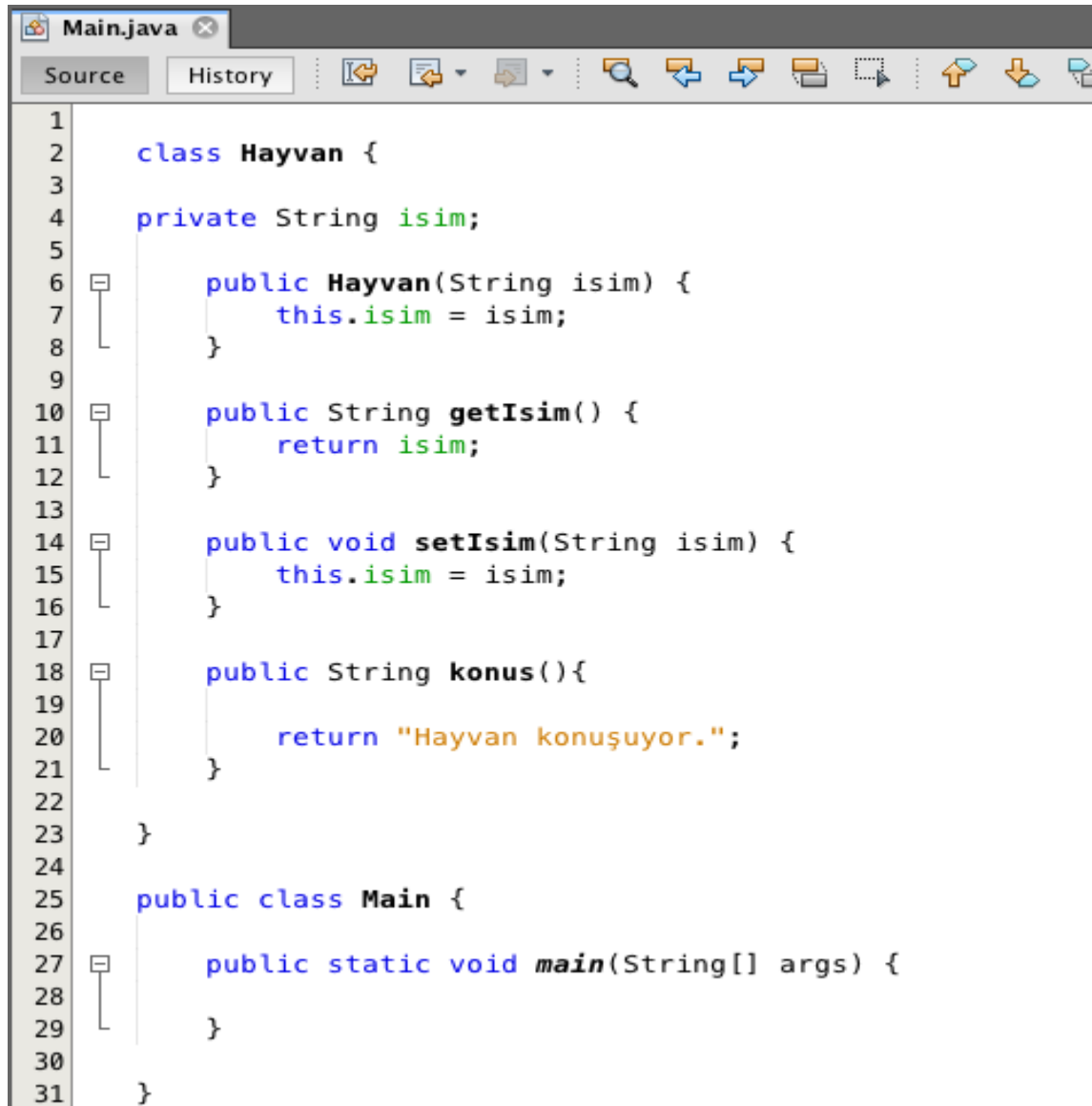


Polymorphism (Çok Biçimcilik) Örnek (Override-Yeniden Tanımlama): class Hayvan şeklinde classımızı yarattık ve isim özelliğini ekledik. Constructor ve getter setter metodlarımızı oluşturduk. Son olarak **konus()** adında bir metod yazdık.



```
1
2  class Hayvan {
3
4  private String isim;
5
6      public Hayvan(String isim) {
7          this.isim = isim;
8      }
9
10     public String getIsim() {
11         return isim;
12     }
13
14     public void setIsim(String isim) {
15         this.isim = isim;
16     }
17
18     public String konus(){
19
20         return "Hayvan konuşuyor.";
21     }
22 }
23
24 public class Main {
25
26     public static void main(String[] args) {
27
28     }
29
30 }
31 }
```

Şimdi Hayvan classından miras alan 3 tane class yazalım.

Kedi classı

Kedi classımız **Hayvan** classımızdan miras alacağı için **extends** yaptık. Constructorımızda oluşturduk. Ek olarak **konus()** metodumuzu **override** ettik.

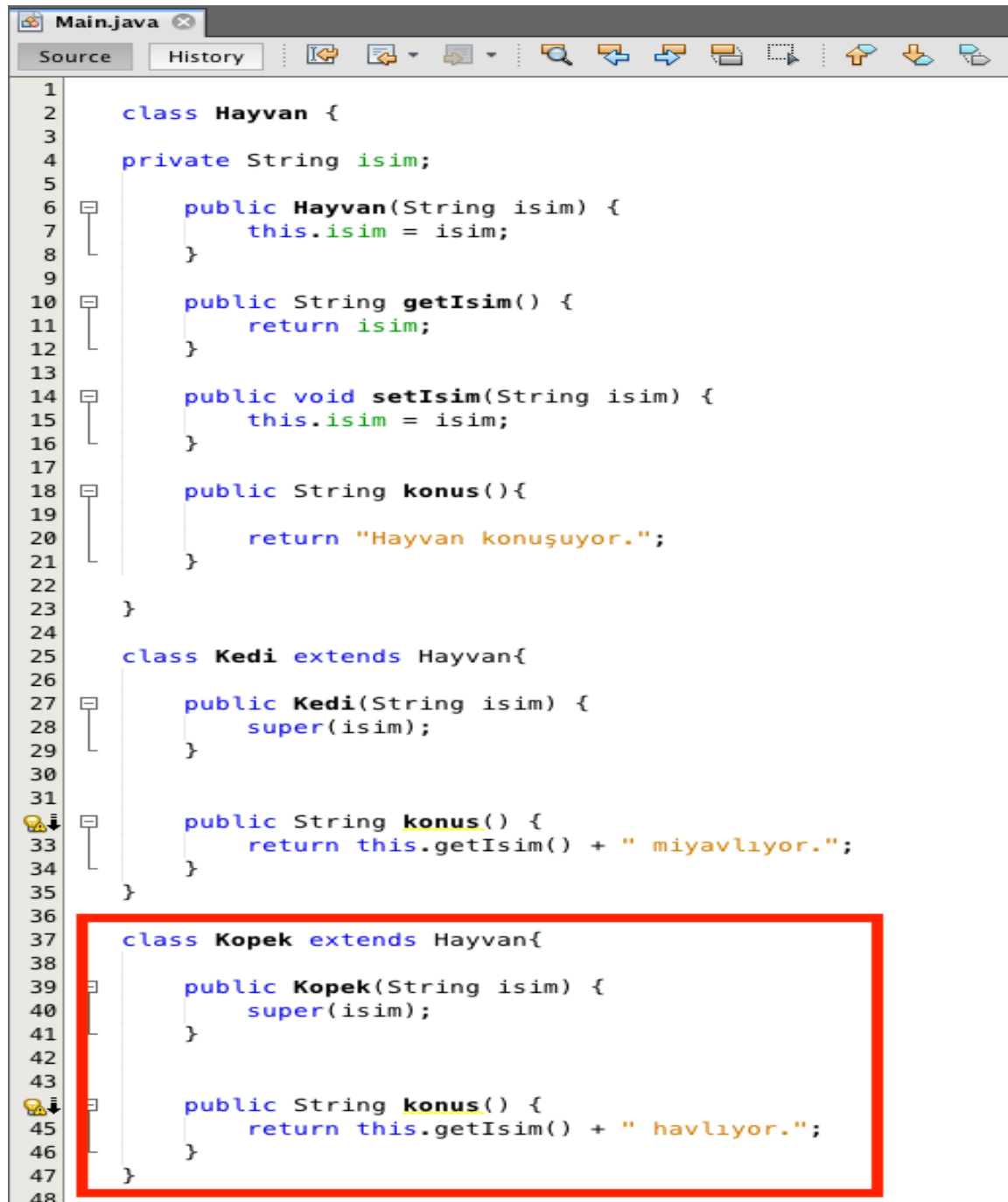
```
class Hayvan {  
    private String isim;  
  
    public Hayvan(String isim) {  
        this.isim = isim;  
    }  
  
    public String getIsim() {  
        return isim;  
    }  
  
    public void setIsim(String isim) {  
        this.isim = isim;  
    }  
  
    public String konus(){  
        return "Hayvan konuşuyor.";  
    }  
}  
  
class Kedi extends Hayvan{  
    public Kedi(String isim) {  
        super(isim);  
    }  
  
    public String konus() {  
        return super.konus();  
    }  
}
```

Ancak biz **Hayvan** classındaki metodu değilde kendi metodumuzu yazmak istiyoruz. Bu yüzden aşağıdaki gibi düzenliyoruz.

```
class Kedi extends Hayvan{  
    public Kedi(String isim) {  
        super(isim);  
    }  
  
    public String konus() {  
        return this.getIsim() + " miyavlıyor.";  
    }  
}
```

Şimdi Köpek classımızı yazalım

Constructorımızı oluşturduk. **konus()** metodunu override ettik.

The image shows a screenshot of a Java IDE window titled 'Main.java'. The window has a 'Source' tab and a 'History' tab. The code is written in Java and defines three classes: 'Hayvan', 'Kedi', and 'Kopek'. The 'Hayvan' class is the base class with a private attribute 'isim' and three methods: 'Hayvan(String isim)', 'getIsim()', and 'setIsim(String isim)'. The 'Kedi' class extends 'Hayvan' and overrides the 'konus()' method to return 'miyavlıyor.'. The 'Kopek' class also extends 'Hayvan' and overrides the 'konus()' method to return 'havlıyor.'. The 'Kopek' class is highlighted with a red rectangular border. The code is as follows:

```
1  class Hayvan {
2
3
4  private String isim;
5
6  public Hayvan(String isim) {
7      this.isim = isim;
8  }
9
10 public String getIsim() {
11     return isim;
12 }
13
14 public void setIsim(String isim) {
15     this.isim = isim;
16 }
17
18 public String konus(){
19     return "Hayvan konuşuyor.";
20 }
21
22 }
23
24 class Kedi extends Hayvan{
25
26 public Kedi(String isim) {
27     super(isim);
28 }
29
30
31
32 public String konus() {
33     return this.getIsim() + " miyavlıyor.";
34 }
35
36 }
37
38 class Kopek extends Hayvan{
39
40 public Kopek(String isim) {
41     super(isim);
42 }
43
44
45 public String konus() {
46     return this.getIsim() + " havlıyor.";
47 }
48 }
```

At sınıfımızı oluşturuyoruz. Yine aynı şekilde constructorımızı oluşturduk ve metot override yaptık.



```
1  class Hayvan {
2
3      private String isim;
4
5      public Hayvan(String isim) {
6          this.isim = isim;
7      }
8
9      public String getIsim() {
10         return isim;
11     }
12
13     public void setIsim(String isim) {
14         this.isim = isim;
15     }
16
17     public String konus(){
18
19         return "Hayvan konuşuyor.";
20     }
21 }
22
23 class Kedi extends Hayvan{
24
25     public Kedi(String isim) {
26         super(isim);
27     }
28
29     public String konus() {
30         return this.getIsim() + " miyavlıyor.";
31     }
32 }
33
34 class Kopek extends Hayvan{
35
36     public Kopek(String isim) {
37         super(isim);
38     }
39
40     public String konus() {
41         return this.getIsim() + " havlıyor.";
42     }
43 }
44
45 class At extends Hayvan{
46
47     public At(String isim) {
48         super(isim);
49     }
50
51     public String konus() {
52         return super.konus();
53     }
54 }
55
```

Şimdi artık polymorphismi görebiliriz.

Normalde main classı içerisinde bu zamana kadar objeyi hep aşağıdaki gibi tanımlamıştık.

```
57 public class Main {
58
59     public static void main(String[] args) {
60
61         Hayvan hayvan = new Hayvan("Richi");
62         System.out.println(hayvan.konus());
63     }
64 }
65
66 }
```

Output - Polymorphism (run) x

run:
Hayvan konuşuyor.
BUILD SUCCESSFUL (total time: 0 seconds)

Fakat şimdi hiç görmediğimiz bir tanımla şeklini göreceğiz. Aşağıdaki gibi tanımladığımızda javanın hata vermediğini görüyoruz.

```
public class Main {
    public static void main(String[] args) {
        Hayvan hayvan = new Kedi("Mikedi");
        System.out.println(hayvan.konus());
    }
}
```

Output - Polymorphism (run) x

run:
Mikedi miyavlıyor.
BUILD SUCCESSFUL (total time: 0 seconds)

İşte polymorphism bu aslında. Bir tane referans birden farklı obje gibi davranıyor. Yani bizim hayvan referansımız var ancak biz buna Kedi referansı atadık.

Polymorphism yapmak için şu şartların olması gerekir.

1-) Ana classınız olacak. Bizim bu örnekteki ana classımız Hayvan classıydı.

2-) Geri kalan classlar alt sınıf olacak yani Hayvan classından miras alacaklar.

Biz bir objeyi birden fazla obje gibi kullanabiliyoruz. Ama bunun için mutlaka inheritance şeklinde bir yapınızın olması gerekiyor.

```
public class Main {  
    public static void main(String[] args) {  
        Hayvan hayvan = new Kedi("Mikedi");  
        Hayvan hayvan1 = new Kopek("Richi");  
        Hayvan hayvan2 = new At("Monyak");  
  
        System.out.println(hayvan.konus());  
        System.out.println(hayvan1.konus());  
        System.out.println(hayvan2.konus());  
    }  
}
```

Main

Output - Polymorphism (run)

run:
Mikedi miyavlıyor.
Richi havlıyor.
Hayvan konuşuyor.
BUILD SUCCESSFUL (total time: 0 seconds)

Kedi , Kopek ve At classlarından **hayvan , hayvan1 ve hayvan2** referansları oluşturduk.

Mantık olarak şu şekilde işliyor.

hayvan1 e biz **kopeği** atıyoruz. Ve yukarıdan başlıyor. **konus()** metodum var mı diye bakıyor. Bana Kopek referansı atanmıştı. Kopeğin içinde bakıyor **konus()** metodu var mı? Varsa onu yoksa kendi metodunu çağırıyor.

super () methodu örnek: Super anahtar sözcüğü bir üst sınıftan miras alan bir sınıfın üst sınıfın metotlarını, özelliklerini alt sınıftan çağırmamıza ve kullanmamıza olanak sağlar.

Super metodu ile üst sınıfın kurucu metodunu çağırma

Super anahtar sözcüğünü üst sınıfın yapıcı metodunu çağırmak istediğimizde kullanırız . Bunun için kendi yapıcı metodumuz içerisinde **super()** olarak çağırırız .

İki adet sınıfımız olsun bunlar oyuncu ve esporOyuncusu sınıfları, esporOyuncusu sınıfı oyuncu sınıfından miras alsın.

Oyuncu sınıfı :

```
public class oyuncu {  
  
    private int yas = 20;  
  
    public oyuncu () {  
        System.out.println("Oyuncu sınıfı");  
    }  
}
```

esporOyuncusu sınıfı :

```
public class esporOyuncusu extends oyuncu {  
    public esporOyuncusu() {  
        System.out.println("Espor oyuncu sınıfı");  
    }  
}
```

Evet gördüğümüz gibi sınıflar bu şekilde . Herhangi bir super() metot çağırısı yok . Ama gelin bir tane esporOyuncusu nesnesi oluşturalım.

```
public static void main(String arr[]) {  
  
    esporOyuncusu esporcu = new esporOyuncusu();  
}
```

Çıktı:

Oyuncu sınıfı
Espor oyuncu sınıfı

Gördüğümüz gibi herhangi bir üst sınıf kurucu metodu çağırısı yapmadan üst sınıfın kurucu metodu da çalıştı . Bunun nedeni Java'nın arka planda kendisi bunu zorunlu kılması .

Eğer üst sınıfın kurucu metodu parametrelili olsaydı , bizim de parametrelili bir `super()` çağırısı yapmamız gerekecekti .

Örneğin üst sınıfın kurucu metodunu parametre ister şekilde güncelersek :

```
public oyuncu (String zorunluParametre){  
    System.out.println("Oyuncu sınıfı");  
}
```

Evet String tipinde bir adet parametre aldık . Ve miras alan sınıfımıza döndüğümüzde şöyle bir hata alacağız:

There is no default constructor available in 'com.company.oyuncu'

Bu hatanın nedeni alt sınıftaki default olarak oluşturulan `super` metodunun işe yaramadığıdır. Bunun iki çözümü vardır :

a) Alt sınıftaki `super` çağırısına üst sınıfta istenen parametreleri yolamak .

```
public esporOyuncusu() {  
    super("Parametre eklendi");  
    System.out.println("Espor oyuncu sınıfı");  
}
```

b) Üst sınıfta bir adet parametresiz kurucu metot oluşturmak .

```
private int yas = 20;  
  
public oyuncu(){  
    // Parametresiz kurucu metot  
}  
  
public oyuncu (String zorunluParametre){  
    System.out.println("Oyuncu sınıfı");  
}
```

Super metodu ile üst sınıfın metotlarını kullanma

Miras alınan sınıfın metotlarını kullanmak istediğimiz zamanda da `super` çağırısı yaparız . Genelde bu çağırısı üst sınıfın bir metotunu `override` ettiğimizde kullanırız . Bu kullanım sayesinde metodu kendimize göre uyarlamış oluruz ve yukarıdaki aynı isimdeki metotun yaptığı işi tekrar yazmamak amacıyla metot içerisinde tekrar çağırabiliriz .

```

public class oyuncu {

    private int yas = 20;

    void kos(){
        System.out.println("Oyuncu koşuyor");
    }

}

class esporOyuncusu extends oyuncu {
    void enerjiAl(){

    }
    void kos(){
        enerjiAl();
        // Üst sınıftaki metot burada çağrılıyor .
        super.kos();
    }

}

```

Görüldüğü gibi esporOyuncusu sınıfında kos() diye bir metot kullandık ve bu metodu üst sınıftaki metodu override ederek kullandık ama üst sınıftaki metotun işlemlerini de ihtiyaç duyduğumuz için gelip o metodu super çağrısı ile çağırdık .

Super metodu ile üst sınıfın değişkenlerini kullanma

Miras alınan sınıftaki değişkenlere erişmek istediğimiz zaman da super metodunu kullanırız . Aşağıdaki örneği inceleyebilirsiniz :

```

public class oyuncu {
    int yas = 20;
}

class esporOyuncusu extends oyuncu {
    int yas = 40;

    void yasYazdir(){
        // esporOyuncu sınıfındaki yas değerini yazdırır.
        System.out.println(yas);
        // oyuncu sınıfındaki yas değerini yazdırır .
        System.out.println(super.yas);
    }

}

class calistir{

    public static void main (String args[]){
        esporOyuncusu oyuncumuz = new esporOyuncusu();
        oyuncumuz.yasYazdir();
        // Çıktı : 40 20
    }

}

```