Java

Görsel Programlama

Istanbul Arel Üniversitesi

Bahar 2023

Help class

- Class encapsulates functionality.
- Programın temel parçaları **sınıf**lardır.
- O yüzden her sınıfın tam olarak ne yapması gerektiği belli olmalı.
- Ancak sınıflar aynı zamanda kompakt olmalı. Çok kalabalıklaştırmamalıyız.

This

- Geçen hafta örnekte bunu getter ve setter'larda görmüştük.
- Kullanmak zorunda değiliz.
- Ancak eğer aynı değişken adını kullanacaksak, kullanmak o zaman daha iyi olur.

```
public void setId(int id) {
    this.id = id;
}
```

- Eğer bizim *private* değişken adımız *id* değilse, this kullanmasak da olur.

String operayonları

- boolean equals(str): diğer string ile eşit mi?
- int length(): uzunluk
- char charAt(idx): idx'teki karakter.
- int compareTo(str): Uzunluğu kısaysa *negatif sayı*, eşitse 0, uzunsa *pozitif sayı*.
- int indexOf(str)
- int lastIndexOf(str)

Örnek

```
String str1 = "java burada";
String str2 = new String(str1);
String str3 = "rastgele bir seyler";

int sonuc, idx;
char ch;

System.out.println("str1 uzunluğu: " + str1.length());

//karakter karakter yazdırlaım
for(int i=0;i<str1.length();i++){
    System.out.println(str1.charAt(i));
}

//eşit mi?
if(str1.equals(str3)){
    //eşit
}
else {
    //değil
}
```

Örnek > devam

```
idx = str1.indexOf("java");
System.out.println(idx);

str1 = "java burada java";
idx = str1.lastIndexOf("java");
System.out.println(idx);
```

Substring

```
String orijinal = "Görsel programlama mı Java mı?";
String substring = orijinal.substring(22,26);

System.out.println("Orijinal: " + orijinal);
System.out.println("Sub: " + substring);
```

Args

- Main fonksiyonu içerisinde `String[] args` yazıyor.
- Bunlar aslında komut istemcisinden java programını çalıştırırken verdiğimiz argümanlar.
- Şimdi bir program yazıp onu komut istemcisinden çalıştıralım.

```
class CLDemo {
    public static void main(String[] args) {
        System.out.println(args.length + " tane argüman var");
        System.out.println("Argümanlar: ");
        for(int i=0;i<args.length;i++){
            System.out.println("arg[" + i + "]: " + args[i]);
        }
    }
}
```

- `java CLDemo 1 2 3`

CLI

- Terminal üzerinde çalışan programlar için bu yukarıda gösterdiğimiz örnek çok kullanışlıdır.
- Rehber uygulaması yapmak isterseniz o zaman her seferinde programı çalıştırıp, menüden yapmak yerine bunları direkt olarak terminal üstünden yapabilirsiniz.
- `rehber ekle <isim> <telefon>`
- `rehber bul <isim>`
- ya da bir todo app gibi.
- `java todo ekle <gorev>`
- `java todo sil <id>` vb.

CLI > Örnek

```
String[][] rehber = { {"ahmet","000"}, {"ayse","111"}, {"mehmet","222"}, {"fatma","333"},};

if(args.length != 1) {
        System.out.println("Kullanım: java Phone <isim>");;
}
else
{
    for(int i=0;i<rehber.length;i++)
    {
            if(rehber[i][0].equals(args[0]))
        {
                        System.out.println("isim: " + rehber[i][0] + " telefon: " + rehber[i][1]);
                        break;
                }
        }
}
```

Var

- `var` keyword, tip belirlemeden değişken atamaya olanak sağlıyor.
- Otomatik olarak hangi tipte olduğunu kendisi anlıyor.
- `var x = 4` --> integer
- `var y = "selam"` --> String
- Dizilerde de kullanabiliyoruz.
  - `var myArray = new int[4];`
- Obje yaratırken de kullanabiliriz.
  - `var elma = new Elma()`
- For döngüsü içinde de kullanılabilir
  - `for(var i=0;i..)`

?

- If/Else yerine bazen bunu kullanırız. Özellikle tek satır işlemlerde çok daha güzel görünür ve hızlı yazılabilir. (ternary operator)

```
if(expr){
    //dogruysa
}
else {
    //degilse
}
```

- Exp 1 ? Exp 2 : Exp 3
- Exp 1'e bakılır, eğer Doğruysa, Exp2. Yanlışsa, Exp3.

```
int a = 4;
int sonuc;
sonuc = a == 4 ? 0 : 1;
System.out.println(sonuc); //0 gelir
```

Return tipleri

- Bir Error class'ı yazalım.

```
public class ErrorMsj {
    String[] mesajlar = {"output error","input error","disk full"};

    //method
    String getErrorMsj(int i) {
        if (i >= 0 & i < mesajlar.length) {
            return mesajlar[i];
        }
        else {
            return "geçersiz hata kodu";
        }
    }
}
```

Return > Örnek

- Main içerisinde de çağırabiliriz.

```
var err = new ErrorMsj();
System.out.println(err.getErrorMsj(0));
```

Return > Örnek

- Ancak, return olarak bir *obje* de verebilmek mümkün.

```
//Err.java

String msg;
int severity;

Err(String m, int s) {
    msg = m;
    severity = s;
}
```

Return > Örnek

```
//ErrorInfo.java

String[] mesajlar = {"output error","input error","disk full"};
int[] intensity = {3,3,2};

public Err getErrorInfo(int i){
    if(i >= 0 & i < mesajlar.length){
        return new Err(mesajlar[i], intensity[i]);
    }
    else {
        return new Err("hatali kod", 0);
    }
}
```

Return > Örnek

```
//Main.java
public static void main(String[] args) {
    var err = new ErrorInfo();
    Err e;
    e = err.getErrorInfo(2);
    System.out.println(e.msg + " önem: " + e.severity);
}
```

Inner class

- Main class'ın üstüne iki tane class tanımı yapalım. Bunların bir tanesi diğerinin içinde olsun.

```
class Outer {
    int x = 10;

    class Inner {
        int y = 5;
    }
}

public class Main {
    public static void main(String[] args) {
        Outer outer = new Outer();
        Outer.Inner inner = outer.new Inner();
        System.out.println(outer.x + " " + inner.y)
    }
}
```

Var args

- Eğer method'a ne kadar argüman vereceğimiz bilinmiyorsa o zaman bunları kullanırız.

```
static void vaTest(int ... v)  {
    System.out.println("argüman sayısı: " + v.length);
    System.out.println("içerik: ");
    for(int i=0;i<v.length;i++){
        System.out.println(v[i]);
    }
}
```

```
vaTest(1,2,3);
```

Varargs > devam

`static void vaTest(String msg, int ... v)` da mümkün.

Inheritance

- Nesne tabanlı programlamanın temellerinden biri.
- Java da **extends** keyword'u ile kullanılır.
- Class ve *subclass* var.
    - Parent - child ilişkisi.

Inheritance

- 2DSekiller isminde bir sınıf yaratalım. Bu aslında *genel* (generic) bir sınıf olacak. 2 boyutlu şekillerle ilgili, tüm 2 boyutlu şekillerin kullanabileceği bir sınıf.
- Her iki boyutlu şeklin ne özellikleri olur?
  - Uzunluk ve genişlik.

```
class IkiBoyutluSekil()
{
    double genislik;
    double uzunluk;

    void boyutGoster() {
        System.out.println("Uzunluk: " + uzunluk);
        System.out.println("Genişlik: " + genislik);
    }
}
```

Inheritance > Devam

- Şimdi, bir **Ucgen** sınıfı yaratalım ve bu sınıf bu yukarıda tanımladığımız `IkiBoyutluSekil` sınıfını *inherit* etsin. Orada da bir `alan` methodu tanımlayalım.

```
public class Ucgen extends IkiBoyutluSekil {

    double alan() {
        return uzunluk * genislik / 2;
    }
}
```

- Uzunluk ya da genislik diye değişkenleri tanımlamaya gerek kalmadı. **extends** ettiğimiz için onları alıyor zaten.

Inheritance > Devam

- Şimdi ise Main sınıfımıza gidip bir üçgen yaratalım.

```
Ucgen ucgen1 = new Ucgen();
ucgen1.genislik = 5;
ucgen1.uzunluk = 6;

ucgen1.boyutGoster();

double alanOut = ucgen1.alan();
System.out.println("Alan: " + alanOut);
```

- Normal şartlarda bunlara izin vermez, get/set kullanırız ancak kolaylık olsun diye bu şekilde yapıyoruz, unutmayalım.

Inheritance

- Gördüğümüz gibi Ucgen sınıfında **yeni** metodlar tanımlayabiliyoruz. Yeni değişkenler de ekleyebiliriz istersek.
- Inherit ettiğimizle sınırlı değiliz.
- Bir "Kare" sınıfı da yaratıp, orada yeni bir alan fonksiyonu oluşturarak orada da işlem yapabiliriz.
- Ancak, *inherit* ettiğimiz sınıfın *private* öğeleri varsa eğer, onları inherit etmiyoruz.
- Üçgen örneğindekileri **private** yapıp deneyelim.
  - Hepsi otomatik olarak hata verecek.
- Subclass'lar, *parent* class'ların **private** değişkenlerine erişemez.
  - Yine get/set kullanacağız.

Inheritance

- Bu sefer metodlar içerisinde diğer metodları kullanacağız.

```
double alan() {
    return getGenislik() * getUzunluk() / 2 ;
}
```

- O yüzden Sysout yerine `return` kullanıyoruz.

Inheritance

- Eğer subclass içinde bir Constructor kullanıyorsak, bu sefer de setGenislik(); komutlarıyla bunları oluşturuyoruz.
- Değişkenler *private* oldukları için gelmeyecekler. O yüzden Constructor eğer boş değilse, bunları o metodlara vererek oluşturacağız.
- Eğer hem *class* hem de *subclass* Constructor'lara sahipse işler biraz karışıyor.
  - O zaman **super** kullanacağız. İki kullanım şekli var.
  - İlki, superclass'ın constructor'ını çağırıyor.
  - İkincisi ise, subclass tarafından erişimi engellenmiş bir superclass elemanına ulaşmayı sağlıyor.

Super

- Şimdi biraz önceki örneği değiştirelim. Değişkenler private olsun ve get,set metodlarını tanımlayalım.

```
class TwoDShape {
    private double width;
    private double height;

    TwoDShape(double w, double h) {
        width = w;
        height = h;
    }

    double getWidth() { return width; }
    double getHeight() { return height; }
    void setWidth(double w) { width = w;}
    void setHeight(double h) { height = h;}

    void showDim() { //bilgi ver
        System.out.println("width: " + width + " height: " + height);
    }
}
```

Super > Devam

- Şimdi ise Ucgen sınıfını yaratalım ve *super* kullanımını görelim.

```
public class Ucgen extends TwoDShape {
    Ucgen(double w, double h) {
        super(w,h);
    }

    double area() {
        return getWidth() * getHeight() / 2;
    }
}
```

Super > Devam

- Main class'a dönelim ve deneyelim.

```
Ucgen ucgen = new Ucgen(4,5);
double sonuc = ucgen.area();
System.out.println(sonuc);
ucgen.showDim();
```

Super > devam

- Super komutunu boş değil, argümanla çalıştırdık. Ancak bu duruma göre değişiklik gösterebilir.
- Bunun nedeni *superclass* 'ın dolu constructor'ını çağırmamızdan dolayıydı.
- Şimdi superclass'a boş bir constructor ekleyelim ve onu çağıralım.

```
TwoDShape(double w, double h) {
    width = w;
    height = h;
}
TwoDShape() {
    width = height = 0.0;
}
```

Super > devam

- Şimdi Ucgen sınıfına gidip ona da bir boş constructor ekleyelim.

```
Ucgen(double w, double h) {
    super(w,h);
}

Ucgen() {
    super();
}
```

- Bu şekilde artık eğer `new Ucgen()` diye çağırırsak, height ve width 0 olacak.

Super > ikinci tür kullanım

- Bir örnekle anlamak daha kolay olacak.

```
class A {
    int i;
}

class B extends A {
    int i;

    B(int a, int b) {
        super.i = a;
        i = b;
    }

    void goster() {
        System.out.println("superclass i: " + super.i);
        System.out.println("subclass i: " + i);
    }
}

public class Main{
    public static void main(String[] args){
        B b = new B(1,2);
        b.show();
    }
}
```

Super > ikinci tür kullanım

- A ve B sınıfları yarattık. B, A'yı inherit etti. Ancak oradaki değişkenin aynısının ismini kullandı.
  - Bu aslında A'daki elemanı gizliyor. Artık "i" dediğimizde B'deki **i** anlaşılacak.
  - Eğer A'daki **i** ye erişmek istersek, `super.i` kullanmak gerekecek.

Abstract sınıflar ve metodlar

- Abstract sınıflar, *sınırlı* sınıflardır. Bu sınıfları kullanarak bir **obje** yaratamayız, ancak bu sınıfları **extend** edebiliriz (inherit).
  - Abstract metodlar yalnızca abstract sınıflar içerisinde tanımlanabilir ve **içeriği yoktur**.
  - Bu metodlar, abstract sınıfı inherit eden sınıflar tarafından **override** edilmelidir.
- Abstract sınıflar ya da **interface** ler ile uygulanır.
- Abstract sınıfların metodlarının içeriği olmaz. Dolayısıyla bu sınıf içinde bulunan metodların hepsi subclass'lar tarafından *override* edilmelidir.

Abstraction

Örneğin bir abstract sınıf yaratalım, içine bir tane abstract metod ve normal metod yerleştirelim.

```
public abstract class Hayvan {
    public abstract void ses();
    public void uyku() {
        System.out.println("zzz");
    }
}
```

- Şimdi main altında yeni bir Hayvan objesi yaratmak istersek hata verecek.
- `Hayvan hayvan = new Hayvan()`
- Çünkü tek başına obje yaratmak için yeterli değil.
    - Yeni bir class yaratıp *inherit* etmeliyiz.

Abstraction

- Yeni bir Kedi sınıfı yaratalım ve bu abstract sınıfı inherit etsin.

```
public class Kedi extends Hayvan {

}
```

- Ancak bu da tek başına yeterli değil, hata verecek.
  - Çünkü inherit ettiğimiz sınıf içerisinde bir abstract metod var.
  - Bunu override etmeliyiz.

```
public class Kedi extends Hayvan {
    public void ses() {
        System.out.println("meow");
    }
}
```

Abstraction

Şimdi Main içerisinde `Kedi kedi = new Kedi();` dedikten sonra **ses** metodunu çağırabiliriz.

Final

- Overriding ve inheritance yapmamıza engel olan bir keyword.
- Şu örneğe dikkat edelim.

```
class A {
    final void meth() {
        System.out.println("bu bir final metod");
    }

}

class B extends A {
    void meth() { //calismayacaktir
        System.out.println("olmaz");
    }
}
```

- meth() metodu **final** olarak tanımlandığı için, B tarafından override edilemez.

Final

- Metoda **final** diyebildiğimiz gibi, sınıflar için de bunu kullanabiliriz.

```
final class A {
    //..
}
class B extends A {
    //..
}
```

- Bu çalışmayacaktır, çünkü *inherit* edemeyiz **final** bir sınıfı.

Final

- Değişkenler için de kullanılabilir.
  - Bu aslında değişkenler için kullanıldığında **const** gibidir.
  - Yani değişmeyecek elemanlar, sabit elemanlar için kullanılır.
- Herhangi bir Java programında `final int x = 0` diye tanımladıktan sonra, bunu değiştirmeye çalışırsanız hata verecek ve size izin vermeyecektir.
- Ancak **private** gibi düşünmeyin. Yani bir metod da tanımlasanız, yine olmayacak.

Object Class

- yeterli

References

- Java - A beginner's Guide, 9th Edition