bu aşağıdaki sonucu üretecektir:

```
Name : James Age : 20
```

Kapsüllemenin faydaları:

- Sınıfın alanları, salt okunur veya salt yazılır yapılabilir.
- Sınıfın kullanıcıları, verinin nasıl tutulduğu hakkında bir bilgisi yoktur. Bir sınıf, bir alanın veri türünü değiştirebilir ve sınıfın kullanıcılarının kodlarını değiştirmeleri gerekmez.