



Nesne Tabanlı Programlama-II

2

4.HAFTA

KALITIM

Java'da Kalıtım (IS-A)

- **Java'da kalıtım**, bir nesnenin üst nesnenin tüm özelliklerini ve davranışlarını edindiği bir mekanizmadır.
- Java'da kalıtımın arkasındaki fikir, mevcut sınıflar üzerine inşa edilmiş yeni sınıflar oluşturabilmenizdir.
- Varolan bir sınıftan miras aldığınızda, üst sınıfın yöntemlerini ve alanlarını yeniden kullanabilirsiniz.
- Ayrıca, mevcut sınıfınıza yeni yöntemler ve alanlar da ekleyebilirsiniz.
- Kalıtım, *ebeveyn-çocuk* ilişkisi olarak da bilinen **IS-A ilişkisini** temsil eder.

Java'da neden kalıtım kullanılır?

4

- Yöntem Geçersiz Kılma için (böylece çalışma zamanı polimorfizmi elde edilebilir).
- Kod Yeniden Kullanılabilirliği için.

Kalıtımda kullanılan terimler

- **Sınıf:** Sınıf , ortak özelliklere sahip bir nesne grubudur. Nesnelerin oluşturulduğu bir şablon veya taslaktır.
- **Alt Sınıf / Çocuk Sınıf:** Alt sınıf, diğer sınıfı devralan bir sınıftır. Ayrıca türetilmiş sınıf, genişletilmiş sınıf veya alt sınıf olarak da adlandırılır.
- **Süper Sınıf / Üst / Ebeveyn Sınıf:** Süper sınıf, bir alt sınıfın özellikleri miras aldığı sınıftır. Buna temel sınıf veya üst sınıf da denir.
- **Yeniden kullanılabilirlik:** Adından da anlaşılacağı gibi, yeniden kullanılabilirlik, yeni bir sınıf oluşturduğunuzda varolan sınıfın alanlarını ve yöntemlerini yeniden kullanmanızı sağlayan bir mekanizmadır. Önceki sınıfta tanımlanmış olan alanları ve yöntemleri kullanabilirsiniz.

Java Kalıtım sözdizimi

6

- **Extends anahtar sözcüğü** , varolan bir sınıftan türetilen yeni bir sınıf yaptığınızı gösterir. "**Extends**" in anlamı işlevselliği arttırmaktır.
- Java terminolojisinde, devralınan bir sınıfa üst(parent, superclass), yeni sınıfa ise alt (child, subclass)sınıf denir.

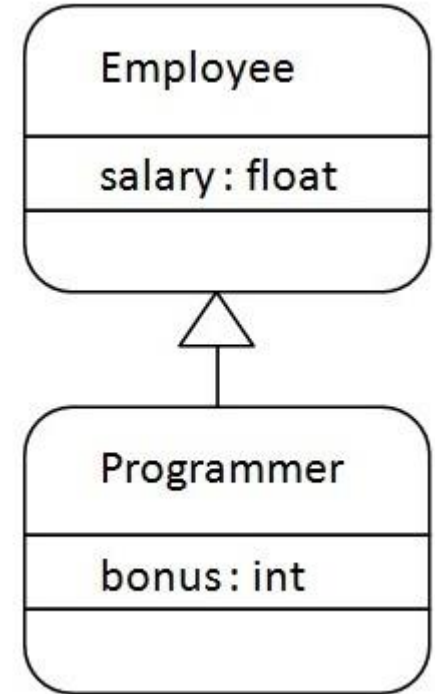
```
class Subclass-name extends Superclass-name
{
    //methods and fields
}
```

Java Kalıtım Örneği

7

MAUZEM

Yandaki şekilde görüldüğü gibi, programmer alt sınıf ve Employee üst sınıftır. İki sınıf arasındaki ilişki programmer **IS-A** Employee . Bu programmer'ın bir tür Employee olduğu anlamına gelir.



Java Kalıtım Örneği

8

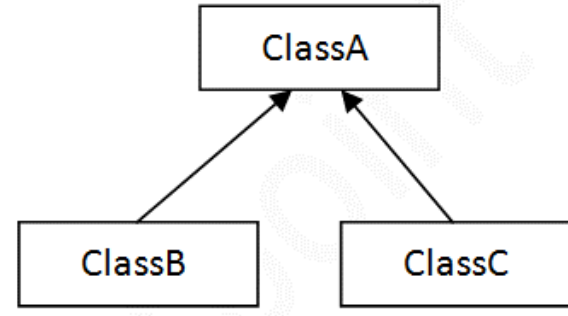
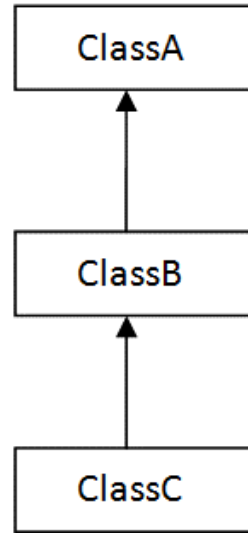
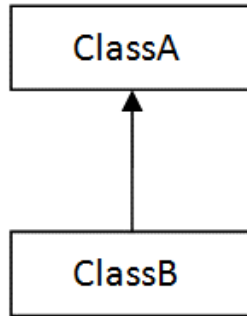
MAUZEM

```
class Employee{
    float salary=40000;
}
class Programmer extends Employee{
    int bonus=10000;
    public static void main(String args[]){
        Programmer p=new Programmer();
        System.out.println("Programmer salary is:"+p.salary);
        System.out.println("Bonus of Programmer is:"+p.bonus);
    }
}
```

```
Programmer salary is:40000.0
Bonus of programmer is:10000
```


Java'da kalıtım türleri

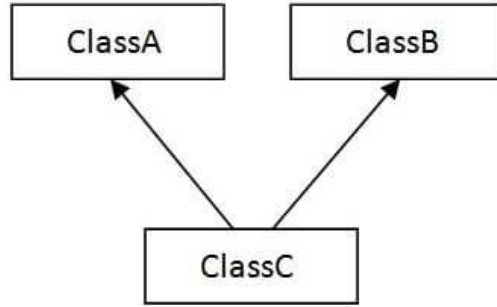
- Sınıf temelinde, java'da üç tür kalıtım olabilir: tek, çok düzeyli ve hiyerarşik.
- Java programlamasında, çoklu ve karma kalıtım yalnızca arayüz(interface) üzerinden desteklenir. Arayüzler hakkında daha sonra öğreneceğiz.



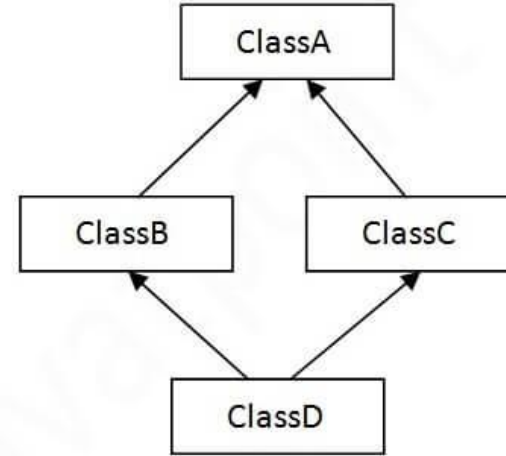
Java'da kalıtım türleri

10

- Bir sınıf birden çok sınıfı miras aldığında, birden çok kalıtım olarak bilinir.
- **Not: Birden fazla kalıtım Java ile sınıf üzerinden desteklenmez.**



4) Multiple



5) Hybrid

Tek Kalıtım Örneği

11

- Bir sınıf başka bir sınıfı miras aldığında, *tek bir miras* olarak bilinir. Aşağıda verilen örnekte, Dog sınıfı Animal sınıfını devralır, bu nedenle tek miras vardır.

```
class Animal{
    void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
    void bark(){System.out.println("barking...");}
}
class TestInheritance{
    public static void main(String args[]){
        Dog d=new Dog();
        d.bark();
        d.eat();
    }}

```

Output:

```
barking...
eating...
```

Çok Düzeyli Kalıtım Örneği

12

- Bir miras zinciri olduğunda, *çok düzeyli miras* olarak bilinir. Aşağıda verilen örnekte görebileceğiniz gibi, BabyDog sınıfı, yine Animal sınıfını devralan Dog sınıfını devralır, bu nedenle çok düzeyli bir miras vardır.

Output:

```
weeping...
barking...
eating...
```

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep(){System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

Hiyerarşik Kalıtım Örneği

13

- İki veya daha fazla sınıf tek bir sınıfı miras aldığında, *hiyerarşik kalıtım* olarak bilinir. Aşağıda verilen örnekte, Köpek ve Kedi sınıfları Animal sınıfını devralır, bu nedenle hiyerarşik kalıtım vardır.

Output:

```
meowing...
eating...
```

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}}
```

Java'da neden birden fazla kalıtım desteklenmiyor?

- Karmaşıklığı azaltmak ve dili basitleştirmek için java'da çoklu kalıtım desteklenmez.
- A, B ve C'nin üç sınıf olduğu bir senaryo düşünün. C sınıfı A ve B sınıflarını devralır. A ve B sınıfları aynı yönteme sahipse ve bunu alt sınıf nesnesinden çağırırsanız, A veya B sınıfı yöntemini çağırma belirsizliği olacaktır.
- Derleme zamanı hataları çalışma zamanı hatalarından daha iyi olduğundan, 2 sınıfı devralırsanız Java derleme zamanı hatası oluşturur. Aynı yöntem veya farklı bir yöntem olsun, derleme zamanı hatası olacaktır.

Örnek

15

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were

public static void main(String args[]){
    C obj=new C();
    obj.msg();//Now which msg() method would be invoked?
}
}
```

Compile Time Error

Java'da toplama (HAS-A) (Aggregation-münasebet)

16

- Bir sınıfın varlık referansı varsa, Toplama olarak bilinir. Toplama HAS-A ilişkisini temsil eder.
- *Employee* nesnesi id, name, email vb. Gibi birçok bilgiyi içerir.
- Aşağıda verilen şehir, eyalet, ülke, posta kodu gibi kendi bilgilerini içeren *address* adında bir nesne daha içerir.
- Bu durumda, *Employee* varlık referans adresi vardır, bu nedenle ilişki *Employee* HAS-A adresidir.

```
class Employee{  
    int id;  
    String name;  
    Address address;//Address is a class  
    ...  
}
```


Toplama neden kullanılır?

17

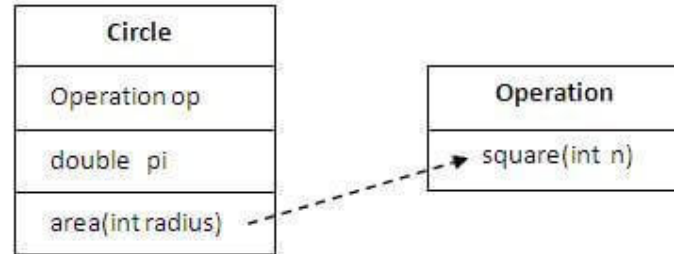
- Kod Yeniden Kullanılabilirliği için.

```
class Operation{  
    int square(int n){  
        return n*n;  
    }  
}
```

```
class Circle{  
    Operation op;//aggregation  
    double pi=3.14;
```

```
    double area(int radius){  
        op=new Operation();  
        int rsquare=op.square(radius);//code reusability (i.e. delegates the method call).  
        return pi*rsquare;  
    }  
}
```

```
public static void main(String args[]){  
    Circle c=new Circle();  
    double result=c.area(5);  
    System.out.println(result);  
}
```



Output : 78.5

Toplama ne zaman kullanılır?

18

- Kodun yeniden kullanımı sırasında en iyi ilişki kurulamazsa bu durumda ilişki toplama ile elde edilir.
- Kalıtım yalnızca ilişki is-a ilgili nesnelerin ömrü boyunca korunursa kullanılmalıdır; aksi takdirde, toplama en iyi seçimdir.

Emp.java

```
public class Emp {
    int id;
    String name;
    Address address;

    public Emp(int id, String name,Address address) {
        this.id = id;
        this.name = name;
        this.address=address;
    }

    void display(){
        System.out.println(id+" "+name);
        System.out.println(address.city+" "+address.state+" "+address.country);
    }

    public static void main(String[] args) {
        Address address1=new Address("gzb","UP","india");
        Address address2=new Address("gno","UP","india");

        Emp e=new Emp(111,"varun",address1);
        Emp e2=new Emp(112,"arun",address2);

        e.display();
        e2.display();

    }
}
```

Address.java

```
public class Address {
    String city,state,country;

    public Address(String city, String state, String country) {
        this.city = city;
        this.state = state;
        this.country = country;
    }
}
```

```
Output:111 varun
        gzb UP india
        112 arun
        gno UP india
```

Telif ve Kaynaklar

Kaynak gösterilmek şartıyla her türlü kullanıma açıktır.