

به نام خدا



برنامه‌سازی پیشرفته

دانشگاه شهید بهشتی . دانشکده مهندسی و علوم کامپیوتر

دکتر مجتبی وحیدی اصل

وراثت

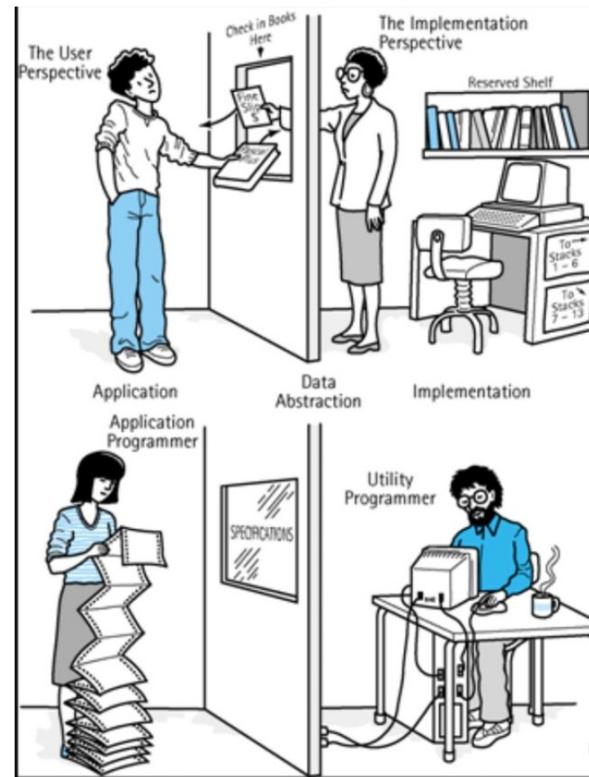
فهرست مطالب

1. انتزاع و کپسوله بندی
2. پلی مورفیسم
3. ارثبری
4. انواع ارث بری
5. زیرکلاس ها و ابرکلاس ها
6. کلمه کلیدی SUPER

انتزاع (Abstraction)

انتزاع چیست؟

یکی از مزایای شیء گرایی انتزاع است، به پنهان کردن جزئیات داخلی از چشم کاربران، **انتزاع (abstraction)** گفته می شود.
کاربر باید تنها کارکرد و خروجی وسیله را مطابق با نیاز های خویش بداند.





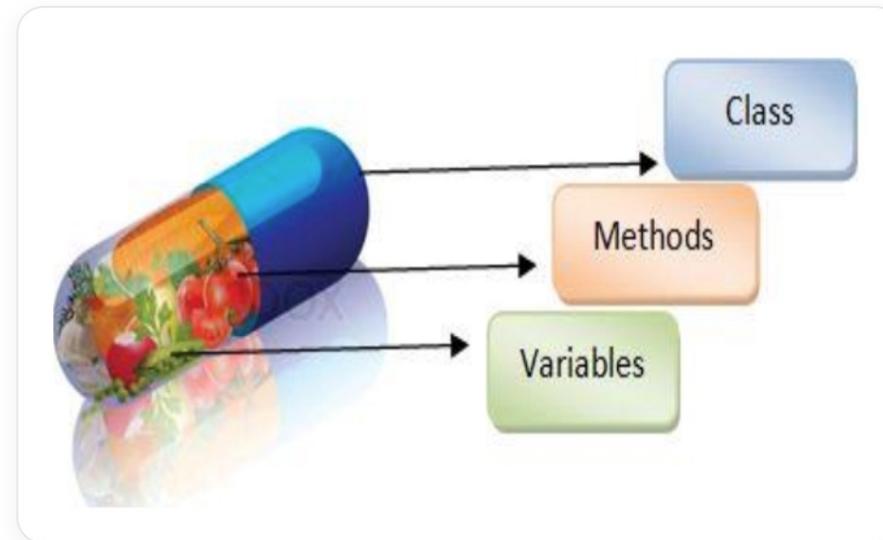
:نکته

در جاوا برای رسیدن به انتزاع، از کلاس **interface** و **abstract** استفاده میکنیم.

متدى که برای همه اشیاء یک کلاس وجود دارد اما جزئیات دقیق و پیاده سازی آن، در خود کلاس غیر ممکن بوده و در هر زیرکلاس پیاده سازی میشود را متد انتزاعی میگویند.

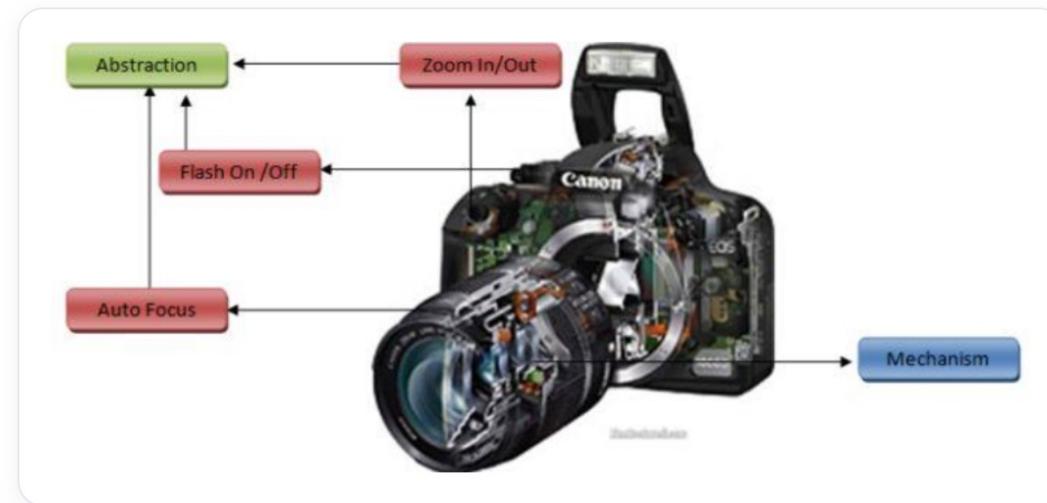
کپسوله بندی (encapsulation)

کپسوله بندی چیست؟ ترکیب و بسته بندی داده ها و متدها در یک واحد به نام شیء، اصطلاحا کپسوله بندی گفته میشود. به عنوان مثال هر کلاس در جاوا، یک کپسوله بندی است.



تفاوت کپسوله بندی و انتزاع

یک دوربین عکاسی دیجیتال مانند تصویر زیر در نظر بگیرید:



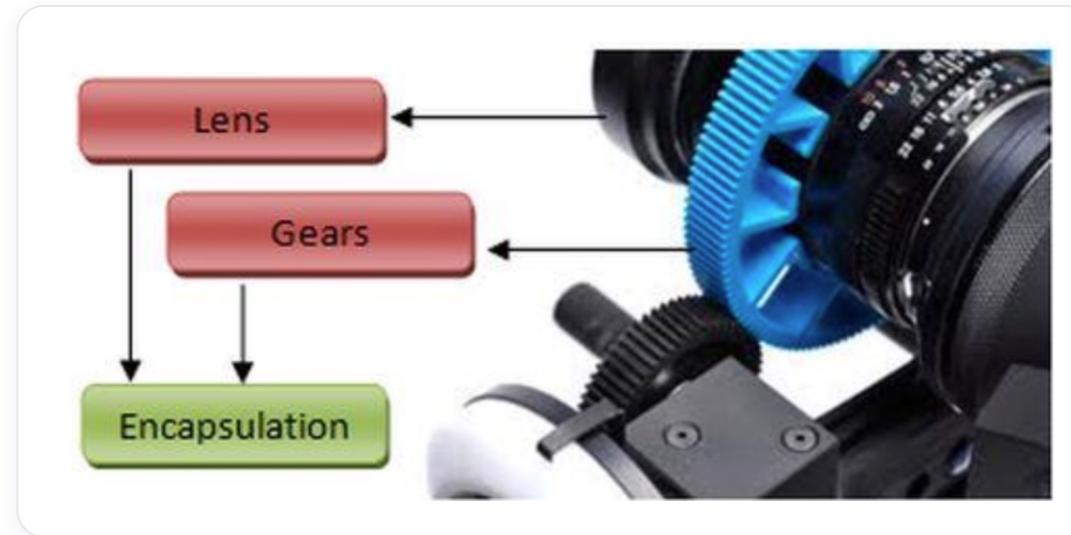
بر روی دفترچه راهنمای این دوربین نوشته شده است:

"کاربر محترم، در هنگام استفاده از دوربین دیجیتال کافیست بر روی دکمه های zoom in و zoom out کلیک کنید و در هنگام زوم شدن دوربین حرکت لنز را حس خواهید نمود. "

حال اگر دوربین را باز کنید با مکانیزم پیچیده آن رویرو خواهید شد که برای شما قابل فهم نخواهد بود!

فشردن دکمه عکاسی و گرفتن عکس (نتیجه دلخواه) **انتزاع** نامیده میشود.

همانطور که در تصویر زیر مشخص است، زمانی که ما بر روی دکمه zoom in/out کلیک میکنیم، در درون ساختار دوربین مکانیزم هایی شامل چرخ دنده ها و لنز ها، این خواسته مارا برآورده میکنند:



به ترکیب این چرخ دنده ها و لنز ها اصطلاحا **کپسوله بندی** گفته میشود که به عکاس امکان می دهد زومینگ را به نرمی و سادگی انجام دهد.

پس میتوان نتیجه گرفت: انتزاع از طریق کپسوله بندی امکان پذیر شده است. در واقع انتزاع جنبه طراحی مسئله را انجام میدهد و کپسوله بندی مسئول پیاده سازی مسئله است.

چند ریختی (Polymorphism)

پلی مورفیسم یا چند ریختی چیست؟ پلی مورفیسم یعنی کاری به شکل ها یا شیوه های مختلفی انجام شود. به عنوان مثال "صحبت کردن" در بین همه حیوانات مشترک است، اما هر حیوانی گویش و صدای خاص خودش را دارد:



در جاوا برای رسیدن به پلی مورفیسم از باز نویسی متده overriding استفاده میکنیم.

نکته: در این مثال متده "صحبت کردن" برای کلاس حیوان انتزاعی **abstract** میباشد.

ارثبری (inheritance)

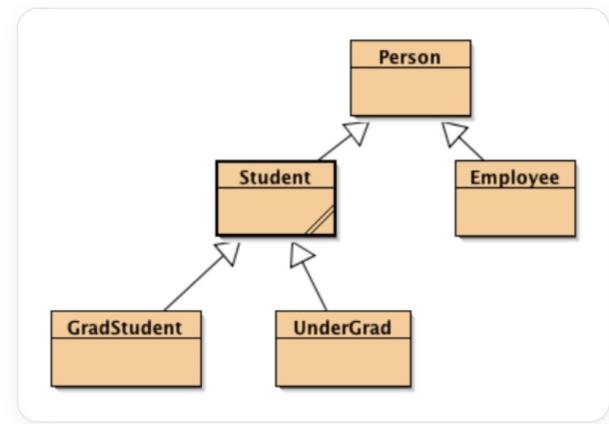
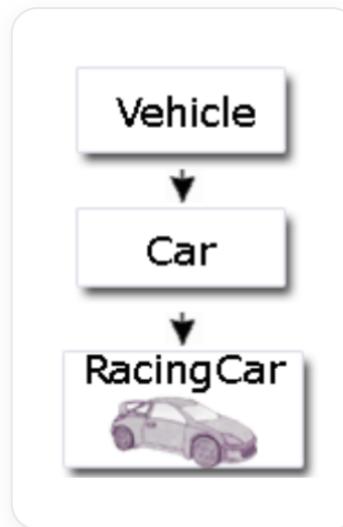
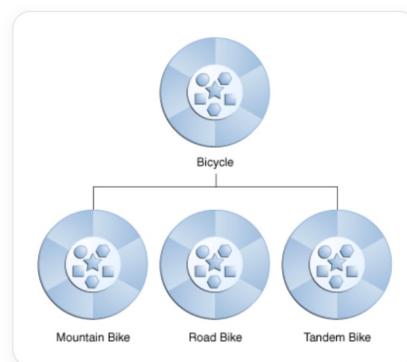
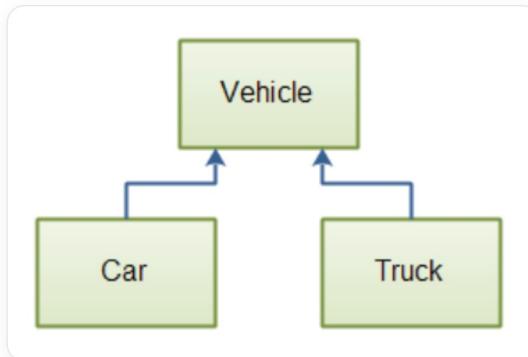
ارثبری چیست؟

فرض کنید میخواهید کلاسها یی تعریف کنید تا شکل هایی مثل دایره، مثلث و مستطیل را مدلسازی کند.

این کلاسها دارای صفات مشترک زیادی هستند مانند مساحت، محیط و...

به همین دلیل بهترین راه برای طراحی این کلاسها و پرهیز از تکرار نویسی، استفاده از **وراثت** است!

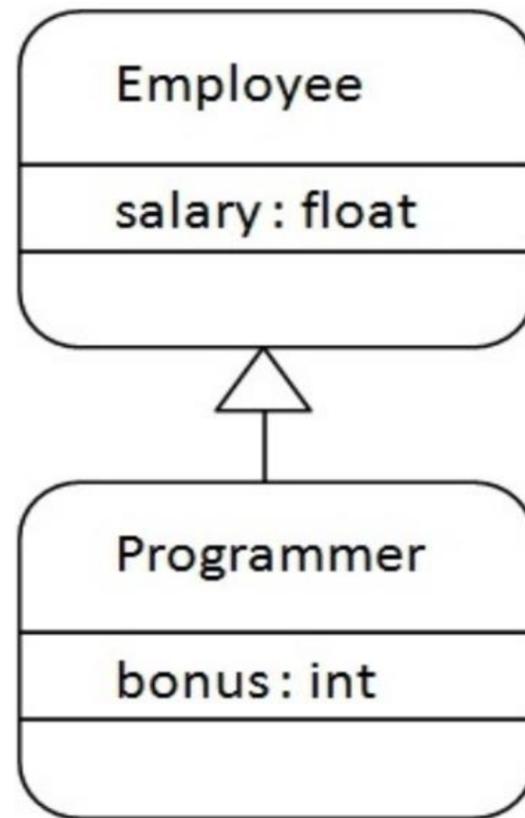
مثال هایی از وراثت در شکل های زیر مشاهده می کنید:



قاعده نحوی وراثت به صورت زیر است:

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

کلمه کلیدی **extends** میگوید که شما کلاس جدیدی ایجاد کرده اید که از یک کلاس موجود ارث بری می کند. به کلاس موجود، **ابرکلاس** یا کلاس والد (پدر) گفته میشود و به کلاس جدید نیز، **کلاس فرزند** میگویند. در تصاویر، پیکان همیشه از فرزند به پدر رسم میشود:



بر اساس شکل بالا، برنامه ای بنویسید که حقوق (salary) و پاداش (bonus) را که از کارمند ارث بری میکند، چاپ کند.

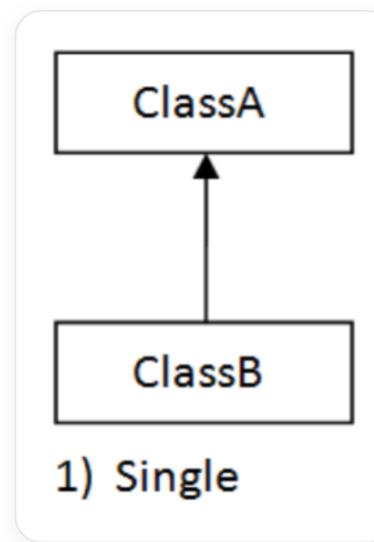
```
class Employee {  
    float salary = 40000;  
}  
  
class Programmer extends Employee {  
    int bonus = 10000;  
    public static void main(String args[]){  
        Programmer p = new Programmer();  
        System.out.println("Programmer salary is: " + p.salary);  
        System.out.println("Bonus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

Programmer salary is: 40000.0
Bonus of Programmer is: 10000

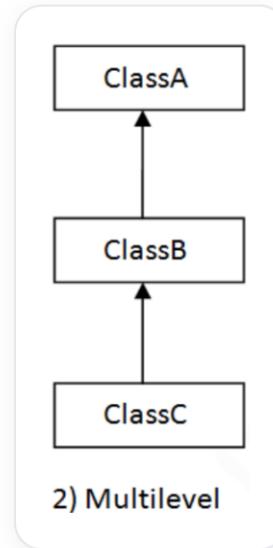
انواع ارث بری

سه نوع ارث بری بر اساس کلاس والد قابل تعریف است:

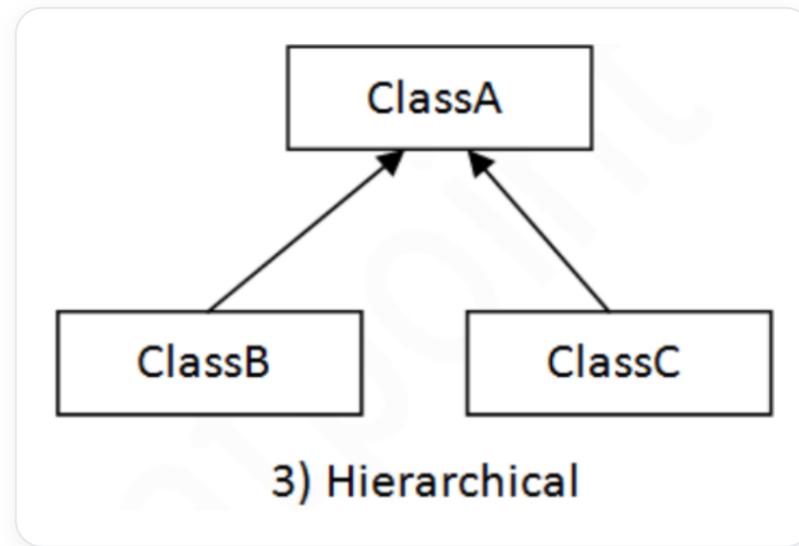
- ساده:



• چند سطحی:



- سلسله مراتبی:

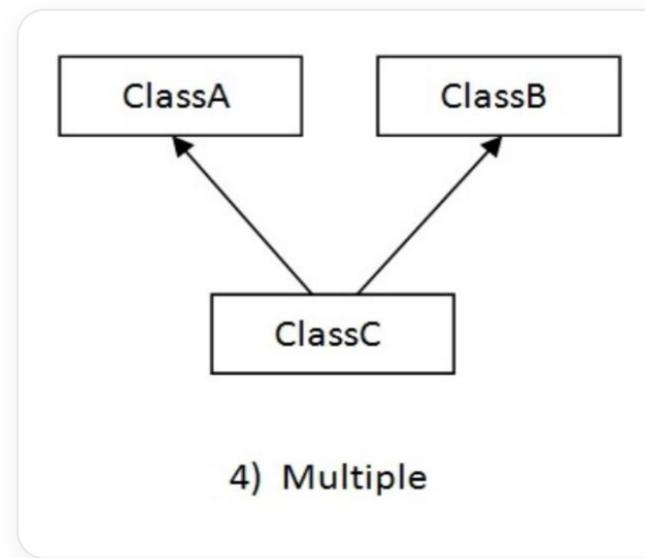


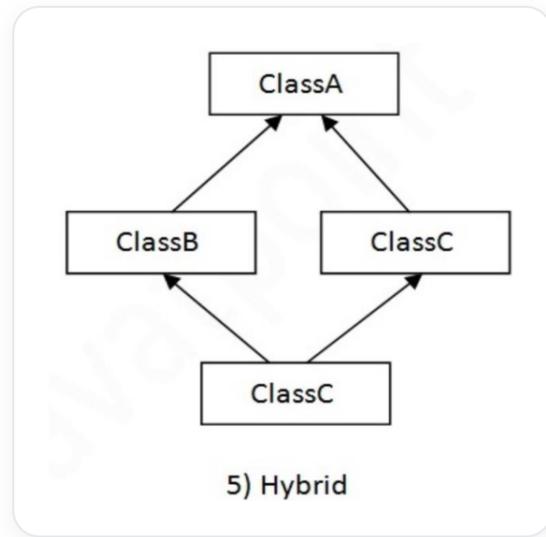
نکته: در جاوا ارث بری ترکیبی (هیبرید) و چندگانه تنها از طریق کلاس‌های واسط امکان پذیر است که بعده توضیح داده خواهد شد.

ارت برى چندگانه

ارت برى چندگانه چیست؟

هرگاه کلاسی بخواهد از چند کلاس ارت برى کند، به این نوع ارت برى، ارت برى چندگانه گفته می‌شود.





در جاوا ارث بری چندگانه پشتیبانی نمیشود. دلیل این امر، کاهش پیچیدگی و ساده سازی زبان است!

مثالی از ارث بری چندگانه:

در مثال زیر، دو کلاس والد متدهای نام msg دارند و کامپایلر نمی‌داند کدام را برای فرزند ارث بری کند. لذا در هر حالت (حتی عدم وجود متدهای هم نام در کلاسهای والد) جاوا برای ارث بری چندگانه، خطای زمان کامپایل می‌گیرد.

```
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 be invoked?
    }
}
```

تمرین زیرکلاس و ابرکلاس

1 class name:

GeometricObject

fields:

- color: String
 - The color of the object (default: white)
- filled: boolean
 - Indicates whether the object is filled with a color (default: false)
- dateCreated: java.util.Date
 - The date when the object was created

methods:

- + GeometricObject ()
 - Creates a GeometricObject.
- + GeometricObject (color: String, filled: boolean)
 - Creates a GeometricObject with the specified color and filled values.
- + getColor (): String
 - Returns the filled property.
- + setColor (color: String): void
 - Sets a new color.
- + isFilled (filled: boolean): void
 - Returns the filled property.
- + setFilled (filled: boolean): void
 - set a new filled property.
- + getDateCreated (): java.utin.Date
 - Returns the dateCreated.
- + toString (): String
 - Returns a string representation of this object

برنامه ای با مشخصات زیر بنویسید:

ابر کلاس:

2 **class name:**

Circle

fields:

- radius: double

methods:

+ Circle ()

+ Circle (radius: double)

+ Circle (radius: double, color: String, filled: boolean)

+ getRadius (): double

+ setRadius (radius: double): void

+ getArea (): double

+ getPerimeter (): double

+ getDiameter (): double

+ printCircle (): void

3 **class name:**

Rectangle

fields:

- width: double

- height: double

methods:

+ Rectangle ()

+ Rectangle (width: double, height: double)

+ Rectangle (width: double, height: double, color: String, filled: boolean)

+ getWidth (): double

+ setWidth (width: double): void

+ getHeight (): double

+ setHeight (height: double): void

+ getArea (): double

+ getPerimeter (): double

```
public class GeometricObject1 {  
    private String color = "white";  
    private boolean filled;  
    private java.util.Date dateCreated;  
    /** Construct a default geometric object */  
    public GeometricObject1() {  
        dateCreated = new java.util.Date();  
    }  
    /** Construct a geometric object with the specified color  
     * and filled value */  
    public GeometricObject1(String Color, boolean filled) {  
        dateCreated = new java.util.Date();  
        this.color = color;  
        this.filled = filled;  
    }  
    /** Return color */  
    public String getColor() {  
        return color;  
    }  
    /** Set a new color */  
    public void setColor(String color) {  
        this.color = color;  
    }  
    /** Return filled. Since filled is boolean,  
     * its get method is named isFilled */  
    public boolean isFilled() {  
        return filled;  
    }
```

```
    public void setFilled(boolean filled) {  
        this.filled = filled;  
    }  
    /** Get dateCreated */  
    public java.util.Date getDateCreated() {  
        return dateCreated;  
    }  
    /** Return a string representation of this object */  
    public String toString() {  
        return "created on " + dateCreated + "\ncolor: " +  
            color + " and filled: " + filled;  
    }  
}
```

```
public class Circle4 extends GeometricObject1 {
    private double radius;
    public Circle4() {
    }
    public Circle4(double radius) {
        super();
        this.radius = radius;
    }
    public Circle4(double radius, String color, boolean filled)
        super(color, filled);
        this.radius = radius;
        //setColor(color);
        //setFilled(filled);
    }
    /** Return radius */
    public double getRadius() {
        return radius;
    }
    /** Set a new radius */
    public void setRadius(double radius) {
        this.radius = radius;
    }
    /** Return area */
    public double getArea() {
        return radius * radius * Math.PI;
    }
}
```

```
/** Return diameter */
public double getDiameter() {
    return 2 * radius;
}
/** Return perimeter */
public double getPerimeter() {
    return 2 * radius * Math.PI;
}
/* Print the circle info */
public void printCircle() {
    System.out.println(toString() + "The circle is
        created " + getDateCreated() +
        " and the radius is " + radius);
}
public String toString() {
    return "Circle WWW " + getColor() +
        super.toString();
}
```

```
public class Rectangle1 extends GeometricObject1 {
    private double width;
    private double height;
    public Rectangle1() {
    }
    public Rectangle1(double width, double height) {
        this.width = width;
        this.height = height;
    }
    public Rectangle1(double width, double height, String color, boolean filled) {
        this.width = width;
        this.height = height;
        setColor(color);
        setFilled(filled);
    }
    /** Return width */
    public double getWidth() {
        return width;
    }
    /** Set a new width */
    public void setWidth(double width) {
        this.width = width;
    }
    /** Return height */
    public double getHeight() {
        return height;
    }
    /** Set a new height */
    public void setHeight(double height) {
        this.height = height;
    }
    /** Return area */
    public double getArea() {
        return width * height;
    }
    /** Return perimeter */
    public double getPerimeter() {
        return 2 * (width + height);
    }
}
```

```
public class TestCircleRectangle {  
    public static void main(String[] args) {  
        Circle4 circle = new Circle4(1);  
        System.out.println("A circle " + circle.toString());  
        System.out.println("The radius is " + circle.getRadius());  
        System.out.println("The area is " + circle.getArea());  
        System.out.println("The diameter is " + circle.getDiameter());  
        Rectangle1 rectangle = new Rectangle1(2, 4);  
        System.out.println("\nA rectangle " + rectangle.toString());  
        System.out.println("The area is " + rectangle.getArea());  
        System.out.println("The perimeter is " + rectangle.getPerimeter());  
    }  
}
```

```
A circle Circle WWW white created on Sat Nov 15 10:05:44 IRST 2025  
color: white and filled: false  
The radius is 1.0  
The area is 3.141592653589793  
The diameter is 2.0  
  
A rectangle created on Sat Nov 15 10:05:44 IRST 2025  
color: white and filled: false  
The area is 8.0  
The perimeter is 12.0
```

ارث بری از ابرکلاسها

آیا سانده ابرکلاسها، ارث بری میشوند؟

خیر به ارث برده نمی شوند! قبل تر دیدیم که سازنده ها به صورت ضمنی یا صریح در بدن کلاس قرار داده می شوند.

یک سازنده برای ایجاد یک نمونه (یک شیء جدید) از یک کلاس استفاده میشود. برخلاف فیلد های داده ای و متدها، با استفاده از کلمه کلیدی **super** انجام می شود. اگر کلمه کلیدی **super** استفاده نشود سازنده no-arg کلاس پدر به طور خودکار فراخوانی می شود.

درواقع یک سازنده ممکن است یک سازنده سربارگذاری شده یا سازنده ابرکلاسش را فراخوانی کند. اگر هیچ یک از این دو، مشخصاً فراخوانی نشوند، کامپایلر به طور خودکار `super()` را به عنوان اولین دستور در سازنده قرار می دهد. به عنوان مثال کد زیر:

```
public A(){}
```

به صورت زیر کامپایل میشود:

```
public A(){ super();}
```

همچنین کد زیر:

```
public A(double d){ // some statements }
```

به صورت زیر کامپایل می شود:

```
public A(){ super(); // some statements}
```

کلمه کلیدی **:super**

کلمه کلیدی **super** در کلاس فرزند استفاده می شود و به والد آن کلاس اشاره می کند. اگر از کلمه کلیدی **super** برای ارجاع به متغیر نمونه کلاس والد، وقتی هر دو یک فیلد هم نام دارند، استفاده نشود:

```
class Vehicle{  
    int speed = 50;  
  
    void display(){  
        System.out.println(speed); // will print speed of Bike  
    }  
}  
  
class Bike3 extends Vehicle{  
    int speed = 100;  
}  
public static void main(String args[]){  
    Bike3 b = new Bike3();  
    b.display();  
}
```

50

```
class Vehicle{
    Vehicle(){
        System.out.println("Vehicle is created!");
    }
}

class Bike5 extends Vehicle{
    Bike5(){
        super(); //will invoke parent class constructor
        System.out.println("Bike is created");
    }
}

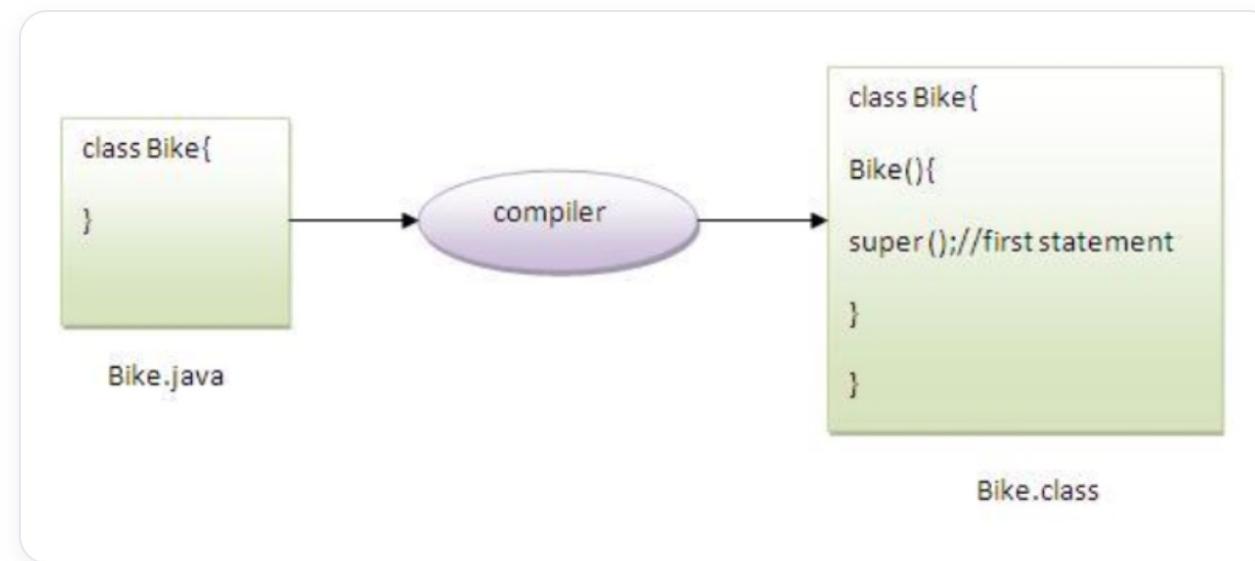
public static void main(String args[]){
    Bike5 b = new Bike5();
}
```

Vehicle is created!
Bike is created

کلمه کلیدی super برای فراخوانی متدهای والد

همانطور که میدانیم، اگر خود ما سازنده ای قرار ندهیم، سازنده پیش فرض توسط کامپایلر ایجاد می شود.

در کلاس فرزند، کامپایلر به طور خودکار کلمه `super()` را به عنوان اولین دستور به سازنده فرزند اضافه می کند.



در موقعی که متد فرزند با متد والد همنام باشد، از کمله کلیدی **super** برای فراخوانی متد کلاس والد استفاده می شود:

```
class Person{
    void message(){
        System.out.println("Welcome");
    }
}

class Student16 extends Person{
    void message(){
        System.out.println("Welcome to java");
    }
    void display(){
        message(); //will invoke current class message() method
        super.message(); //will invoke parent class message() method
    }
}

class main{
    public static void main(String args[]){
        Student16 s = new Student16();
        s.display();
    }
}

main.main(new String[5])
```

Welcome to java
Welcome

زنجیره سازنده ها

ایجاد یک شیء از یک کلاس تمامی سازنده های آن و ابرکلاسهای آن را در قالب زنجیره ای وراثتی فراخوانی می کند که به آن زنجیره سازنده ها گفته می شود.

```
public class Faculty extends Employee {
    public static void main(String[] args) {
        new Faculty();
    }
    public Faculty() {
        System.out.println("(4) Faculty's no-arg constructor is invoked");
    }
}

class Employee extends Person {
    public Employee() {
        this("(2) Invoke Employee's overloaded constructor");
        System.out.println("(3) Employee's no-arg constructor is invoked");
    }
    public Employee(String s) {
        System.out.println(s);
    }
}

class Person {
    public Person() {
        System.out.println("(1) Person's no-arg constructor is invoked");
    }
}
```

به عنوان مثال کد زیر را در نظر بگیرید:

ردگیری اجرای کد

در هنگام اجرای کد، به ترتیب زیر اجرا می شود:

1_ Start from the main method:

```
public static void main(String args[])
```

در ابتدا خط دوم، تابع **main** اجرا میشود.

2_ Invoke Faculty constructor:

```
new Faculty();
```

سپس خط سوم، یعنی متدهای **Faculty** اجرا میشود که ارجاع میشود به **public Faculty()**.

3_ Invoke Employee's no-arg constructor:

```
public Employee();
```

در ادامه چون **employee** از **faculty** ارث بری میکند، سازنده **employee** اجرا میشود یعنی خط یازدهم.

4_ Invoke Employee(String) constructor;

```
this("(2) Invoke Employee's overloaded constructor");  
public Employee (String s);
```

خود سازنده بدون آرگومان، سازنده **Employee** با ورودی رشته را اجرا میکند.

5_ Invoke Person() constructor:

```
public Person();
```

چون **Employee** از کلاس **Person** ارث بری میکند، در ادامه سازنده بدون آرگومان (**Person()**) را صدا میزند.

در ادامه متدهای **println** به ترتیب زیر اجرا میشوند:

```
6_ System.out.println ("(1) Person's no-arg constructor is invoked");
7_ System.out.println (s);
8_ System.out.println ("(3) Employee's no-arg constructor is invoked")
9_ System.out.println ("(4) Faculty's no-arg constructor is invoked");
```

بعد اجرای متدهای پرینت، برنامه پایان میابد.

نکته: اگر در یک ابرکلاس، سازنده ای غیر no-arg تعریف کنیم، باید در زیرکلاس ها سازنده **super** را با آرگومان صدا کنیم.

خطای برنامه زیر را بیابید:

```
public class Apple extends Fruit {  
}  
class Fruit {  
    public Fruit(String name) {  
        System.out.println("Fruit's constructor is invoked");  
    }  
}
```

خودمون رو بسنجیم

این بخش برای این طراحی شده که در پایان مطالعه این اسلاید، بتونی خودت رو محک بزنی و ببینی آیا مفاهیم رو به خوبی یاد گرفتی یا نه. سوالات زیر رو مرور کن و سعی کن بدون نگاه کردن به متن درس، به اون ها پاسخ بدی.

- انتزاع یا **abstraction** چیست و چه تفاوتی با کپسوله بندی **encapsulation** دارد؟
- برای وراثت یک مثال بزنید و انواع آن را نام ببرید.
- کلمه کلیدی **super** چه کاربرد و عملکردی دارد؟

پایان

در صورت هرگونه سوال یا پیشنهاد میتوانید با من
در ارتباط باشید:

gmail: Parsahamzeie@gmail.com
telegram: @ParsaHami