TECHPROED

PROFESSIONAL TECHNOLOGY EDUCATION

WELCOME TO TECHPROED JAVA TUTORIAL

Testi baslatmak icin asagidaki adımları takip ediniz

Go to www.socrative.com

Click on Login

Click on Student Login

***Room Name*: A L P T E K I N 3 5 2 3**

Kayıtta kullandığınız ismi tam olarak yazınız

Time: 11 Minutes

TECHPROED
PROFESSIONAL TECHNOLOGY EDUCATION

Method Signature

Method signature method name ve parameter list'den oluşur.

1) Parametre'lerin sayısını artırarak veya azaltarak iki farklı method elde edebilirsiniz.

```
public String add(String a, int b){
    String c = a+b;
    return c;
}
private int add(String y, int x){
    int c = x;
    return c;
}
public static void main(String args[]){

}
```

```
class Example {

    public String add(String a, int b){
        String c = a+b;
        return c;
    }
    private int add(int y){
        int c = y;
        return c;
    }
    public static void main(String args[]){

    }
}
```

TECHPROED
PROFESSIONAL TECHNOLOGY EDUCATION

2) Farklı data type'ındaki parametre'lerin yerlerini değiştirirseniz farklı bir method elde edersiniz.

```
class Example {

    public String add(String a, int b){
        String c = a+b;
        return c;
    }
    public int add(String y, int x){
        int c = x;
        return c;
    }
    public static void main(String args[]){

    }
```

```
class Example {

    public String add(String a, int b){
        String c = a+b;
        return c;
    }
    public int add(int x, String y){
        int c = x;
        return c;
    }
    public static void main(String args[]){

    }
```

TECHPROED
PROFESSIONAL TECHNOLOGY EDUCATION

3) Method'un ismini değiştirirseniz farklı bir method elde edersiniz.

```java
class Example {

    public int add(int b){
        int c = b;
        return c;
    }
    private double add(int y){
        int c = y;
        return c;
    }
    public static void main(String args[]){

    }
}
```

```java
class Example {

    public int add(int b){
        int c = b;
        return c;
    }
    private double addChanged(int y){
        int c = y;
        return c;
    }
    public static void main(String args[]){

    }
}
```

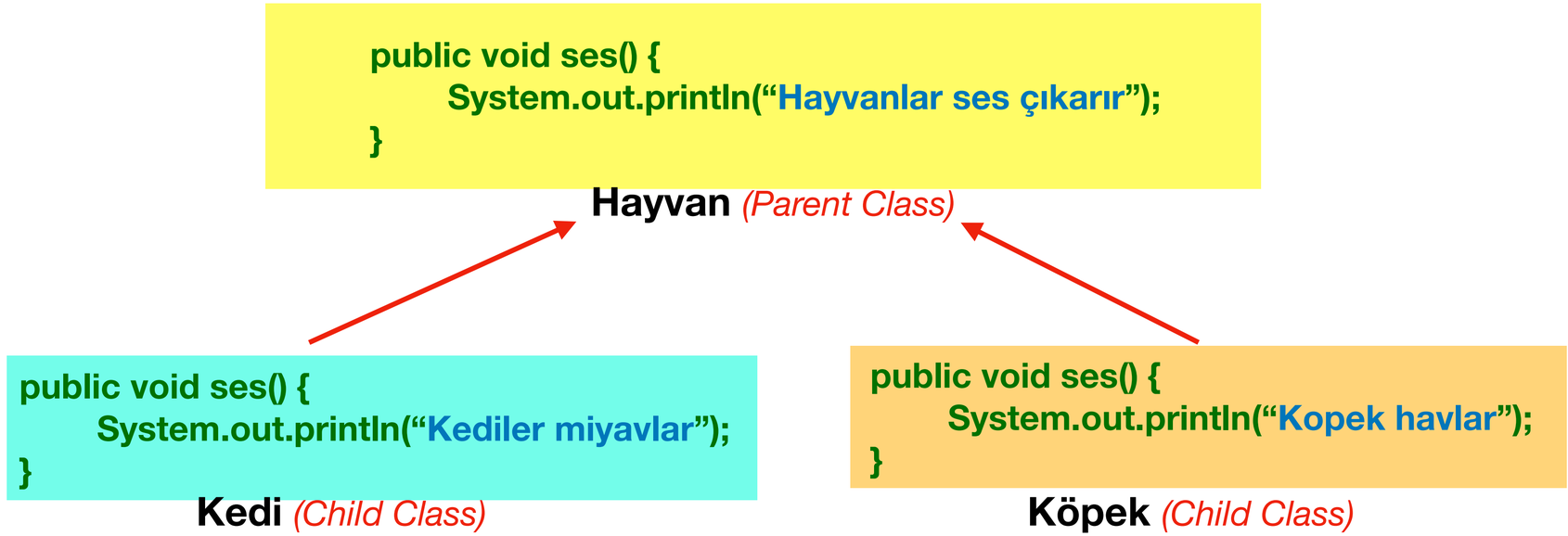
TECHPROED
PROFESSIONAL TECHNOLOGY EDUCATION

Method Overriding nedir ?

1) Parent class'da varolan bir methodu method signature'i değiştirmeden, gerekirse method body'sini değiştirerek kullanmaya Method Overriding denir.

2) Overriding kullanarak, child class'in parent class'daki methodu kendine uyarlayarak kullanmasını sağlamış oluruz.

3) Overriding yapıldığında parent class'daki methoda Overridden Method, child class'daki methoda Overriding Method denir.



TECHPROED
PROFESSIONAL TECHNOLOGY EDUCATION

Overriding and Overriden Methods

```java
public class Animal {

    public void eat(){
        System.out.println("Animals eat meat and vegetable");
    }

    public static void main(String[] args) {

        Animal animal = new Animal();
        animal.eat();

    }
}
```

Overriden Method

```java
public class Dog extends Animal {

    public void eat(){
        System.out.println("Dogs eat meat");
    }

    public static void main(String[] args) {

        Dog dog = new Dog();
        dog.eat();

    }
}
```

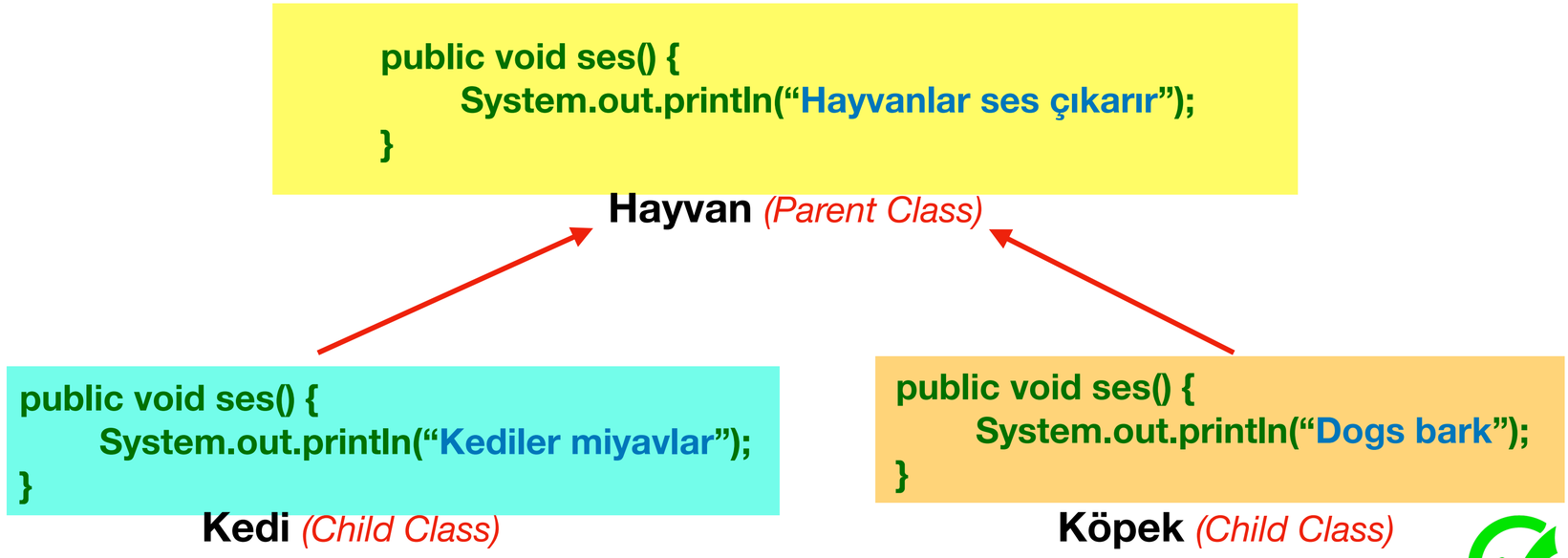
```java
public class Lamb extends Animal {

    public void eat(){
        System.out.println("Lambs eat grass");
    }

    public static void main(String[] args) {

        Lamb lamb = new Lamb();
        lamb.eat();

    }
}
```

Overriding Method

Method Overriding'in Faydaları

Overriding parent class'daki method'u değiştirmeden child class'ın kendine uygun method üretmesini sagla

Aşağıdaki kodda görüldüğü gibi parent class'daki ses() methodu değiştirilmedi.
Kedi ve Köpek class'larda ses() methodu o class'lara uyarlanarak kullanıldı.



TECHPROED
PROFESSIONAL TECHNOLOGY EDUCATION

Overriding Kuralları

1) **Method Signature'ı** *(name and parameters)* **değiştirmeyin**

2) **Child class'daki method'un** (overriding method) **Access Modifier'ı parent class'daki method'un** (overridden
Modifier'ından daha dar olamaz.

3) **Overriding method covariant return type kullanmalıdır.**

4) **private, static and final method'lar overriding yapılamazlar.**

5 ve 6 sonra açıklanacak

5) **Child class'daki method** (overriding method), **parent class'daki method'un** (overridden method) **compile tim
edip etmediğine bakmaksızın compile time exception throw edebilir.**
Fakat child class'daki method (overriding method), **parent class'daki method'dan** (overridden method) **daha
bir run time exception throw edemez.**

6) **Eğer abstract olmayan bir class abstract class'a extend ediyorsa veya bir interface'i implement ediyorsa
method'ları override**

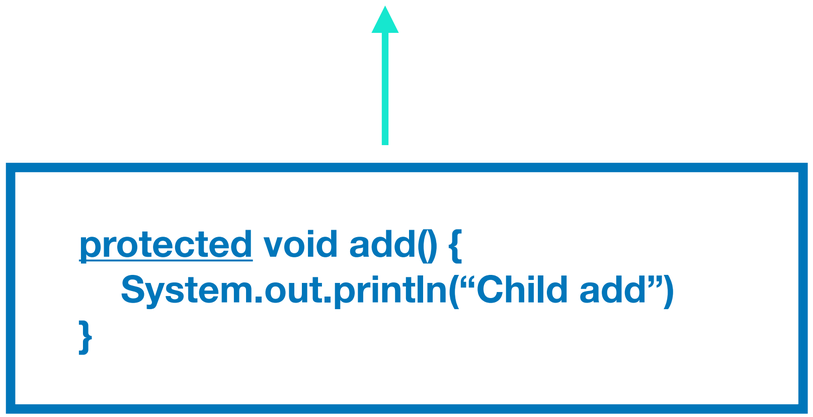
Access modifiers in “Overriding”

False

Child parent’ı sınırlayamaz

```
public void add() {
    System.out.println(“Parent add”)
}
```

Parent



```
protected void add() {
    System.out.println(“Child add”)
}
```

Child extends Parent

True

Child’ın access modifier’ı parent’ın access modifier’ı ile aynı veya daha geniş olmalıdır.

```
protected void add() {
    System.out.println(“Parent add”)
}
```

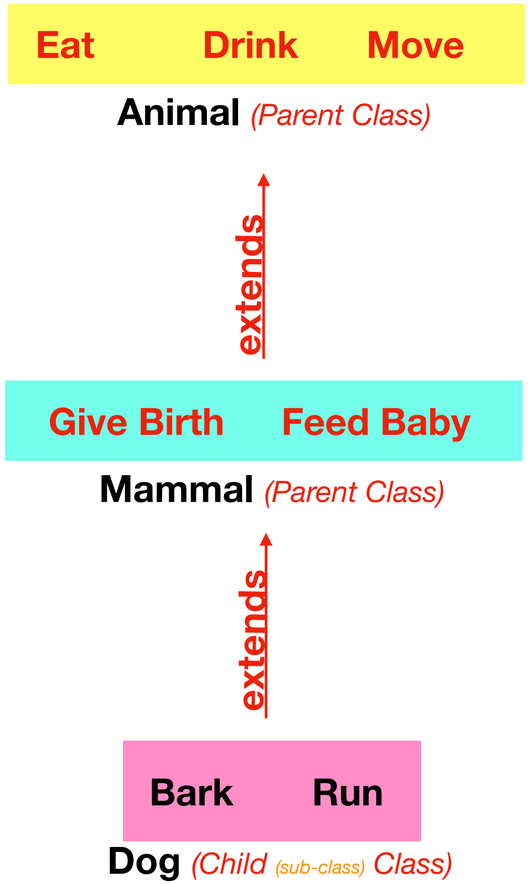
Parent

```
protected / public void add() {
    System.out.println(“Child add”)
}
```

Child extends Parent

TECHPROED
PROFESSIONAL TECHNOLOGY EDUCATION

IS-A (↑) and HAS-A (↓) Relationship



Dog IS-A Mammal ==> True

Dog IS-A Animal ==> True

Mammal IS-A Animal ==> True

Animal IS-A Dog ==> False

Animal IS-A Mammal ==> False

Mammal IS-A Dog ==> False

Mammal HAS-A Dog ==> True

Animal HAS-A Mammal ==> True

Animal HAS-A Dog ==> True

Dog HAS-A Animal ==> False

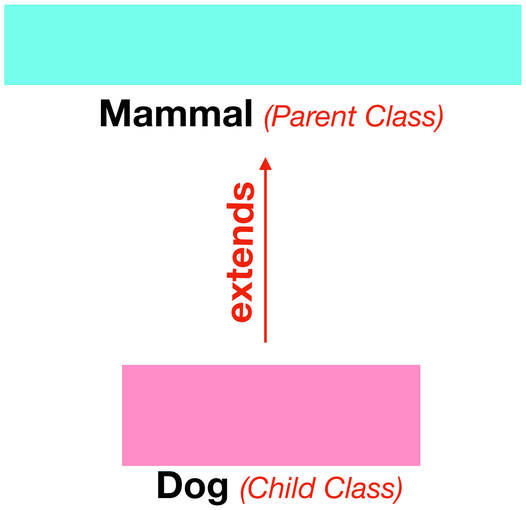
Mammal HAS-A Animal ==> False

Return Types in “Overriding”

There must be IS-A relationship between the return types when you look at from overriding method(child) to overridden method(parent)

When you override a method, return type of overriding method *(method inside child)* **must be sub-class** *(child class)* **of return type of overridden method** *(method inside parent)*

Return Types



Overriding

```
public Mammal eat() {
    System.out.println(“Eat when you are hungry”)
}
```

Parent

```
public Dog eat() {
    System.out.println(“Eat less”)
}
```

Child extends Parent

TECHPROED
PROFESSIONAL TECHNOLOGY EDUCATION

Overriding Hakkında Sorular…

1)

```
class Derived {

    public void getDetails(String temp){
        System.out.println("Derived class " + temp);
    }
}

public class Test extends Derived {

    public int getDetails(String temp){
        System.out.println("Test class " + temp);
        return 0;
    }

    public static void main(String[] args){
        Test obj = new Test();
        obj.getDetails("GFG");
    }
}
```

a) Derived class GFG

b) Test class GFG

c) Compilation error

d) Runtime error

Overriding yapılan method'larin return type'lari ya ayni olmalıdır veya is-a relation olmalıdır. Compile Time Error verir

2)

```
class Derived{

    public void getDetails(){
        System.out.println("Derived class");
    }
}

public class Test extends Derived{
    protected void getDetails(){
        System.out.println("Test class");
    }

    public static void main(String[] args){
        Derived obj = new Test(); // line xyz
        obj.getDetails();
    }
}
```

a) Test class

b) Compilation error due to line xyz

c) Derived class

d) Compilation error due to access modifier

Child class'daki method'larin access modifier'lari ayni olmalıdır veya daha geniş olmalıdır.
Compile Time Error verir access modifier icin

3)

```
class Derived{

    public void getDetails(){
        System.out.printf("Derived class ");
    }
}

public class Test extends Derived{
    public void getDetails(){
        System.out.printf("Test class ");
        super.getDetails();
    }

    public static void main(String[] args){
        Derived obj = new Test();
        obj.getDetails();
    }
}
```

a) Test class Derived class

b) Derived class Test class

c) Compilation error

d) Runtime error

Child class'daki getDetails() methodu calisir ve ekrana "Test class" yazdırır. Sonra super keyword ile parent class'daki getDetails() method cagrilir ve ekrana Derived class yazdırır.

4)

```
class Derived
{
    protected final void getDetails()
    {
        System.out.println("Derived class");
    }
}

public class Test extends Derived
{
    protected final void getDetails()
    {
        System.out.println("Test class");
    }
    public static void main(String[] args)
    {
        Derived obj = new Derived();
        obj.getDetails();
    }
}
```

a) Derived class

b) Test class

c) Runtime error

d) Compilation error

final method'lar override yapılamaz. Compile Time Error verir.

“Polymorphism” nedir?

Polymorphism *(çok şekillilik)* **bir methodu farklı yapılarda kullanabilmemizi sağlar.**

Polymorphism = Overloading + Overriding

```java
public class Animal {

    public void eat(){
        System.out.println("Animals eat meat and vegetable");
    }

    public static void main(String[] args) {

        Animal animal = new Animal();
        animal.eat();

    }
}
```

```java
public class Dog extends Animal {

    public void eat(){
        System.out.println("Dogs eat meat");
    }

    public static void main(String[] args) {

        Dog dog = new Dog();
        dog.eat();

    }
}
```

```java
public class Lamb extends Animal {

    public void eat(){
        System.out.println("Lambs eat grass");
    }

    public static void main(String[] args) {

        Lamb lamb = new Lamb();
        lamb.eat();

    }
}
```

“Polymorphism” Çeşitleri

1) Method Overloading bir compile time (static) polymorphism’dir.

Method Overloading sayesinde aynı isme, aynı body’e, farklı parametrelere sahip bir çok method üretip kullanabiliriz.

2) Method Overriding bir run time (dynamic) polymorphism’dir.

Method Overriding sayesinde aynı isme, aynı parametrelere’e, farklı body’e sahip bir çok method üretip kullanabiliriz.

Overloading ve Overriding Farklari

1) Overloading’de method signature degisir, Overriding’de degismez.

2) Overloading’de body istenirse değiştirilebilir, Overriding’de body %99 değiştirilir.

3) final, static ve private methodlar Overload edilebilir, ama Override edilemezler.

4) Overloading Compile Time Polymorphism (static)’dir,
Overriding is Run Time Polymorphism’(dynamic)dir.

5) Overloading’de inheritance gerekmez, Overriding’de gerekir.

6) Overloading’de istediğimiz sekilde access modifier ve return type kullanabiliriz ama Overriding’de access modifier ve return type kullanma belli kurallara baglidir.