آموزش زبان برنامه نویسی جاوا

ارث بری

جلسه بیستم

نویسنده : رحمان زارعی

جاوا را ساده،آسان و شیرین بنوشید!!!!

در این جلسه بخش اول ارث بری را بررسی میکنیم:

ما از جلسه ۱ تا ۱۹ ابتدا با ابزارها و برنامه های مورد نیاز برای نوشتن یک برنامه به زبان جاوا و مباحث پایه ای جاوا آشنا شدیم،می توان گفت این مباحث پایه ای نظیر تعریف متغیرها،حلقه ها،شرط ها،متدهاو... در بین همه زبان های برنامه نویسی مشترک هستند،مهم ترین بخش که جاوا را با سایر زبان های برنامه نویسی متمایز میکند همین مبحث شی گرایی جاوا می باشد!!! که ارث بری یا وراثت جزیی از این بخش هستش. تمام سعی خودم رو میکنم این جلسه مهم را به زبان ساده بیان کنم!خب بریم سراغ اصل مطلب!

ارث بری یا وراثت!

ارث بری در برنامه نویسی جاوا یعنی چه؟!!! ارث بری یعنی یک کلاس از خصوصیات (متدها و متغیرهای) کلاس دیگر استفاده کند!

ارث بری در جاوا مثل دنیای واقعی هست!!!! بزارید با مثالی از دنیای واقعی این مبحث را جا بندازم:

یک فرزند از پدر خود می تواند مال و منالی رو چه در زمان حیات و چه بعد از مرگ به ارث ببرد!!!

مثلا در زمان حیات پدر جعفر یک خانه + یک پراید دارد،جعفر چون فرزند پدرش هست میتونه در خانه پدر زندگی کنه و از امکانات منزل پدر استفاده کند و گهگاهی هم دزدکی پدر پراید رو از خونه خارج کنه و جیم بشه ☺ تنها جعفر به دلیل این که

فرزند پدرش هست حق داره در خانه زندگی کند و حالا بره با پراید ی دور دوری هم کنه ☺ مثلا گرگعلی پسر همسایه نمی تونه بیاد در خانه پدر جعفر زندگی کنه چون پدر جعفر بابای گرگعلی نیست!!! نتیجه میگیریم با یک رابطه پدر و فرزندی،فرزند میتونه از امکانات پدر استفاده ببرد در جاوا هم همین طور هست!!!! یعنی یک کلاس میتونه در صورتی از ویژگی ها و رفتار های کلاس دیگر استفاده کند که اون کلاس را به ارث ببرد ،کلاسی که ویژگی ها و رفتارهای کلاس دیگر را به ارث می برد کلاس فرزند یا زیر کلاس و کلاسی که کلاس فرزند از خصوصیات و رفتارهای آن استفاده میکند کلاس پدر می گوییم.

مثال: فرض کنید دو کلاس به نام A و B داریم،در صورتی کلاس A پدر کلاس B و B فرزند کلاس A هستش که کلاس B کلاس A را به ارث ببرد!!!!خب چطور یک کلاس می تواند کلاس دیگر را به ارث ببرد؟!!! پاسخ در ادامه آموزش!

کلمه کلیدی extends :

با استفاده از کلمه کلیدی extends یک کلاس می تواند ویژگی ها و رفتار های کلاس دیگر را به ارث ببرد.یعنی یک کلاس با استفاده از این کلمه کلیدی می تواند خودش را فرزند کلاس دیگر جا بزند! و اون کلاس نقش پدرش را بازی کند.

نحوه به ارث بردن دو کلاس را در زیر مشاهده میکنیم:

```
package javalike;

public class Child extends parent {


}

class Parent {

}
```

- در این کد برنامه، کلاس Child با استفاده از کلمه کلیدی extends کلاس Parent را به ارث برده است که با این کار کلاس Child فرزند کلاس Parent و کلاس Parent پدر کلاس Child شده است،به گونه ای که تمامی خصوصیات،ویژگیها و رفتار های کلاس Parent برای کلاس Child قابل دستیابی هستش.چیزی شبیه رابطه جعفر با پدرش ☺
- در اصطلاحات ارث بری یا وراثت به کلاس پدر ،کلاس super،کلاس بزرگ،کلاس اصلی،کلاس پایه،پدربزرگ یا هر چیزی که نشان از بزرگی داره میگن ☺ و به کلاس فرزند،کلاس sub ،زیر کلاس میگن و هر چیزی که جزیی از کل را نشان دهد.
- در زیر نمونه کد دیگر که تنها اسمشون فرق میکنه مشاهده میکنید:

```
class Super {

}
class Sub extends Super {

}
```

- کلاس Sub کلاس Super را به ارث برده است.یعنی کلاس Sub زیر کلاس یا فرزند کلاس Super هستش و Super پدر کلاس Sub می باشد.
- قبل از کلمه کلیدی extends، نام کلاس فرزند (کلاسی که قصد داره از متغیرها و متدهای کلاس دیگر استفاده کند) و بعد از آن نام کلاس پدر (کلاسی که قراره یک کلاس از ویژگی ها و رفتارش استفاده ببرد) قرار میگیرد.یعنی این کلمه تنها در کلاس فرزند استفاده می شود برای به ارث بردن کلاس پدر.

```
class A {

}
class B extends A {

}
```

- کلاس B فرزند کلاس A و کلاس A پدر کلاس B هستش، زیرا کلاس B کلاس A را به ارث برده است.
- در زیر یک مثال ساده کاربردی ارث بری یک کلاس از کلاس دیگر را بررسی میکنیم:

```
package javalike;
class Calculation {
   int z;

   public void addition(int x, int y) {
      z = x + y;
      System.out.println("The sum of the given numbers:"+z);
   }

   public void Subtraction(int x, int y) {
      z = x - y;
      System.out.println("The difference between the given numbers:"+z);
   }
}
```

```
public class My_Calculation extends Calculation {
   public void multiplication(int x, int y) {
      z = x * y;
      System.out.println("The product of the given numbers:"+z);
   }

   public static void main(String args[]) {
      int a = 20, b = 10;
      My_Calculation demo = new My_Calculation();
      demo.addition(a, b);
      demo.Subtraction(a, b);
      demo.multiplication(a, b);
   }
}
```

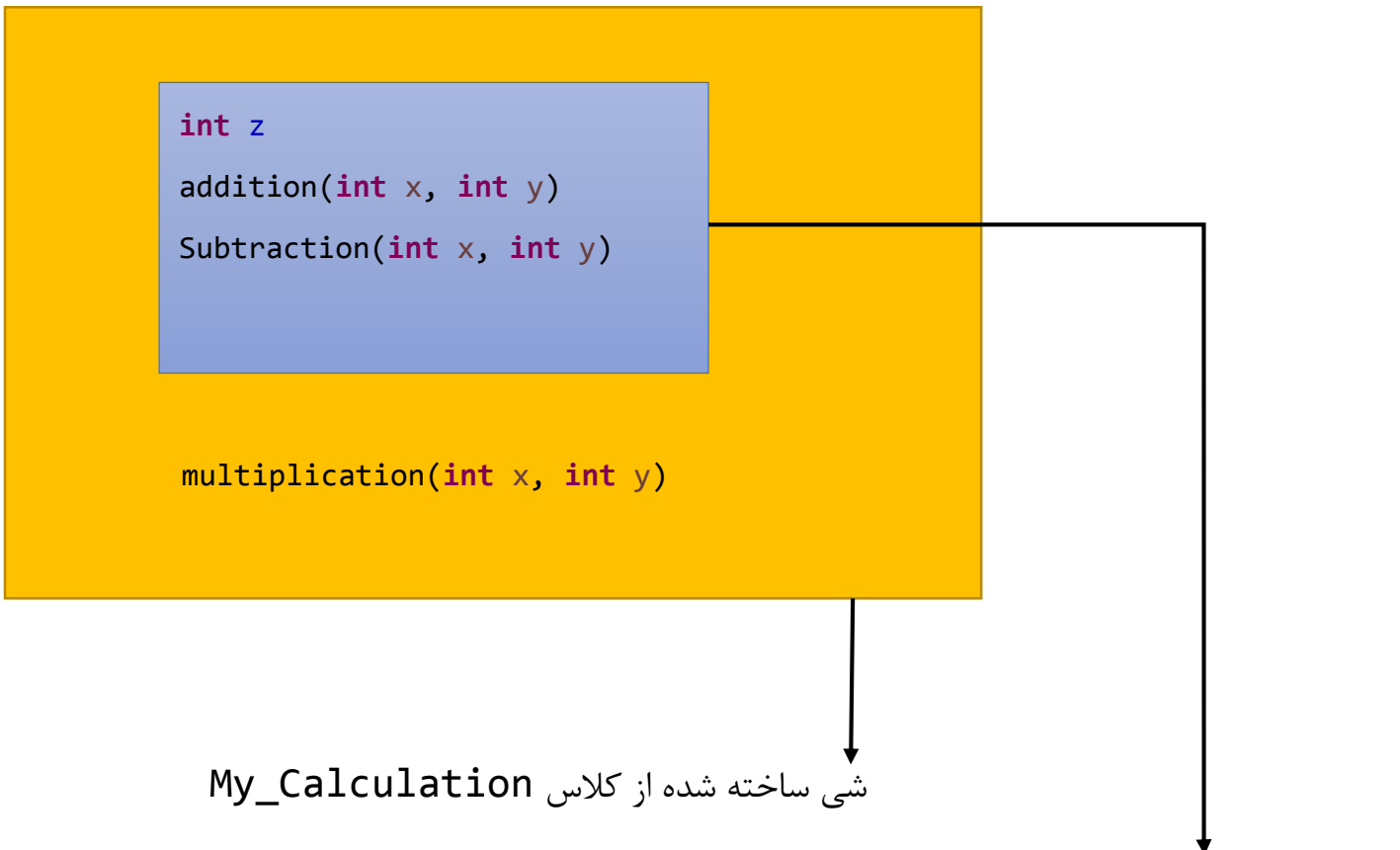
خروجی:

```
The sum of the given numbers:30
The difference between the given numbers:10
The product of the given numbers:200
```

- در این مثال دو کلاس به نام های Calculation وMy_Calculation داریم،که کلاس My_Calculation کلاس Calculation را به ارث برده است،و طبق قانون رسما کلاس My_Calculation فرزند کلاس Calculation می شود و تمامی متغیرها و متدهای کلاس پدر یعنی Calculation برای کلاس فرزند یعنی My_Calculation قابل دستیابی است.
- کلاس Calculation دارای یک متغیر از نوع عدد صحیح به نام z و دو متد addition و Subtraction هستند که هر دو متد دو پارامتر به عنوان ورودی میگیرن و اولی حاصل جمع و دومی حاصل تفریق دو عدد را در خروجی چاپ میکنند.
- کلاس My_Calculation دارای یک متد به نام multiplication می باشد که دو عدد صحیح از پارامتر ورودی خود میگیرد و حاصلضرب دو عدد را در خروجی چاپ میکند، و همچنین دارای متد main برای اجرای برنامه هستش،در متد main این کلاس دو متغیر محلی از نوع عدد صحیح به نام های a,b تعریف و مستقیم مقداردهی کرده ایم.بعد یک شی از کلاس My_Calculation به نام demo ایجاد کرده ایم.حال با شی demo متدهای این کلاس یعنی addition, Subtraction, multiplication را صدا زده ایم!!! اما یک نکته مهم!!!در بدنه کلاس ما متدهای addition, Subtraction پیاده سازی که نشده؟!!!چرا برنامه خطا نمیده؟!! پاسخش اینه که چون کلاس My_Calculation کلاس Calculation را به ارث برده است یعنی کلاس My_Calculation فرزند کلاس Calculation می باشد و طبق قانون ارث بری وقتی یک کلاس کلاس دیگر را به ارث می برد می تواند

از تمامی متدها و ویژگی های اون کلاس دسرسی داشته باشد بدون این که نیاز به پیاده سازی آنها در کلاس خود داشته باشد یعنی این متدها از پدر و استفاده از فرزندامانند این است که پراید مال پدر جعفر بود اما جعفر هم ازش استفاده میکرد!!

- در این برنامه وقتی ما یک شی از کلاس My_Calculation میسازیم،علاوه بر این که این شی حاوی ویژگی ها و رفتار های کلاس My_Calculation هستش،یک کپی از محتویات کلاس پدر خود Calculation را نیز در بر دارد،به همین خاطر از طریق شی ساخته شده از کلاس فرزند یعنی My_Calculation می توان به ویژگی ها و رفتار های کلاس پدر یعنی Calculation دسرسی داشت.این مفهوم بصورت شکل در زیر آمده است:



کلمه کلیدی super :

کلمه کلیدی super شبیه کلمه کلیدی this هستش.در زیر جاهایی که کلمه کلیدی super استفاده می شود آمده است:

- این کلمه کلیدی برای **فرق گذاشتن بین اعضای هم نام** (متغیرها و متدها) کلاس پدر و فرزند می باشد.اگه یادتون باشه کلمه کلیدی this نیز برای فرق گذاشتن بین متغیر محلی و نمونه کلاس بود،کلمه کلیدی super نیز درجاهایی که متغیرها یا متدهای کلاس فرزند و پدر همنام باشند برای صدا زدن متغیرها یا متدهای پدر از کلمه کلیدی super استفاده میکنیم.
- کاربرد دیگر این کلمه کلیدی برای **صدا زدن** سازنده (constructor) **کلاس پدر** در کلاس فرزند می باشد.

یادآوری :

اعضای کلاس یعنی چه؟ به متغیرها و متدهای درون بدنه یک کلاس اعضای کلاس گفته می شود.

فرق گذاشتن بین اعضای هم نام کلاس پدر و فرزند:

هنگامی که یک کلاس،خصوصیات (متغیرها و متدها) کلاس دیگری را به ارث می برد، اگر اعضای کلاس پدر با اعضای کلاس فرزند همنام باشد،برای فرق گذاشتن بین ویژگی ها و رفتارهای کلاس پدر و فرزند از کلمه کلیدی super برای صدا زدن متغیرها و متدهای کلاس پدر استفاده میکنیم.

```
super.variable;
super.method();
```

- با کلمه کلیدی super می توان متغیرها و متد های کلاس پدر را صدا زد و با این کار بین متغیرها و متدهای همنام کلاس پدر و فرزند فرق میگذاریم که از یکدیگر قابل تشخیص شوند.

در مثال زیر کاربرد کلمه کلیدی super را بررسی میکنیم:

```
package javalike;
class Super_class {
   int num = 20;

   // display method of superclass
   public void display() {
      System.out.println("This is the display method of superclass");
   }
}

public class Sub_class extends Super_class {
   int num = 10;

   // display method of sub class
```

```
   public void display() {
      System.out.println("This is the display method of subclass");
   }

   public void my_method() {
      // Instantiating subclass
      Sub_class sub = new Sub_class();

      // Invoking the display() method of sub class
      sub.display();

      // Invoking the display() method of superclass
      super.display();

      // printing the value of variable num of subclass
      System.out.println("value of the variable named num in sub class:"+
sub.num);

      // printing the value of variable num of superclass
      System.out.println("value of the variable named num in super class:"+
super.num);
   }

   public static void main(String args[]) {
      Sub_class obj = new Sub_class();
      obj.my_method();
   }
}
```

خروجی:

```
This is the display method of subclass
This is the display method of superclass
value of the variable named num in sub class:10
value of the variable named num in super class:20
```

- در این برنامه دو کلاس به نام های Super_class و Sub_class داریم، کلاس Sub_class فرزند (زیر کلاس) کلاس Super_class می باشد،زیرا کلاس Sub_class ،کلاس Super_class را به ارث برده است یعنی کلاس Sub_class پدر کلاس Super_class هستش و کلاس Sub_class به تمام ویژگی ها و رفتار های کلاس Super_class دسرسی دارد.
- کلاس Super_class دارای یک متغیر از نوع عدد صحیح به نام num ویک متد که پیامی را در خروجی چاپ میکند به نام display() دارد.

- کلاس Sub_class نیز یک متغیر از نوع عدد صحیح و یک متد همنام با پدر خود و متد دیگری به نام my_method دارد!!!

```
    Sub_class sub = new Sub_class();

      sub.display();

      super.display();
```

- در بدنه کلاس Sub_class ، درون متد my_method ما یک شی از کلاس Sub_class به نام sub ایجاد کرده ایم.در بدنه این متد ،با استفاده از شی sub ، متد display() درون کلاس Sub_class را صدا زده ایم و با کلمه کلیدی super متد display() درون کلاس پدر Sub_class یعنی Super_class را صدا کرده ایم.در اینجا با وجود همنام بودن متد کلاس فرزند و پدر با استفاده از کلمه کلیدی super بین متد فرزند و پدر فرق گذاشته ایم با این کار متدهای همنام بین پدر و فرزند قابل تشخیص می شوند.

```
      System.out.println("value of the variable named num in sub class:"+
sub.num);

            System.out.println("value of the variable named num in super class:"+
super.num);
```

- در اینجا نیز متغیر از نوع عدد صحیح در کلاس فرزند (Sub_class) و کلاس پدر (Super_class) همنام می باشد پس برای تمایز قائل شدن و قابل تشخیص بودن این متغیر که آیا مربوط به پدر (Super_class) هست یا فرزند(Sub_class) از کلمه کلیدی super برای صدا زدن متغیر کلاس پدر (Super_class) استفاده کرده ایم.برای صدا زدن متغیر num مربوط به کلاس فرزند(Sub_class) از شی ایجاد شده از کلاس فرزندمون(Sub_class) یعنی sub استفاده کرده ایم.در خروجی مقدار این متغیر برای کلاس پدر (Super_class) و فرزند(Sub_class) متفاوت است یعنی برای کلاس فرزند(Sub_class) مقدار ۱۰ و برای کلاس پدر (Super_class)این متغیر مقدارش ۲۰ می باشد.پی در اینجا با وجود همنام بودن متغیر کلاس پدر (Super_class) و فرزند(Sub_class) با کلمه کلیدی super بین ویژگی همنام پدر (Super_class) و فرزند(Sub_class) تمایز ایجاد کردیم.

```
   public static void main(String args[]) {
      Sub_class obj = new Sub_class();
      obj.my_method();
   }
```

- در پایان در متد main کلاس Sub_class یک شی از کلاس فرزند (Sub_class) ایجاد کرده و توسط شی متد my_method() را صدا زده ایم با این کار تمام دستورات درون بدنه این متد اجرا می شود.

صدا زدن سازنده کلاس پدر:

اگر یک کلاس خصوصیات (متغیرها و متدهای) کلاس دیگر را به ارث ببرد،کلاس فرزند به صورت اتوماتیک در سازنده (constructor) خود،سازنده پیشفرض (default constructor) کلاس پدر خود را صدا میزند،حال اگر کلاس پدر علاوه بر سازنده پیشفرض (default constructor) ،سازنده پارامتر دار نیز داشت ،برای صدا زدن سازنده پارامتر دار (parameterized constructor) پدر در سازنده فرزند نیاز داریم که از کلمه کلیدی super در سازنده (constructor) فرزند برای صدازدن و مقداردهی به پارامترهای سازنده (constructor) پدر استفاده کنیم.

نحوه پیاده سازی و صدا زدن سازنده پارامتر دار پدر در سازنده فرزند:

```
super(values);
```

super(پارامترهای سازنده پدر)

- نیازی به بکارگیری مستقیم نام سازنده (constructor) پارامتر دار پدر نیست تنها کافیست از کلمه کلیدی super که درون پرانتز جلوی آن پارامترهای سازنده (constructor) مورد نظر کلاس پدر قرار دارد را مقدار دهی کنیم.
- در مثال ساده زیر نحوه صدا زدن سازنده کلاس پدر در کلاس فرزند با استفاده از کلمه کلیدی super بررسی می کنیم:

```
package javalike;
class Superclass {
   int age;

   Superclass(int age) {
      this.age = age;
   }

   public void getAge() {
      System.out.println("The value of the variable named age in super class is:
" +age);
   }
}
```

```
public class Subclass extends Superclass {
   Subclass(int age) {
      super(age);
   }
   public static void main(String argd[]) {
      Subclass s = new Subclass(24);
      s.getAge();
   }
}
```

خروجی:

```
The value of the variable named age in super class is: 24
```

- در این برنامه دو کلاس با نام های Superclass و Subclass داریم،که Subclass کلاس Superclass را به ارث برده است.

```
   Superclass(int age) {
      this.age = age;
   }
```

- در کلاس پدر یعنی Superclass یک سازنده(constructor) پارامتر دار تعریف شده است پس هر کلاسی که این کلاس را به ارث ببرد و فرزند این کلاس باشد لازم و اجرا است!! که سازنده (constructor) پارامتر دار این کلاس را در سازنده(constructor) خود صدا زند و آن را مقدار دهی کند.کار این سازنده(constructor) این است که این یک مقدار را به عنوان پارامتر ورودی میگیرد و آن را درون متغیر نمونه کلاس میریزد.

```
   public void getAge() {
      System.out.println("The value of the variable named age in super class is:
" +age);
   }
```

- متد getAge درون کلاس پدرSuperclass مقدار age را در خروجی چاپ میکند.

```
public class Subclass extends Superclass {
```

- در این خط کد کلاس Subclass خصوصیات کلاس Superclass را به ارث می برد.

```
   Subclass(int age) {
      super(age);
   }
```

- در این دستور سازنده (constructor) کلاس Subclass را مشاهده میکنید،چون کلاسSuperclass را به ارث برده است پس طبق قانون لازم است که سازنده پارامتر دار (parameterized constructor) پدر (Superclass) را در بدنه سازنده(constructor) خود صدا زند،برای این کار از کلمه کلیدی super استفاده شده است که درون ورودی پرانتز آن یک

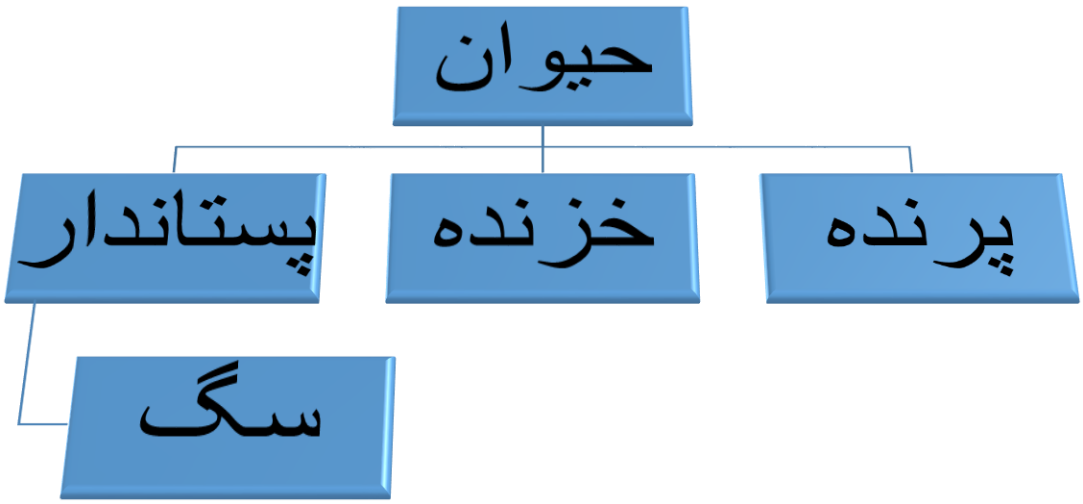
متغیر قرار داده ایم.همان طور که می بینید اسمی از سازنده پارامتر دار پدر (Superclass) آورده نشده است و تنها با کلمه کلیدی super به این سازنده پارامتر دار پدر (Superclass) اشاره کرده ایم.

```
public static void main(String argd[]) {
   Subclass s = new Subclass(24);
   s.getAge();
}
```

- در متد main درون کلاس Subclass یک شی از این کلاس به نام s ساخته و به سازنده آن مقدار ۲۴ را داده ایم و از طریق این شی متد getAge() را صدا زده ایم.

رابطه IS-A (IS-A Relationship):

گاهی چندین کلاس خصوصیات یک کلاس را به ارث می برند.در این حالت شی های ایجاد شده از این کلاس ها از نوع آن کلاس هستند و پدر همه آنها یکی است! برای درک بهتر میریم سراغ دنیای واقعی!!! در دنیای واقعی **پستاندار،خزنده،پرنده** همه **حیوان** هستند و **سگ** یک **حیوان پستاندار** هستش!!! در اینجا یک هرم ایجاد شده است به گونه ای سگ زیر مجموعه پستاندار و همه پستاندار،خزنده و پرنده زیر مجموعه حیوان می باشند یعنی حیوان پدر پستاندار،خزنده و پرنده می باشد نمودار سلسه مراتب زیر را مشاهده کنید:



- حیوان پدر پرنده می باشد.
- حیوان پدر خزنده می باشد.

- حیوان پدر پستاندار می باشد.
- سگ نیز فرزند پستاندار و حیوان می باشد.

خب رابطه IS-A در اینجا چه کاربردی دارد؟!

ما با توجه این نمودار می توانیم بگوییم:

پرنده یک حیوان است.

Bird IS-A Animal

خزنده یک حیوان است.

Reptile IS-A Animal

پستاندار یک حیوان است.

Mammal IS-A Animal

سگ یک پستاندار است.

Dog IS-A Mammal

از این رو سگ یک حیوان است.

Hence: Dog IS-A Animal

در اینجا IS-A یعنی این که نوع X از نوع Y است، در اینجا X نام شی و Y نام کلاس می باشد و در صورتی X از نوع Y هستش که X فرزند Y باشد.

اصطلاح IS-A در جاوا: گاهی تعدادی شی داریم و میخواهیم بفهمیم که این اشیا فرزند کلاس مورد نظر هست یا خیر،برای این کار دست به مقایسه شی با کلاس مورد نظر می زنیم ،که در اینجا رابطه IS-A ،یعنی این شی از نوع کلاس مورد نظر است.
مثلا :

شی پستاندار از نوع حیوان است.

شی پرنده از نوع حیوان است.

اما شی صندلی از نوع حیوان نیست.

شی دمپایی از نوع حیوان نیست.

در کل رابطه IS-A دو جواب مثبت و منفی (درست یا نادرست) به ما می دهد میگه فلان شی از نوع کلاس مورد نظر است یا خیر.

❖ میدونم شاید هنوز مفهوم جا نیانداختم و گنگ باشه اصلا نگران نباشید ادامه را بخوانید حلش میکنیم ☺

به مثال زیر توجه کنید:در اینجا تمام مثال هایی که در دنیای واقعی از حیوانات زدیم رو بصورت کد برنامه نویسی مشاهده میکنید.

```
package javalike;

public class Animal {
}

 class Mammal extends Animal {
}

 class Reptile extends Animal {
}

 class Dog extends Mammal {
}
```

- در اینجا چهار کلاس داریم که دو کلاس Mammal و Reptile وخصوصیات کلاس Animal به ارث برده اند و همچنین کلاس Dog خصوصیات کلاس Mammal به ارث برده است.
- بر اساس مثال بالا ،اصلاحات شی گرایی زیر همگی درست و true هستند:

❖ کلاس Animal پدر کلاس Mammal است.

❖ کلاس Animal پدر کلاس Reptile است.

❖ کلاس های Mammal و Reptile فرزندان کلاس Animal هستند.

❖ کلاس Dog فرزند هر دو کلاس Animal و Mammal است.

❖ کلاس Mammal پدر کلاس Dog است.

حالا اگر رابطه IS-A را برای این کلاس ها در نظر بگیریم بصورت زیر است:

```
Mammal IS-A Animal
```

Mammle یک Animal است. (پستاندار یک حیوان است)

`Reptile IS-A Animal`

`Reptile` یک `Animal` است. (خزنده یک حیوان است)

`Dog IS-A Mammal`

`Dog` یک `Mammal` است. (سگ یک پستاندار است)

`Hence: Dog IS-A Animal`

از این رو: `dog` یک حیوان است (از این رو چون سگ فرزند پستاندار و پستاندار فرزند حیوان هستش ، پس سگ یک حیوان است به نوعی حیوان پدربزرگ سگ هستش ☺)

با استفاده از کلمه کلیدی `extends` ، کلاس فرزند می تواند به تمام خصوصیات کلاس پدر دسرسی داشته باشد به جز خصوصیات،ویژگی هاو رفتار های که از نوع `private` در کلاس پدر باشند.مثل اینکه گویا پدر جعفر پراید را از حالت `public` به `private` تغییر داده و یک قفل بزرگی دمش زده و جعفر دیگه نمیتونه دزدکی پراید رو از خونه خارج کنه ☺☺☺

تا اینجا رابطه `IS-A` را بصورت کلامی و تئوری یادگرفتیم حالا برای درک بهتر سراغ مثال کاربرد این اصطلاح برای مقایسه نوع اشیا با یک کلاس می رویم:

نکته مهم : برای مقایسه یک شی که ایا از نوع کلاس مورد نظر است یا خیر از دستور زیر استفاده میکنیم:

```
instanceof
```

سمت چپ دستور بالا نام شی و سمت راست آن نام کلاس آورده می شود.

مثلا فرض کنید شی ای از کلاس `Dog` به نام `jesi` ساخته ایم! حالا میخوایم بدونیم آیا `jesi` قصه ما از نوع حیوان هستش یا خیر؟!!بصورت زیر عمل میکنیم:

```
jesi instanceof Animal
```

- معنی این خط کد، یعنی آیا شی `jesi` از نوع کلاس `Animal` هستش؟! که این دستور دو جواب `true` یا `false` به ما برمیگرداند. اگر شی `jesi` فرزند،نوه،نتیجه و... کلاس `Animal` بود،مقدار `true` برمیگرداند و در غیر این صورت جواب `false` به ما می دهد.

❖ شکل کلی استفاده از دستور **instanceof** که معنای اصطلاح رابطه IS-A را در جاوا می دهد بصورت زیر است:

```
objectName instanceof className
```

نام کلاس مورد نظر **instanceof** نام شی مورد نظر

حالا مثالی از کاربرد این دستور را بررسی میکنیم:

```
package javalike;

class Animal {
}

class Mammal extends Animal {
}

class Reptile extends Animal {
}

public class Dog extends Mammal {

      public static void main(String args[]) {
            Animal a = new Animal();
            Mammal m = new Mammal();
            Dog d = new Dog();

            System.out.println(m instanceof Animal);
            System.out.println(d instanceof Mammal);
            System.out.println(d instanceof Animal);
      }

}
```

خروجی:

```
true
true
true
```

- در اینجا سه کلاس Mammal و Reptile کلاس Animal را به ارث برده اند و کلاس Dog کلاس Mammal را به ارث برده است.

```
Animal a = new Animal();
Mammal m = new Mammal();
Dog d = new Dog();
```

- سه شی از کلاس های Animal،Mammal و Dog در متد main ایجاد کرده ایم.

```
System.out.println(m instanceof Animal);
```

- شی ساخته شده از کلاس Mammal را با کلاس Animal مقایسه میکنیم.اگر شیm از نوع Animal بود جواب true در غیر این صورت جواب false خواهد بود.

```
System.out.println(d instanceof Mammal);
```

- شی ساخته شده از کلاس Dog را با کلاس Mammal مقایسه میکنیم.

```
System.out.println(d instanceof Animal);
```

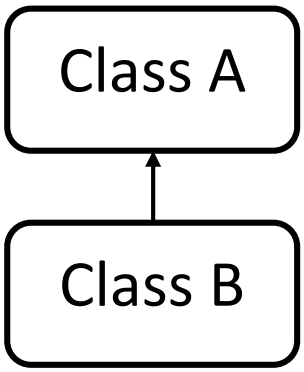
- شی ساخته شده از کلاس Dog را با کلاس Animal مقایسه میکنیم.

انواع ارث بری و وراثت:

در زیر انواع ارث بری را مشاهده میکنید: من همین جوری ی عنوان فارسی براشون نوشتم خیلی سخت نگیرید کلا عنوان رو بیخیال توضیح را بچسبید ☺ ☺

ارث بری تکی(singe inheritance) :

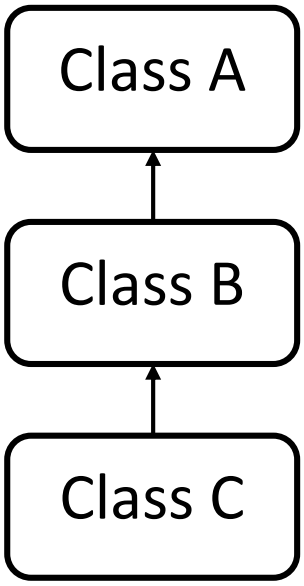
وقتی تنها یک کلاس خصوصیات کلاس دیگر را به ارث می برد.




```
class A {

}
class B extends A {

}
```

ارث بری چندسطحی(Multi Leval Inheritance) :

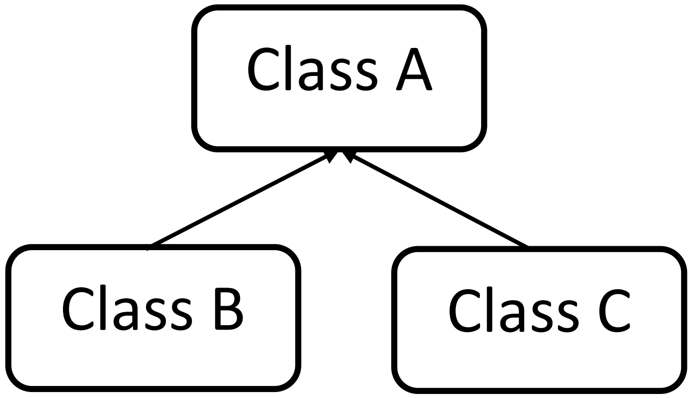
وقتی کلاس C خصوصیات کلاس B و کلاس B خصوصیات کلاس A را به ارث می برد.




```
package javalike;

class A {

}
class B extends A {

}
class C extends B {

}
```

ارث بری سلسه مراتبی (Hierarchical inheritance)

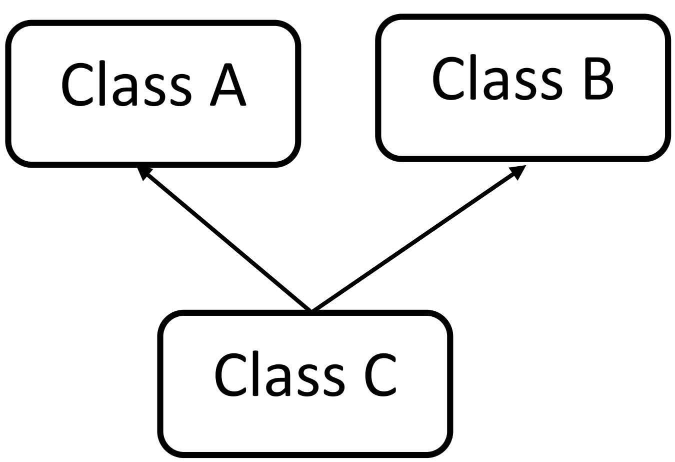
وقتی دو کلاس خصوصیات یک کلاس را به ارث می برند.




```
package javalike;

class A {

}
class B extends A {

}
class C extends A {
}
```

:Multi Inheritance

وقتی یک کلاس بیش از یک کلاس را به ارث می برد!!



```
package javalike;

class A {

}
class B  {

}
class C extends A,B {

}
```

اخطار!!!!!!!!خطای کامپایل!!!!

جاوا از این که یک کلاس خصوصیات بیش از یک کلاس را به ارث ببرد حمایت نمیکند و چنین امکانی در جاوا وجود ندارد!!!! این بدین معناست که یک کلاس تنها یک کلاس دیگر را می تواند به ارث ببرد.

یکی از مهم ترین و پرکاربرد ترین مباحث شی گرایی جاوا مبحث ارث بری یا وراثت هستش که در این جلسه مفاهیم این مبحث همراه با مثال هایی که تنها قصد آن جا انداختن این مفاهیم بود بکار بردیم.

من که دست از ارث بری برای یک جلسه بر نمی دارم چون نمیخوام ی چیزی گفته باشم بره!!!!! پس این جلسه را **بخش اول ارث بری** می نامیم و **در بخش دوم ارث بری** که در جلسات آینده آمادش میکنم بصورت حرفه ای بصورت کاملا عملی و پروژه محور همراه با مثال بزرگ و مسئله ای این مبحث را بازتر میکنیم!!! پس این جلسه رو خوب مطالعه کنید که برای **بخش دوم ارث بری** کاملا اماده باشید ☺

البته تمام مفاهیم ارث بری در این جلسه گفته شده است و حداقل مفاهیم پرکاربرد و مهم آن از قلم نیفتاده است!!! در بخش دوم ارث بری تنها مثال و مسئله حل خواهیم کرد.

پیروز و موفق باشید

سایت آموزش زبان جاوا به زبان ساده،آسان و شیرین!!!

www.JAVAPro.ir

آموزش جاوا SE را با تجربه شخصی و به زبان خودمونی یاد بگیرید!!!!!

بازدید ازکانال

بازدید از سایت

هر روز مفاهیم و مثال های جدید به سایت اضافه می شود برای اطلاع از مطالب جدید روی سایت عضو کانال شوید.