برنامه نویسی پیشرفته

وحیدی اصل

مباحث پیشرفته شیئ گرایی

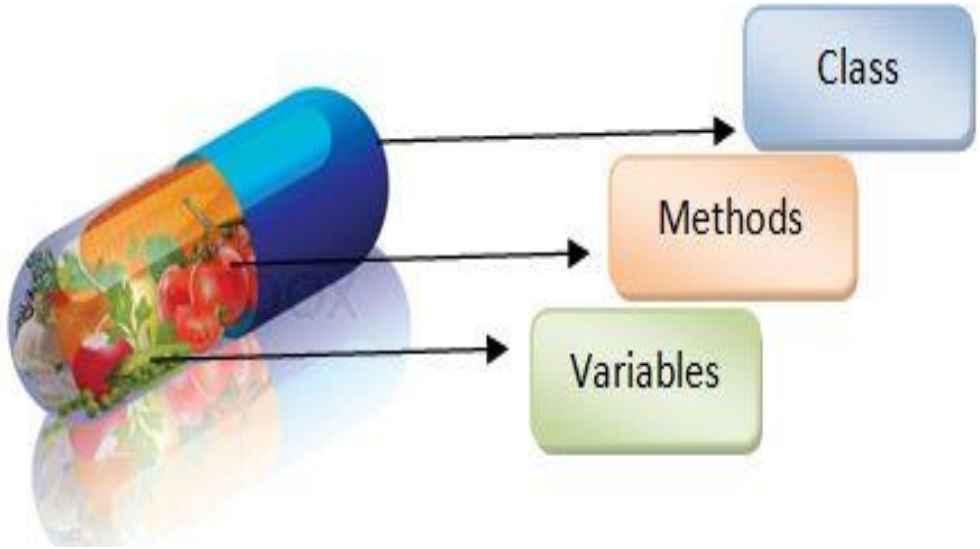
مزایای شیئ گرایی – انتزاع (Abstraction)

- **پنهان کردن جزئیات داخلی از چشمان کاربر، انتزاع گفته می شود. کاربر، فقط باید کارکرد (قابلیتها) یا خروجی وسیله را منطبق بر نیازهای خویش بداند.**

- در جاوا، از کلاس abstract و interface برای رسیدن به انتزاع استفاده می کنیم.

مزایای شیئ گرایی – کپسوله بندی (encapsualtion)

- **ترکیب و بسته بندی داده ها (فیلدهای داده ای) و متدها در یک واحد به نام شیئ اصطلاحاً کپسوله بندی گفته می شود.**



- یک کلاس در جاوا مثالی از کپسوله بندی است.

تفاوت کپسوله بندی و انتزاع

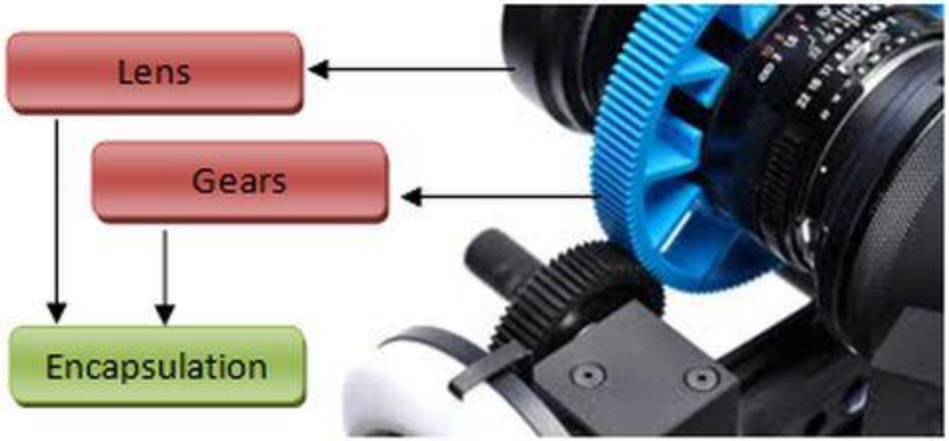
- یک دوربین عکاسی دیجیتالی زیر را در نظر بگیرید:

- برروی راهنمای این دوربین نوشته شده:

"کاربر محترم، در هنگام استفاده از دوربین دیجیتالی کافیست برروی 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 بازکنید با مکانیزم پیچیده آن روبرو خواهید که برای شما قابل فهم نخواهد بود. فشردن دکمه عکاسی و گرفتن عکس (نتایج دلخواه) انتزاع نامیده می شود.

تفاوت کپسوله بندی و انتزاع



- وقتی ما برروی دکمه zoom in/out کلیک می کنیم، در درون دوربین مکانیزمهای شامل چرخ دنده ها و لنزها خواسته ما را برآورده می کنند.
- ترکیب این چرخ دنده ها و لنزها اصطلاحاً کپسوله بندی گفته می شود که به عکاس امکان می دهد زومینگ را به نرمی و سادگی انجام دهد.
- به زبان ساده می گوییم: انتزاع از طریق کپسوله بندی امکان پذیر شده است.
- یا: انتزاع جنبه **طراحی** مسئله را انجام می دهد و کپسوله بندی مسئول **پیاده سازی** مسئله است!

- مثال کیف پول را در نظر بگیرید:

مزایای شیئ گرایی-پلی مورفیسم

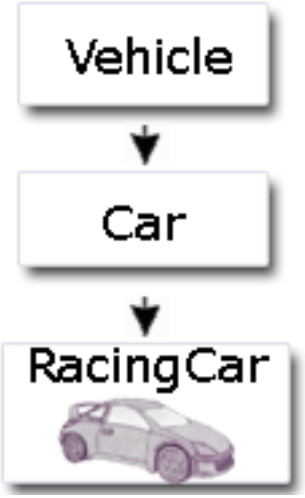
- پلی مورفیسم یعنی اینکه کاری به شکلها یا شیوه های مختلفی انجام شود.
- اینکه بگوییم " صحبت کردن" بین همه حیوانات مشترک است، اما هر حیوانی گویش و صدای خاص خود را دارد.
- اینکه بگوییم، شکلی را کشیده ایم. اما یک بار دایره بکشیم، یکبار مستطیل و به همین ترتیب.
- در جاوا از بازنویسی (overriding) متد برای رسیدن به پلی مورفیسم استفاده می کنیم.

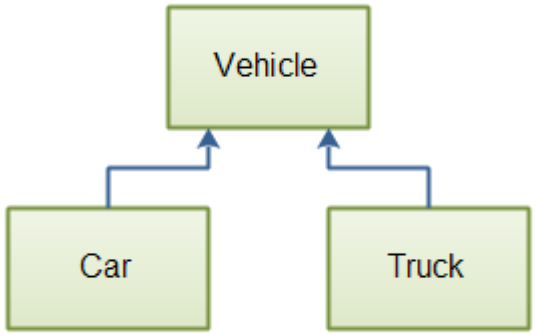
ارثبری- انگیزه

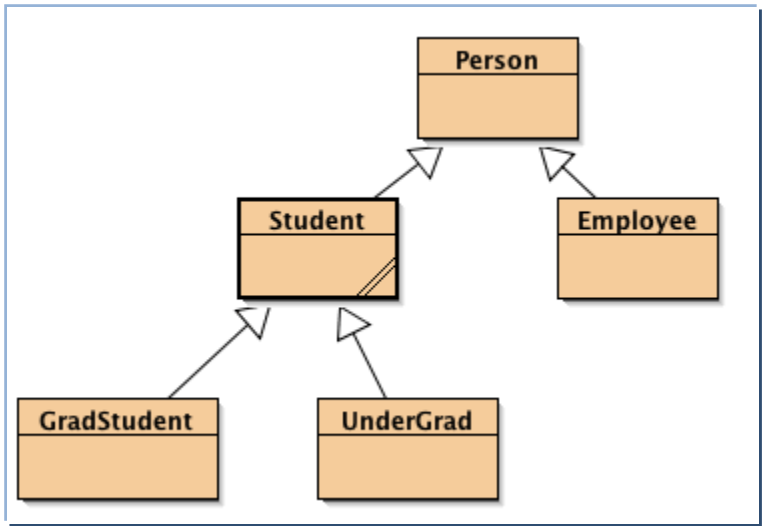
فرض کنید می خواهید کلاسهایی تعریف کنید تا دایره، مستطیل و مثلث را مدلسازی کنید. این کلاسها دارای صفات مشترک زیادی هستند.

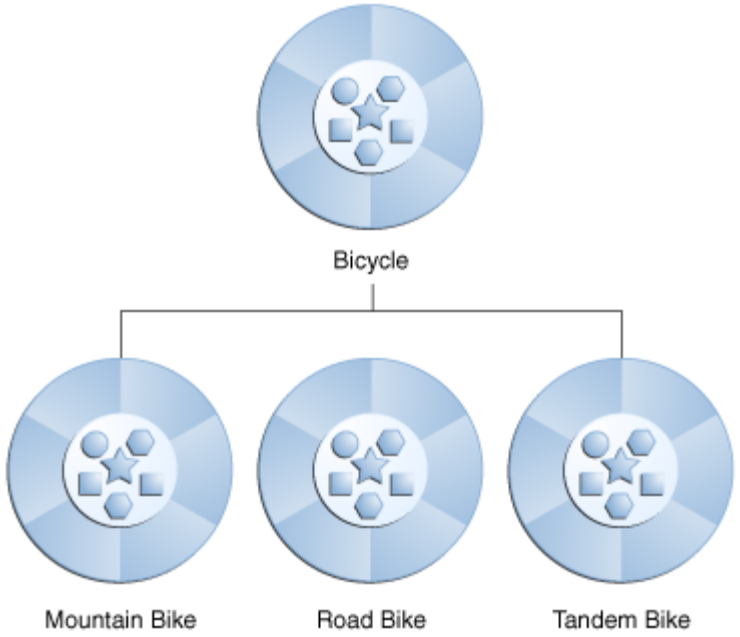
بهترین راه برای طراحی این کلاسها برای پرهیز از تکرارنویسی چیست؟ پاسخ، استفاده از وراثت است.

مثالهای از ارث بری (وراثت)





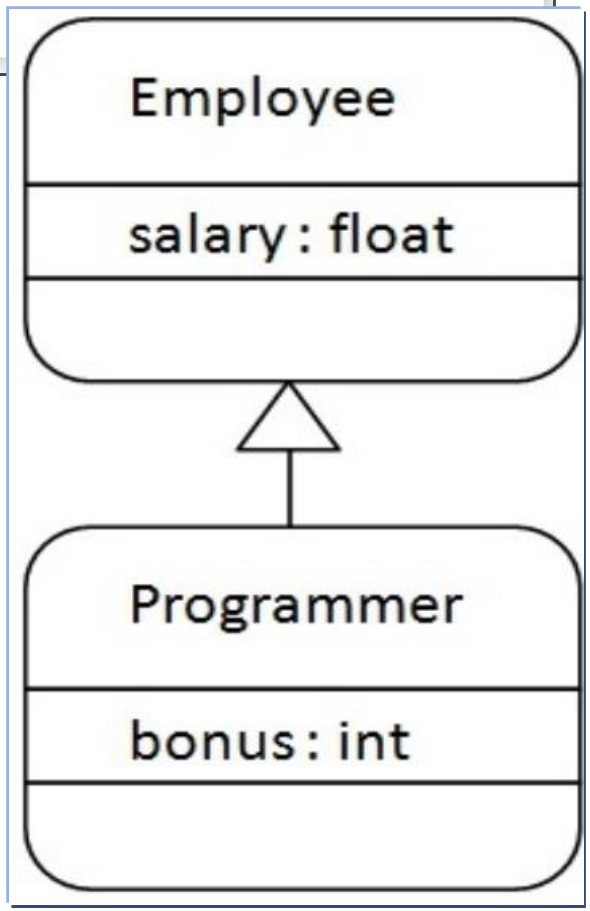




ساختار وراثت

- قاعده نحوی وراثت:

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



- کلمه کلیدی extends می گوید شما کلاس جدیدی ایجاد کرده اید که از یک کلاس موجود ارث بری می کند.
- به کلاس موجود، ابرکلاس یا والد (پدر) گفته می شود.
- به کلاس جدید؛ زیرکلاس یا فرزند می گویند.
- پیکان همیشه از فرزند به پدر رسم می شود.
- براساس شکل سمت راست برنامه ای بنویسید که حقوق (salary) و پاداش (bonus) یک برنامه نویس را که از کارمند ارث بری می کند، چاپ کند.

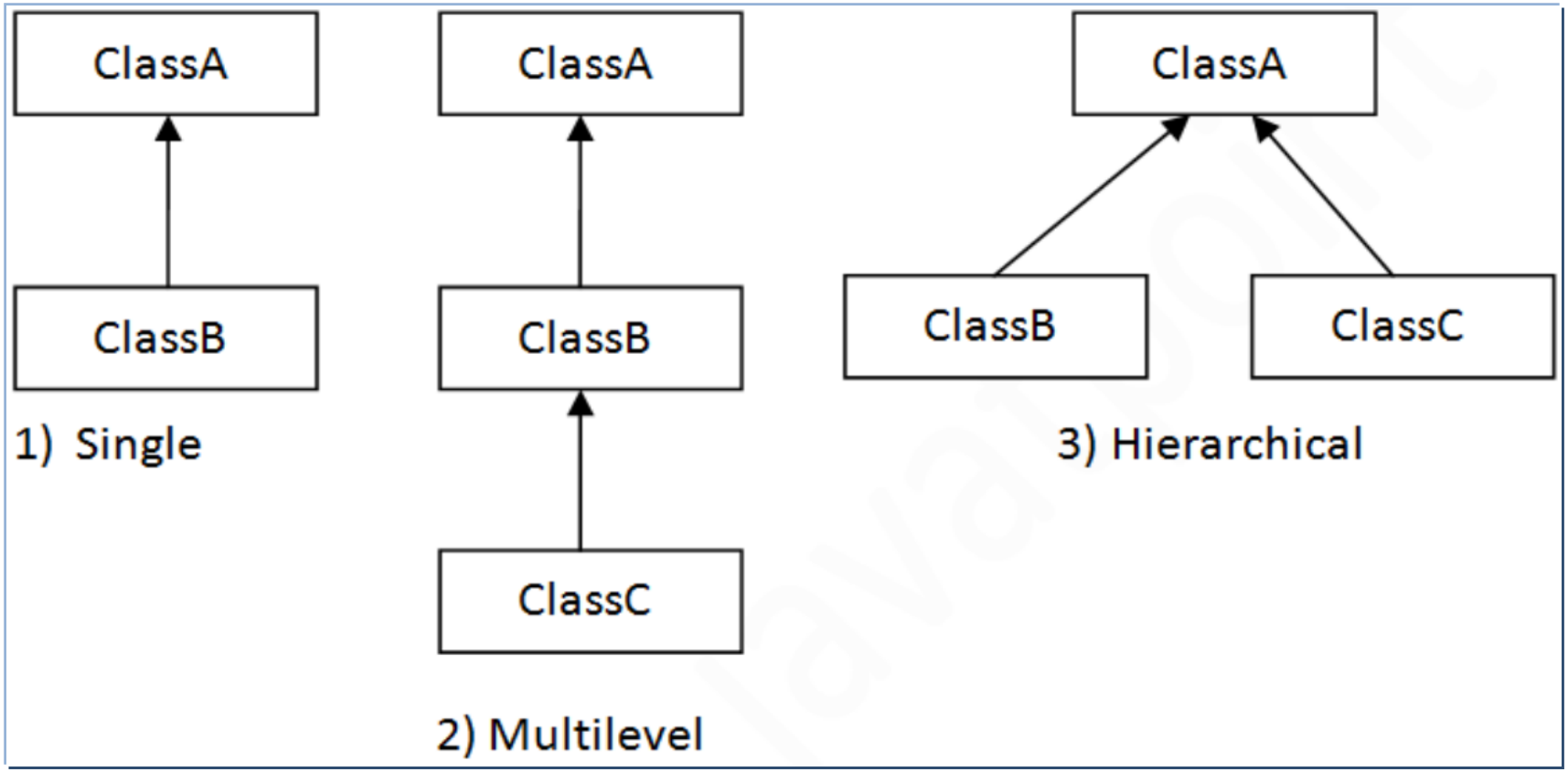
تمرین ارث بری

```
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);
}
}
```

Test it Now

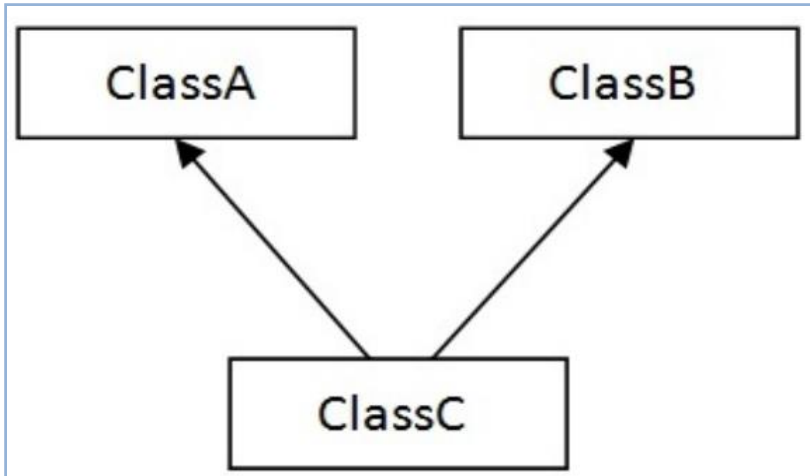
```
Programmer salary is:40000.0
Bonus of programmer is:10000
```

- سه نوع ارث بری براساس کلاس والد قابل تعریف است: ساده، چند سطحی و سلسله مراتبی
- در جاوا، ارث بری ترکیبی (هیبرید) و چندگانه تنها از طریق کلاسهای واسط امکان پذیر است که بعدا توضیح داده خواهد شد.

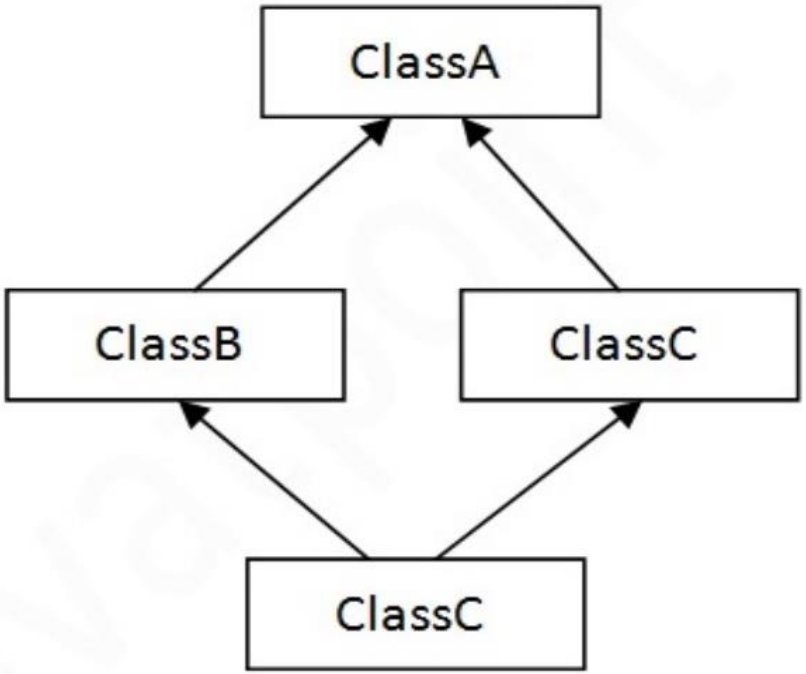


ارث بری چندگانه

- هرگاه کلاسی بخواهد از چند کلاس ارث بری کند، به نوع ارث بری، چندگانه گفته می شود.
- در جاوا ارث بری چندگانه پشتیبانی نمی شود. دلیل این امر، کاهش پیچیدگی و ساده سازی زبان است!



4) Multiple



5) Hybrid

ارث بری چندگانه

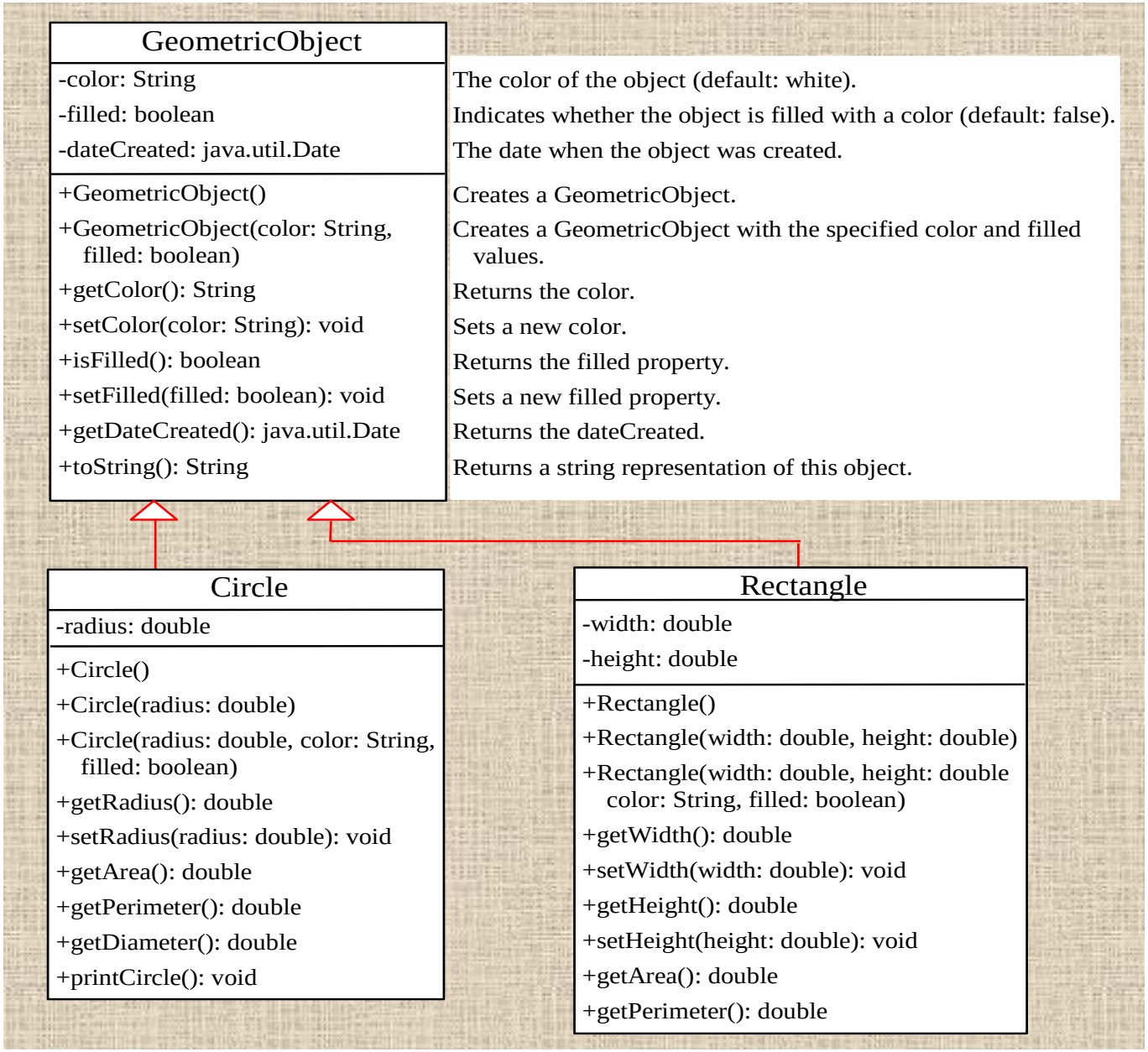
- در مثال زیر دو کلاس والد متدی به نام msg دارند و کامپایلر نمی داند کدام را برای فرزند ارث بری کند.
- لذا در هر حالت (حتی عدم وجود متدهای هم نام در کلاسهای والد) جاوا برای ارث بری چندگانه، خطای زمان کامپایل می گیرد.

```
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 would be invoked?
}
}
```

Test it Now

```
Compile Time Error
```

زیرکلاسها و ابرکلاسها



زیرکلاسها و ابرکلاسها

```
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;
  }

  /** Set a new 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;
  }
}
```

زیرکلاسها و ابرکلاسها

```
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 WWWW " + getColor() +
super.toString();
  }
}
```

زیرکلاسها و ابرکلاسها

```
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);
  }
}
```

زیرکلاسها و ابرکلاسها

```
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());
  }
}
```

آیا سازنده ابرکلاسها، ارث بری می شوند؟

- خیر، به ارث برده نمی شوند!
- قبلا گفتیم که سازنده ها به صورت ضمنی یا صریح در بدنه کلاس قرار داده می شوند.
- یک سازنده برای ایجاد یک نمونه (یک شیئ جدید) از یک کلاس استفاده می شود. برخلاف فیلدهای داده ای و متدها، سازندهای یک ابرکلاس به زیرکلاسهایش ارث بری نمی شوند.
- فراخوانی سازنده های پدر از طرف سازنده های فرزندان، با استفاده از کلمه کلیدی super انجام می شود.
- اگر کلمه super استفاده نشود، سازنده no-arg کلاس پدر به طور خودکار فراخوانی می شود.

سازنده یک ابرکلاس همیشه فراخوانی می شود

- 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.
- برای مثال:

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

is equivalent to

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

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

is equivalent to

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

استفاده از کلمه کلیدی super

- کلمه کلیدی super در کلاس فرزند استفاده می شود و به والد آن کلاس اشاره می کند.
- کاربردهای این کلمه در اسلایدهای بعدی نشان داده شده است.

عدم استفاده از کلمه کلیدی super

اگر از کلمه کلیدی super برای ارجاع به متغیرنمونه کلاس والد، وقتی هر دو یک فیلد هم نام دارند، استفاده نشود:

```
class Vehicle{
  int speed=50;
}

class Bike3 extends Vehicle{
  int speed=100;

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

}
}
```

Test it Now

کلمه کلیدی super برای فراخوانی سازنده کلاس والد استفاده می شود.

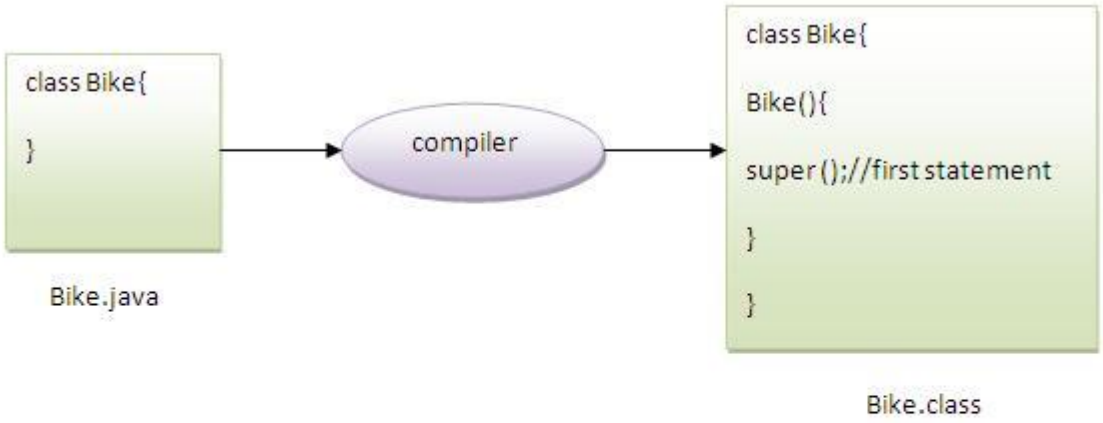
```
class Vehicle{
  Vehicle(){System.out.println("Vehicle is created");}
}

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();

}
}
```

Test it Now

```
Output:Vehicle is created
       Bike is created
```

استفاده از کلمه کلیدی super در فراخوانی سازنده

- همانطور که می دانیم سازنده پیش فرض توسط کامپایلر ایجاد می شود (اگر ما خود سازنده ای ننوشته باشیم).
- در کلاس فرزند، کامپایلر به طور خودکار کلمه super() را به عنوان اولین دستور به سازنده فرزند اضافه می کند.



استفاده از کلمه کلیدی super برای فراخوانی متد کلاس والد

- در مواقعی استفاده می شود که متد فرزند با متد والد همنام باشد.

```
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
}

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

Test it Now

```
Output:welcome to java
       welcome
```

زنجیره سازنده ها

ایجاد یک شیئ 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");
  }
}
```

ردگیری اجرا

```
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");
  }
}
```

1. Start from the main method

ردگیری اجرا

```
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");
  }
}
```

2. Invoke Faculty constructor

ردگیری اجرا

```
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");
  }
}
```

3. Invoke Employee's no-arg constructor

ردگیری اجرا

```
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");
  }
}
```

4. Invoke Employee(String) constructor

ردگیری اجرا

```
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");
  }
}
```

5. Invoke Person() constructor

ردگیری اجرا

```
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");
  }
}
```

6. Execute println

ردگیری اجرا

```
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");
  }
}
```

7. Execute println

ردگیری اجرا

```
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");
  }
}
```

8. Execute println

ردگیری اجرا

```
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");
  }
}
```

9. Execute println

اگر یک ابرکلاس را با سازنده ای غیر no—argتعریف کنیم

خطاهای این برنامه رابیابید:

```
public class Apple extends Fruit {
}

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