NESNEYE YÖNELİK PROGRAMLAMA 26.10.2017

Yrd. Doç.Dr. Pelin GÖRGEL

İstanbul Üniversitesi
Bilgisayar Mühendisliği Bölümü

Kod 1

```
class Teacher {
   String designation = "Teacher";
   String collegeName = "Beginnersbook";
   void does(){
        System.out.println("Teaching");
   }
}

public class PhysicsTeacher extends Teacher{
   String mainSubject = "Physics";
   public static void main(String args[]){
        PhysicsTeacher obj = new PhysicsTeacher();
        System.out.println(obj.collegeName);
        System.out.println(obj.designation);
        System.out.println(obj.mainSubject);
        obj.does();
   }
}
```

Output:

```
Beginnersbook
Teacher
Physics
Teaching
```

Kod 2: Kalıtımda Constructor Overloading ve Super Kullanımı

```
class B{
    int xB=2;
  B(int i,int j) {
       //super(); Hiçbir etkisi yok
    System.out.println("B nin iki parametreli constr:"+i+"ve "+j); }

  B (int i){
        System.out.println("B nin tek parametreli constr:"+i);}

  B(){
     System.out.println("B nin boş constr.");}

    void metodB()
    {
        System.out.println("metod B:"+xB);
    }
}
```

Kod 2

```
class C extends B{
    int xC=1;
    int xB=4;
    C(int m,int n){
      //super(m); İstenirse böyle de kullanılır.
      super(m,n);
      System.out.println("C nin iki parametreli constr:"+m+" ve "+n); }
    C(int m){
     super(m);
      System.out.println("C nin tek parametreli constr:"+m); }
    C(int m,int n,int k){
     super();
      System.out.println("C nin üç parametreli constr ");}
    void metodC()
    {
        System.out.println("metod C"); }
   @Override
    void metodB()
    {
        System.out.println("C deki metod b:"+super.xB);
        super.metodB();   }
}
```

Kod 2

```
public class Test {
   public static void main(String args[])
   {
      B b1=new B(3,5);
      B b2=new B(7);
     C c1=new C(10,20);
      C c2=new C(6);
      C c3=new C(7,8,9);
     System.out.println(b1.xB);
     b1.metodB();
      System.out.println(c1.xB);
      c1.metodB();
   }
}
```

```
B nin iki parametreli constr:3ve 5

B nin tek parametreli constr:7

B nin iki parametreli constr:10ve 20
C nin iki parametreli constr:10 ve 20

B nin tek parametreli constr:6
C nin tek parametreli constr:6

B nin boş constr.
C nin üç parametreli constr

2

metod B:2

4

C deki metod b:2
metod B:2
```

Java'da Aggregation İlişkisi(Has a)

```
class Address
{
   int streetNum;
   String city;
   String state;
   String country;
   Address(int street, String c, String st, String coun)
   {
       this.streetNum=street;
       this.city =c;
       this.state = st;
       this.country = coun;  } }
class StudentClass
{
   int rollNum;
   String studentName;
   Address studentAddr;
   StudentClass(int roll, String name, Address addr){
       this.rollNum=roll;
       this.studentName=name;
       this.studentAddr = addr;
   }
```

Java'da Aggregation İlişkisi(Has a)

```
    public static void main(String args[]){
        Address ad = new Address(55, "Agra", "UP", "India");
        StudentClass obj = new StudentClass(123, "Chaitanya", ad);
        System.out.println(obj.rollNum);
        System.out.println(obj.studentName);
        System.out.println(obj.studentAddr.streetNum);
        System.out.println(obj.studentAddr.city);
        System.out.println(obj.studentAddr.state);
        System.out.println(obj.studentAddr.country);
    }
}
```

Output:

```
123
Chaitanya
55
Agra
UP
India
```

Neden Aggregation İlişkisi?

To maintain code re-usability. Suppose there are two other classes College and Staff along with above two classes Student and Address. In order to maintain Student's address, College Address and Staff's address we don't need to use the same code again and again. We just have to use the reference of Address class while defining each of these classes like:

- Student Has-A Address (Has-a relationship between student and address)
- College Has-A Address (Has-a relationship between college and address)
- Staff Has-A Address (Has-a relationship between staff and address)
- Hence we can improve code re-usability by using Aggregation relationship.

Kod Genişletilmek İstendiğinde

```
class College{
 String collegeName;
 //Creating HAS-A relationship with Address class
  Address collegeAddr;
 College(String name, Address addr){
this.collegeName = name;
 this.collegeAddr = addr;}}
class Staff{
   String employeeName;
   //Creating HAS-A relationship with Address class
   Address employeeAddr;
   Staff(String name, Address addr){
       this.employeeName = name;
       this.employeeAddr = addr;}}
```