## مقدمه‌ای برای شروع جاوا :

در درس «مبانی برنامه نویسی پیشرفته» با زبان C یا C++ آشنا شدیم و با استفاده از این زبان ها برنامه نویسی روندی را یاد گرفتیم ، اکنون قصد داریم در این درس با استفاده از زبان JAVA برنامه نویسی شی گرا را یاد بگیریم.

تعریف : برنامه نویسی روندی : در این روش مساله‌ای که باید حل شود به چند زیر مساله ساده تر تقسیم می‌شود سپس زیر مساله ها با جواب هایشان با هم ترکیب میشود و مساله را برای ما حل میکند .

تعریف : برنامه نویسی شی گرا : برنامه را به عنوان مجموعه‌ای از اشیا در نظر میگیریم که قرار است با هم تعامل داشته باشند .

در برنامه نویسی شی گرا با دو مفهوم اساسی کلاس (Class) و شی (Object) آشنا میشویم ، که تشخیص و درک این دو مفهوم در ادامه فرایند یادگیری ، ضروری است.

تعریف : کلاس : یک نوع داده است یا یک قسمتی از برنامه که حقیقتی را توصیف میکند.

تعریف : شی : یک نمونه ساخته شده از روی کلاس است.

به زبان ساده تر کلاس را میتوانیم یک الگو یا اسکلت در نظر بگیریم که یک شی با استفاده از آن ساخته میشود.

چرا از برنامه نویسی شی گرا استفاده میکنیم ؟ استفاده از برنامه نویسی روندی در پروژه های واقعی ضعف خودش رو نشون میده ، وقتی که ما قراره برنامه‌ای بزرگ بنویسیم ، استفاده از روش برنامه نویسی روندی باعث بیشتر شدن حجم کد ها و افزایش هزینه ها و زمان نوشتن برنامه میشه ، برنامه نویسی شی گرا میتونه حجم این کد هارو کمتر و کارامد تر و برنامه ما رو بهینه تر کنه .

حالا چرا جاوا ؟ چرا وقتی زبان هایی قدرتمند و ساده مثل پایتون وجود داره باید جاوا رو یاد بگیریم؟

- **کاربرد گسترده :**جاوا یکی از محبوب‌ترین زبان‌های برنامه‌نویسی در دنیاست و برای طیف وسیعی از برنامه‌ها، از جمله برنامه‌های اندرویدی، برنامه‌های سازمانی، وب‌سایت‌ها و سیستم‌های تعبیه‌شده استفاده می‌شه. یادگیری جاوا به شما امکان میده تا در طیف گسترده‌ای از پروژه‌ها کار کنید .
- **عملکرد :**جاوا به طور کلی زبانی سریع و کارآمدتر از پایتونه. این امر به این دلیله که جاوا کامپایل می‌شه، در حالی که پایتون به صورت تفسیری اجرا می‌شه. به این معنی که کد جاوا قبل از اجرا به کد ماشین تبدیل می‌شه، که می‌تونه سرعت اجرای برنامه رو افزایش بده .
- **قابلیت اطمینان:** جاوا به عنوان یک زبان برنامه‌نویسی قوی و قابل اعتماد شناخته شده. این امر به دلیل تمرکز بر روی ایمنی و نوع مدیریت حافظه خودکاره. جاوا همچنین دارای جامعه کاربری بزرگ و فعالی هست که می‌تونه در صورت بروز مشکل به شما کمک کنه.

سفر کدها از دنیای انسان به دنیای کامپیوتر: ماجرای کامپایل شدن

فرض کنید که شما به عنوان یک نویسنده، کتابی به زبان فارسی نوشته‌اید. اما خوانندگان شما فقط زبان انگلیسی را می‌فهمند. در این شرایط، چه کار می‌کنید؟

احتمالاً کتاب خود را به زبان انگلیسی ترجمه می‌کنید تا مخاطبان بیشتری آن را درک کنند.

کامپایل کردن نیز روندی مشابه را در دنیای برنامه‌نویسی دنبال می‌کند . در این فرایند، کدهایی که به زبان‌های برنامه‌نویسی نوشته شده‌اند (زبان انسان) به زبانی که کامپیوتر می‌تواند آن را بفهمد و اجرا کند (زبان ماشین) ترجمه می‌شوند.

چرا به کامپایل کردن نیاز داریم؟

کامپیوترها از زبان انسان سر در نمی‌آورند! زبان برنامه‌نویسی جاوا، زبانی شبیه به زبان انسان است که برای ما قابل فهم و خواندن است. اما کامپیوترها فقط زبان 0 و 1 را می‌فهمند. بنابراین، برای اینکه کامپیوتر بتواند کدهای جاوا را اجرا کند، باید آنها را به زبان 0 و 1 تبدیل کنیم. این کار با استفاده از فرآیندی به نام کامپایل کردن انجام می‌شود.

مشکل معماری‌های مختلف: مشکلی که در اینجا وجود دارد این است که کامپیوترها دارای معماری‌های مختلفی هستند. معماری کامپیوتر به نوع ساختار و قطعات آن اشاره دارد. تفاوت‌ها باعث می‌شود که کدهایی که برای یک کامپیوتر کامپایل شده‌اند ، روی کامپیوتر دیگری اجرا نشوند.

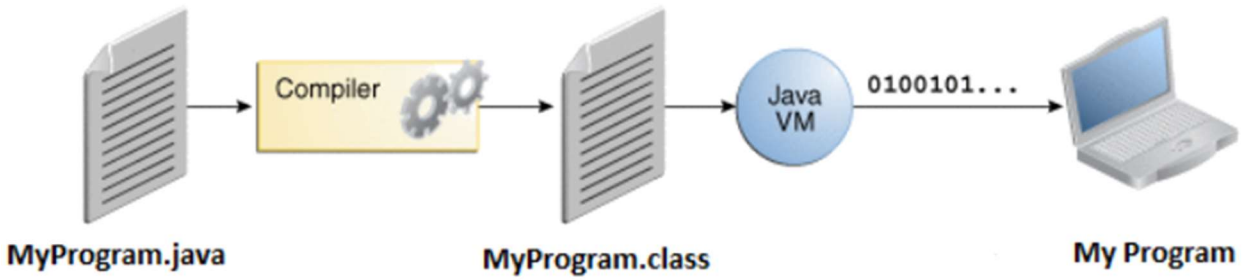
راه حل جاوا : برای حل این مشکل از ماشین مجازی جاوا (JVM) استفاده میکنیم.

جاوا چجوری کار میکنه ؟

برای اجرای برنامه‌های نوشته شده و کامپایل شده به زبان جاوا نیاز به سکویی یا برنامه‌ای است که به آن ماشین مجازی جاوا (Java Virtual Machine)یا به اختصار JVM گفته می‌شود . این ماشین کدهای کامپایل شده به زبان جاوا را گرفته و آنها را اجرا می‌کند .

شاید این جمله را شنیده باشید که کدهای زبان جاوا بر روی هر ماشین قابل اجرا می‌باشند و اصطلاحا جاوا Multi Platform است . شخصی که دستگاهی با سیستم عامل ویندوز دارد ، JVM مربوط به سیستم عامل ویندوز را نصب می‌کند . سپس برنامه‌ای را به زبان جاوا می‌نویسد و آن را کامپایل می نماید . پس از آن برنامه کامپایل شده را برای دوست خود که دستگاه دیگری با سیستم عامل لینوکس دارد ارسال می‌کند . این شخص قبلا JVMمخصوص سیستم عامل لینوکس را بر روی دستگاه خود نصب نموده است . به همین دلیل هیچکدام از این دو نفر لازم نیست نگران باشد که سیستم عامل دستگاهایشان با یکدیگر متفاوت است .

می توان نحوه اجرای کدهای جاوا را به صورت زیر خلاصه کرد :




همانطور که در شکل بالا مشاهده می کنید :

- برنامه‌نویس کدهای خود را درون فایلی با پسوند java. می‌نویسد.
- وقتی برنامه نویس برنامه خود را اجرا می کند، کدهای برنامه توسط کامپایلر جاوا به bytecode تبدیل می‌شوند و درون فایلی با همان نام قبلی اما این بار با پسوند class. ذخیره می‌شوند.
- ماشین مجازی جاوا (Java Virtual Machine) فایل class. را اجرا می کند.

برای شروع هر زبان برنامه نویسی مرسوم است ابتدا نحوه چاپ متن "Hello World!" را در خروجی بیاموزیم.

برای اینکار ما باید یک فایل با پسوند java. بسازیم و نامی دلخواه برای آن انتخاب کنیم (برای مثال Start.java) .

سپس با استفاده از IDE یا ادیتور متن شروع به نوشتن برنامه میکنیم .

هر یک از برنامه های جاوا به عنوان یک کلاس شناخته میشود .

```
public class Start {
    public static void main(String[] args) {
      System.out.println("Hello World!");
    }
  }
```

```
Hello World!
```

هر خط کدی که در جاوا اجرا میشود باید داخل یک Class باشد ، در مثال ما کلاس را Start نام‌گذاری کردیم ، و یک کلاس همیشه باید با حرف بزرگ شروع شود .

**توجه :** جاوا به حروف بزرگ و کوچک حساس است!

به این مساله خیلی باید دقت کنید که نام فایل شما با نام کلاس شما یکی باشد در غیر این صورت جاوا به شما پیغام خطا نشان میدهد.

### متد main() :

در هر فایل جاوا باید متد main() را قرار دهیم . این متد ، متد اصلی برنامه ماست ، وقتی یک برنامه به زبان جاوا را اجرا میکنیم در واقع متد اصلی آن را اجرا میکنیم.

که نحوه تعریف آن به صورت زیر است

```
public static void main(String[] args){...}
```

هر کدی که شامل متد main() باشد اجرا میشود ، نگران کلمات کلیدی مانند public ، static و ... نباشید .

در حال حاضر شما باید بدانید که هر برنامه جاوا یک کلاس دارد که نام کلاس باید با نام فایل یکی باشد و هر برنامه‌ای باید دارای متد main() باشد .

درون متد main() ما میتوانیم از println() برای چاپ کردن متن در خروجی استفاده کنیم .

```java
public static void main(String[] args) {
    System.out.println("Hello World");
  }
```

**توجه :**

- پرانتز ها {} شروع و پایان یک بلوک کد را نشان میدهند ، و ما کد هایمان را درون آن ها مینویسیم .
- System یک کلاس داخلی و از پیش ساخته شده جاوا است که شامل اعضایی مانند out است و متد println() درون آن است که مخفف print line است .
- به یاد داشته باشید که هر دستور را باید با استفاده ازsemicolon به پایان برسانید (;) .

## خروجی ها در جاوا :

با متد println() آشنا شدیم ، با استفاده از آن ما میتوانیم متن مورد نظر خودمان را در خروجی نمایش دهیم ، به این ساده توجه کنید .

```java
public class Main {
  public static void main(String[] args) {
    System.out.println("Hello World!");
    System.out.println("I am learning Java.");
    System.out.println("It is awesome!");
  }
```

```
Hello World!
I am learning Java.
It is awesome!
```

## دابل کوتیشن :

هنگامی که با متن ها کار میکنید باید آن ها را در گیومه یا دابل کوتیش ها قرار دهید "".

اگر شما این کار را انجام ندهید جاوا به شما پیغام خطا نمایش میدهد .

```java
System.out.println("This sentence will work!");
System.out.println(This sentence will produce an error);
```

```
    System.out.println(This sentence will produce an error);
                       ^
```

```
Main.java:3: error: ';' expected
    System.out.println(This sentence will produce an error);
                                          ^
Main.java:3: error: ';' expected
    System.out.println(This sentence will produce an error);
                                                  ^
Main.java:3: error: not a statement
    System.out.println(This sentence will produce an error);
                                                   ^
Main.java:3: error: ';' expected
    System.out.println(This sentence will produce an error);
                                                     ^
5 error
```

### متد print() :

متد print() هم مانند متد println() عمل میکند و تنها تفاوت بین آنها این است که متد print() بر خلاف println() در خروجی سطر جدیدی درج نمیکند.

```java
public class Main {
  public static void main(String[] args) {
    System.out.print("Hello World! ");
    System.out.print("I will print on the same line.");
  }
}
```

```
Hello World! I will print on the same line.
```

همانطور که مشاهده کردید به دلیل اینکه ما از متد print() استفاده کردیم پس از چاپ Hello World! سطر درج نشد و دستورات بعدی هم در همان سطر نمایش داده شد .

```java
public class Main {
  public static void main(String[] args) {
    System.out.println(3);
    System.out.println(358);
    System.out.println(50000);
  }
}
```

```
3
358
50000
```

همچنین میتوانیم محاسبات ریاضی را با استفاده از ()println در خروجی نمایش دهیم .

به مثال زیر توجه کنید :

```
public class Main {
  public static void main(String[] args) {
    System.out.println(3 + 3);
  }
}
```

```
3
```

یا ضرب دو عدد :

```
public class Main {
  public static void main(String[] args) {
System.out.println(2 * 5);
  }
}
```

```
10
```

**توجه و یادآوری :** برای نمایش خود عبارت (3 + 3) در خروجی باید این عبارت را در بین دابل کوتیشن قرار دهیم !

```
public class Main {
  public static void main(String[] args) {
System.out.println("3 + 3");
  }
}
```

```
3 + 3
```

## کامنت گذاری :

برای اینکه قسمتی از برنامه خود را توضیح دهیم برای نفر دیگری که کد ما را میخواند میتوانیم در کنار آن قسمت کامنتی بگذاریم که آن قسمت کد را توضیح دهد .

استفاده دیگر از کامنت گذاری برای وقتی است که قسمتی از کد را میخواهیم غیر فعال کنیم که اجرا نشود ، اما نمیخواهیم آن را پاک کنیم ، پس آن را به اصطلاح کامنت میکنیم .

برای کامنت نویسی ابتدا دو اسلش قرار میدهیم (//) سپس کامنت را بعد از آن مینویسیم

برای کامنت کردن یک خط هم میتوانیم به ابتدای آن خط برویم و دو اسلش را در آنجا درج کنیم.

```java
public class Main {
  public static void main(String[] args) {
    // This is a comment
    System.out.println("Hello World");
  }
}
```

```
Hello World
```

همانطور که مشاهده کردید جاوا هنگام اجرای برنامه کامنت ها را در نظر نمیگیرد .

### کامنت های چند خطی :

گاهی ممکن است متنی که میخواهید کامنت کنید چند خط باشد و برای راحتی کار میتوانید ابتدای متنی که میخواهید کامنت شود را با «/*» و انتهای متن را با «*/» برای جاوا علامت گذاری کنید تا متن شما کامنت شود .

به مثال زیر توجه کنید :

```java
public class Main {
  public static void main(String[] args) {
    /* The code below will print the words Hello World
    to the screen, and it is amazing */
    System.out.println("Hello World");
  }
}
```

```
Hello World
```

## متغیرها :

متغیرها محفظه هایی برای ذخیره مقادیر داده ها هستند.

درجاوا، انواع مختلفی از متغیرها رو داریم:

برای مثال اعداد صحیح را با `int` میشناسیم و اعداد اعشاری را با `float` یا رشته های متنی را با `String` میشناسیم.

در اینجا مثال هایی از انواع داده ها زدیم :

- `String` : رشته های متنی را با دابل کوتیشن در این نوع متغیر ها ذخیره میکنیم ، مثل`"Hello World!"` .
- `int` : اعداد صحیح بدون اعشار را در خود ذخیره میکند ، مثل 124 یا 123- .
- `float` : اعداد اعشاری را درون خود ذخیره میکند ، مثل 19.99 یا 12.5- .
- `char` : یک تک کاراکتر را با استفاده از سینگل کوتیشن میتوانیم در آن ذخیره کنیم ( `'a'` و `'b'` )
- `boolean` : تنها دو مقدار درست و نادرست را میتوانیم در آن ذخیره کنیم (true و false)

## تعریف متغیر در جاوا :

```
type variableName = value;
```

- type:نوع متغیر
- variableName : نام متغیری که میخواهیم تعریف کنیم
- value : مقدار متغیر

مثال : یک متغیر از نوع String تعریف کنید و یک متن را در آن ذخیره کنید ، و سپس آن متغیر را با استفاده از ()println در خروجی نمایش دهید .

```
public class Main {
  public static void main(String[] args) {
    String name = "John";
    System.out.println(name);
  }
}
```

```
John
```

در این مثال ما یک متغیر عددی از نوع int تعریف میکنیم و سپس آن را چاپ میکنیم .

```
public class Main {
  public static void main(String[] args) {
    int myNum = 15;
    System.out.println(myNum);
  }
}
```

```
15
```

شما میتوانید متغیر را تعریف کنید بدون مقدار دهی اولیه و بعدا آن را مقدار دهی کنید ، برای درک این موضوع به این مثال توجه کنید.

```
public class Main {
  public static void main(String[] args) {
    int myNum;
    myNum = 20;
    System.out.println(myNum);
  }
}
```

```
20
```

ما میتوانیم مقدار متغیری که تعریف و مقدار دهی شده را تغییر دهیم ، این موضوع را در مثال زیر نمایش دادیم .

```java
public class Main {
  public static void main(String[] args) {
    int myNum = 15;
    myNum = 20;  // myNum is now 20
    System.out.println(myNum);
  }
}
```

```
20
```

## متغیر های final :

ما میتوانیم برای جلوگیری از تغییر مقدار متغیر هایی که تعریف کردیم آن ها را به صورت final تعریف کنیم .

متغیر های final تغییر پذیر نیستند و در صورتی که تلاشی برای تغییر آنها صورت گیرد با پیغام خطا رو به رو میشویم .

برای مثال :

```java
public class Main {
  public static void main(String[] args) {
    final int myNum = 15;
    myNum = 20; // will generate an error
    System.out.println(myNum);
  }
}
```

```
Main.java:4: error: cannot assign a value to final variable myNum
    myNum = 20; // will generate an error
    ^
1 error
```

## انواع دیگر متغیر ها :

```java
int myNum = 5;
float myFloatNum = 5.99f;
char myLetter = 'D';
boolean myBool = true;
String myText = "Hello";
```

در قسمت های بعدی انواع متغیر ها را بیشتر مورد بحث قرار میدهیم .

برای ترکیب کردن متن و متغیر (الحاق کردن) از کاراکتر + استفاده میکنیم.

```
public class Main {
  public static void main(String[] args) {
    String name = "Ali";
    System.out.println("Hello " + name);
  }
}
```

```
Hello Ali
```

شما میتوانید با استفاده از کارکتر + یک متغیر را به یک متغیر دیگر بچسبانید یا اضافه کنید ( با توجه به نوع داده ای متغیر ها) .
در مثال زیر عمل الحاق بین دو متغیر از نوع رشته انجام میشود و هر دو رشته را به هم میچسباند :

```
public class Main {
  public static void main(String[] args) {
    String firstName = "Ali ";
    String lastName = "Ebrahimi";
    String fullName = firstName + lastName;
    System.out.println(fullName);
  }
}
```

```
Ali Ebrahimi
```

کاراکتر + وقتی بین دو داده عددی قرار بگیرد به عنوان عملگر ریاضی عمل میکند و دو عدد را با هم جمع میکند :

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    int y = 6;
    System.out.println(x + y); // Print the value of x + y
  }
}
```

```
11
```

دیدید که در مثال بالا به دلیل اینکه نوع داده ها عددی بود ، کاراکتر + به عنوان عملگر ریاضی کار کرد . ولی اگر حتی یکی از متغیر ها رشته میبودند کاراکتر + به عنوان یک عملگر الحاق کار میکرد . مثل این مثال :

```
public class Main {
  public static void main(String[] args) {
      int x = 5;
      String y = "6";
      System.out.println(x + y); // Print the combine of x + y
  }
}
```

```
65
```

### رفتار کاراکتر + با نوع داده ای char چگونه است؟

برای اینکه این قسمت را بهتر متوجه شویم باید نوع داده ای char را بهتر بشناسیم که در ادامه آموزش متوجه آن خواهید شد .

```
public class Main {
  public static void main(String[] args) {
      char x = 'a';
      char y = 'b';
      System.out.println(x + y); //Prints the sum of the ASCII values of
characters
  }
}
```

```
195
```

چرا خروجی عدد 195 را نمایش داد؟

جمع کردن دو کاراکتر به جمع کردن کدهای ASCII آنها منجر می شود.

در واقع جدول ASCII کد استاندارد آمریکایی برای تبادل اطلاعات است، یک جدول است که 128عدد را از 0 تا 127 به کاراکترها و نمادهای خاص اختصاص می‌دهد. این جدول به عنوان یک استاندارد بین‌المللی برای نمایش متن در کامپیوترها و سایر دستگاه‌های الکترونیکی استفاده می‌شود.

عدد 97 در جدول ASCII به حرف کوچک 'a' و 98 به حرف 'b' اختصاص داده شده است.

به همین خاطر وقتی این دو کاراکتر را با هم جمع میکنیم حاصل 195 میشود.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | NUL | 16 | DLE | 32 | | 48 | 0 | 64 | @ | 80 | P | 96 | ` | 112 | p |
| 1 | SOH | 17 | DC1 | 33 | ! | 49 | 1 | 65 | A | 81 | Q | 97 | a | 113 | q |
| 2 | STX | 18 | DC2 | 34 | " | 50 | 2 | 66 | B | 82 | R | 98 | b | 114 | r |
| 3 | ETX | 19 | DC3 | 35 | # | 51 | 3 | 67 | C | 83 | S | 99 | c | 115 | s |
| 4 | EOT | 20 | DC4 | 36 | $ | 52 | 4 | 68 | D | 84 | T | 100 | d | 116 | t |
| 5 | ENQ | 21 | NAK | 37 | % | 53 | 5 | 69 | E | 85 | U | 101 | e | 117 | u |
| 6 | ACK | 22 | SYN | 38 | & | 54 | 6 | 70 | F | 86 | V | 102 | f | 118 | v |
| 7 | BEL | 23 | ETB | 39 | ' | 55 | 7 | 71 | G | 87 | W | 103 | g | 119 | w |
| 8 | BS | 24 | CAN | 40 | ( | 56 | 8 | 72 | H | 88 | X | 104 | h | 120 | x |
| 9 | HT | 25 | EM | 41 | ) | 57 | 9 | 73 | I | 89 | Y | 105 | i | 121 | y |
| 10 | LF | 26 | SUB | 42 | * | 58 | : | 74 | J | 90 | Z | 106 | j | 122 | z |
| 11 | VT | 27 | ESC | 43 | + | 59 | ; | 75 | K | 91 | [ | 107 | k | 123 | { |
| 12 | FF | 28 | FS | 44 | , | 60 | < | 76 | L | 92 | \ | 108 | l | 124 | \| |
| 13 | CR | 29 | GS | 45 | - | 61 | = | 77 | M | 93 | ] | 109 | m | 125 | } |
| 14 | SO | 30 | RS | 46 | . | 62 | > | 78 | N | 94 | ^ | 110 | n | 126 | ~ |
| 15 | SI | 31 | US | 47 | / | 63 | ? | 79 | O | 95 | _ | 111 | o | 127 | DEL |

جدول ***ASCII***

## تعریف کردن چند متغیر :

وقتی ما میخواهیم چند متغیر که همه آنها از یک نوع هستند را تعریف کنیم نیازی نیست همه آنها را جداگانه تعریف کنیم ، به مثال زیر دقت کنید :

```
int x = 5;
int y = 6;
int z = 50;
System.out.println(x + y + z);
```

```
61
```

اکنون همه آنها را یکجا تعریف میکنیم :

```
int x = 5, y = 6, z = 50;
System.out.println(x + y + z);
```

```
61
```

**یک مقدار و چند متغیر :**

ما حتی میتوانیم یک مقدار را همزمان به چند متغیر اختصاص دهیم :

```
int x, y, z;
x = y = z = 50;
System.out.println(x + y + z);
```

```
150
```

در این مثال متغیر های x و y و z را همزمان برابر با مقدار 50 قرار دادیم

به هر حال شما میتوانید از هر روشی برای تعریف متغیر استفاده کنید ،

اما باید به یک سری قوانین و قرارداد ها برای نام گذاری متغیر ها در برنامه نویسی دقت کنید .

- نام ها میتوانند شامل حروف ، اعداد ، ( _ ) و $
- نام ها باید با حروف شروع شوند
- نام ها باید با حروف کوچک شروع شوند و نمیتوانند شامل فضا های خالی باشند
- نام ها میتوانند با $ شروع شوند (ما در این آموزش از آن استفاده نمیکنیم)
- نام ها به حروف کوچک و بزرگ حساس هستند
- نمیتوانید از نام های از قبل رزرو شده در جاوا برای تعریف متغیر استفاده کنید ، مثل int یا boolean
- متغیر های final را با حروف کاملا بزرگ تعریف میکنیم مثل (PI , MY_NUM , MY_VAR)

## انواع داده ها در جاوا :

داده های **اولیه** و داده های **مرجع** .

**اولیه** مثل : byte, short, int, long, float, double, boolean و char .

**مرجع** مثل : String و آرایه ها و کلاس ها که بعدا به همه آنها میپردازیم .

| نوع داده | اندازه | جزئیات |
|---|---|---|
| byte | 1 byte | $[-2^7 , 2^7-1]$ |
| short | 2 bytes | $[-2^{15} , 2^{15}-1]$ |
| int | 4 bytes | $[-2^{31} , 2^{31}-1]$ |
| long | 8 bytes | $[-2^{63} , 2^{63}-1]$ |
| float | 4 bytes | برای ذخیره اعداد اعشاری از 6 تا 7 رقم اعشار استفاده میشود |
| double | 8 bytes | برای ذخیره اعداد اعشاری تا 15 رقم اعشار استفاده میشود |
| boolean | 1 bit | دو مقدار درست یا نادرست را ذخیره میکند ( 0 یا 1) |
| char | 2 bytes | یک کاراکتر / حرف یا کد ASCII را ذخیره میکند |

در مثال های قبل با برخی از انواع داده آشنا شدیم ، برای مثال از int برای ذخیره اعداد استفاده میکنیم ولی int همانطور که در جدول مشاهده میکنید دارای محدودیت برای ذخیره اعداد است ، در مواقعی که به اعداد بزرگ تر نیاز داریم میتوانیم از long استفاده کنیم ، یا در برخی موارد که نیاز داریم عددی محدود و کوچک را در متغیر ذخیره کنیم به جای int از short یا byte استفاده میکنیم .

و از double هم برای مواقعی استفاده میکنیم که تعداد اعشار های عدد ما از 7 رقم بیشتر باشد ، در آن زمان float توانایی ذخیره عدد را در خود ندارد و آن را به سمت 7 یا 6 رقم اعشار گرد میکند .

برای تعریف متغیر های long باید در انتهای آنها L را قرار دهیم مثل زیر :

```
long num1 = 15000000000L;
```

همچنین برای تعریف float و double هم باید در انتهای آنها به ترتیب f و d قرار دهیم

```
float num2 = 5.75f;
double num3 = 19.99d;
```

## نماد علمی :

تعریف اعداد با نماد علمی مثل $2x10^{12}$ را میتوانیم به این صورت انجام دهیم :

```
float f1 = 35e3f;
double d1 = 12E4d;
System.out.println(f1);
System.out.println(d1);
```

```
35000.0
120000.0
```

در بسیاری مواقع در برنامه‌نویسی به موقعیت‌هایی برخورد میکنید که نیاز دارید یک متغیر تعریف کنید که دارای دو مقدار باشد مثل

- خاموش / روشن
- بله / خیر
- درست / نادرست

برای این جور مواقع بهترین انتخاب boolean ها است .

```
boolean isJavaFun = true;
boolean isFishTasty = false;
System.out.println(isJavaFun);     // Outputs true
System.out.println(isFishTasty);   // Outputs false
```

```
true
false
```

**توجه :** به جای true و false میتوانیم از 0 و 1 استفاده کنیم (0 با false **هم ارزش است و** 1 با true )

یک مثال دیگر از انواع مختلف داده ای :

```
// Create variables of different data types
int items = 50;
float costPerItem = 9.99f;
float totalCost = items * costPerItem;
char currency = '$';

// Print variables
System.out.println("Number of items: " + items);
System.out.println("Cost per item: " + costPerItem + currency);
System.out.println("Total cost = " + totalCost + currency);
```

```
Number of items: 50
Cost per item: 9.99$
Total cost = 499.50$
```

## داده های غیر ابتدایی (مرجع):

داده های غیر ابتدایی را داده های مرجع مینامیم **زیرا به اشیاء اشاره دارند.**

تفاوت اصلی بین داده های ابتدایی و مرجع عبارتند از :

- داده های ابتدایی از قبل توسط خود جاوا تعریف شده است اما داده های مرجع توسط برنامه نویس ایجاد میشود (به جز `String`)
- ما میتوانیم بر روی یک داده مرجع متد هایی را فراخوانی کنیم ولی روی داده های ابتدایی نمیتوانیم این کار را انجام دهیم .
- داده های ابتدایی همیشه مقداری دارند (حتی اگر مقدار دهی نشده باشند) اما داده های مرجع میتوانند خالی باشند(`null`).
- نام گذاری داده های ابتدایی با حروف کوچک صورت میگیرد در صورتی که داده های مرجع را با حروف بزرگ تعریف میکنیم .

در درس های آینده بعد از آشنا شدن با کلاس ها این موضوع را بهتر درک خواهید کرد اکنون شاید این مبحث را کاملا درک نکنید .

## تغییر نوع یا Type Casting :

برای زمانی است که شما سعی میکنید نوع داده یک متغیر اولیه را به نوعی دیگر غیر از خودش تغییر دهید (مثلا از `int` به `long`).

در جاوا دو نوع تغییر نوع داریم :

تبدیل نوع **ضمنی** (به صورت خودکار انجام میشود) : وقتی نوع یک داده به نوعی دیگر که اندازه بزرگ تری دارد تبدیل میشود (برای مثال `int` که اندازه آن 4 بایت بود به `long` که 8 بایت بود تبدیل میشود).

به ترتیب از کوچک به بزرگ :

`byte` -> `short` -> `char` -> `int` -> `long` -> `float` -> `double`

```java
public class Main {
      public static void main(String[] args) {
        int myInt = 9;
        double myDouble = myInt; // Automatic casting: int to double

        System.out.println(myInt);      // Outputs 9
        System.out.println(myDouble);   // Outputs 9.0
      }
    }
```

```
9
9.0
```

تبدیل نوع **صریح** (به صورت دستی باید انجام شود) : وقتی که سعی میکنیم نوع یک متغیر را به نوعی دیگر که اندازه کمتری دارد تبدیل کنیم (برای مثال `long` را به `int` تبدیل کنیم و شما میدانید که اندازه `long` بیشتر از `int` است) .

```
public class Main {
  public static void main(String[] args) {
    double myDouble = 9.78d;
    int myInt = (int) myDouble; // Manual casting: double to int

    System.out.println(myDouble);   // Outputs 9.78
    System.out.println(myInt);      // Outputs 9
  }
}
```

```
9.78
9
```

## عملگر های جاوا :

در جاوا عملگر ها به این دسته ها تقسیم میشوند :

- عملگرهای حسابی
- عملگرهای انتساب
- عملگرهای مقایسه
- عملگرهای منطقی

**عملگرهای حسابی** : برای انجام عملیات ریاضی رایج انجام میشود .

| مثال | توضیحات | نام | عملگر |
|---|---|---|---|
| x + y | دو مقدار را با هم جمع میکند | جمع | + |
| x - y | یک مقدار را از دیگری کم میکند | تفریق | - |
| x * y | دو مقدار را در هم ضرب میکند | ضرب | * |
| x / y | یک مقدار را بر دیگری تقسیم میکند | تقسیم | / |
| x % y | باقی مانده تقسیم را برمیگرداند | باقی مانده | % |
| x++ | به مقدار متغیر 1 واحد اضافه میکند | افزایش | ++ |
| x-- | از مقدار متغیری یک واحد کم میکند | کاهش | -- |

**عملگرهای انتساب** : از عملگر انتساب میتوانیم یک مقدار را به یک متغیر اختصاص دهیم .

```
int x = 10;
```

مثال :

**انتساب جمع** : یک مقدار را فعلی متغیر اضافه میکند .

```
int x = 10;//x is 10
x += 5;//now x is equal to 15
```

بقیه آنها را در جدول توضیح داده و به دلیل ساده بودن موضوع و جلوگیری از زیاد شدن مثال ها از زدن مثال برای هر کدام از عملگر ها پرهیز شده است شما میتوانید خودتان آنها را امتحان کنید ، توضیحات کافی در جدول موجود است .

| عملگر | مثال | اتفاق که واقعا می‌افته |
|---|---|---|
| = | x = 5 | x = 5 |
| += | x += 3 | x = x + 3 |
| -= | x -= 3 | x = x - 3 |
| *= | x *= 3 | x = x * 3 |
| /= | x /= 3 | x = x / 3 |
| %/ | x %= 3 | x = x % 3 |

**عملگر های مقایسه‌ای :** عملگر های مقایسه برای مقایسه دو مقدار (یا دو متغیر) استفاده میشود و نتیجه مقایسه را به صورت true و false برمیگرداند و ما از این خاصیت در حلقه های for و if و while و در بسیاری دیگر موارد استفاده خواهیم کرد .

با یک مثال بسیار ساده با این موضوع بهتر آشنا شوید :

```
int x = 5;
int y = 3;
System.out.println(x > y); // returns true, because 5 is higher than 3
```

```
true
```

| عملگر | نام | مثال |
|---|---|---|
| == | مساوی | x == y |
| != | نامساوی | x != y |
| > | بزرگ تر و کوچک تر | x > y |
| < | | x < y |
| >= | بزرگ‌تر مساوی و کوچک‌تر مساوی | x >= y |
| <= | | x <= y |

**عملگر های منطقی** : خروجی عملگر های منطقی هم مانند عملگرهای مقایسه‌ای true و false است و برای تعیین منطق بین متغیر ها و مقادیر استفاده میشود .

| عملگر | نام | توضیحات | مثال |
|---|---|---|---|
| && | و | اگر ارزش هر دو طرف آن true باشد مقدار true را برمیگرداند | x<10 && x<5 |
| \|\| | یا | اگر حداقل یکی از آنها true باشد مقدار true را برمیگرداند | x<4 \|\| x<5 |
| ! | نقیض | نتیجه را برعکس میکند | !(x<10 && x<5) |

## رشته ها(String) :

از رشته ها برای ذخیره متن استفاده میکنیم ، یک رشته شامل یک مجموعه از کاراکتر ها است که با استفاده از دو دابل کوتیشن در ابتدا و انتهای آن در متغیر هایی از نوع رشته ذخیره میشود .

```
String greeting = "Hello";
```

در اینجا با برخی ویژگی ها و خواص رشته ها آشنا میشویم .

**طول رشته :** برای به دست آوردن طول رشته میتوانیم بر روی String خود متد length() را فراخوانی میکنیم که خروجی آن عدد است .

```
String txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
System.out.println("The length of the txt string is: " + txt.length());
```

```
The length of the txt string is: 26
```

متد های بیشتری را ما میتوانیم روی رشته ها استفاده کنیم ، برای مثال  و  را برسی میکنیم .

```
String txt = "Hello World";
System.out.println(txt.toUpperCase());   // Outputs "HELLO WORLD"
System.out.println(txt.toLowerCase());   // Outputs "hello world"
```

```
HELLO WORLD
hello world
```

**متد** indexOf() : این یکی با مثال خیلی ساده میشه فهمیدش ، توضیحات پایین مثال:

```
String txt = "Please locate where 'locate' occurs!";
System.out.println(txt.indexOf("locate")); // Outputs 7
```

```
7
```

این متد روی txt فراخوانی میشود و یک ورودی از جنس متن مثل "locate" را دریافت میکند و سعی میکند محل اولین مکانی که "locate" در آنجا وجود دارد را برگرداند .

**توجه :** جاوا شمارش خانه ها را از 0 شروع میکند .

متد چیست؟ در واقع همون توابع هستند که یک سری ورودی هایی رو دریافت میکنند و خروجی هایی دارند که بعد ها مفصل راجع به آنها صحبت خواهیم کرد ، اکنون در همین حد بدانید کفایت میکند .

## الحاق رشته ها :

قبلا این مورد را برسی کردیم و در اینجا یک متد برای ملحق کردن دو رشته به هم استفاده میکنیم .

قبلا با استفاده از عملگر + دو رشته را به هم میچسباندیم ، به این عمل «**concatenation**» میگوییم (یا همون الحاق).

```
String firstName = "Ali";
String lastName = "Ebrahimi";
System.out.println(firstName + " " + lastName);
```

```
Ali Ebrahimi
```

توجه کنید که ما (" ") را برای خوانا شدن خروجی اضافه کردیم.(فاصله اضافه کردیم)

اکنون همین کار را با استفاده از متد ()concat انجام میدهیم :

```
String firstName = "Ali ";
String lastName = "Ebrahimi";
System.out.println(firstName.concat(lastName));
```

```
Ali Ebrahimi
```

متد ()concat روی یک متغیر از نوع String مثل firstName فراخوانی میکنیم و سپس به عنوان ورودی متد ، متنی را وارد میکنیم که قصد داریم به متغیرمان (firstName) بچسبانیم ؛ که در این مثال ما میخواهیم lastName را به firstName بچسبانیم .

**توجه :** همانطور که در ابتدا گفته شد ، وقتی که یک عدد با یک رشته توسط عملگر + جمع میشوند عمل الحاق صورت میگیرد .

## کاراکتر های خاص در رشته ها :

فرض کنید قصد دارید متنی را چاپ کنید که شامل دابل کوتیشن باشد مانند مثال زیر ممکن است به مشکل بخوریم .

```
String txt = "We are the so-called "Vikings" from the north";
```

```
Main.java:3: error: ';' expected
    String txt = "We are the so-called "Vikings" from the north.";
                                       ^
Main.java:3: error: ';' expected
    String txt = "We are the so-called "Vikings" from the north.";
                                             ^
2 errors
```

برای حل این مشکل از بک اسلش (\) استفاده میکنیم ، بک اسلش کاراکتر های خاص را به کاراکتر های رشته ای تبدیل میکند .

| کاراکتر گریز | نتیجه | توضیح |
|---|---|---|
| \' | ' | **یک سینگل کوتیشن درج میکند** |
| \" | " | **یک دابل کوتیشن درج میکند** |
| \\ | \ | **یک بک اسلش درج میکند** |
| \n | New line | **یک خط جدید درج میکند** |
| \t | Tab | **یک فاصله به اندازه یک تب (معمولا 5 اسپیس درج میکند)** |
| \b | Backspace | |
| \r | return | |

با توجه به این جدول مثال قبل را اصلاح کردیم :

```
String txt = "We are the so-called \"Vikings\" from the north.";
```

## کلاس «Math» :

این کلاس در پکیج java.lang وجود دارد و به صورت پیش فرض بدون `import` کردن قابل دسترسی است و ما میتوانیم از متد های این کلاس استفاده کنیم (اگر معنی پکیج ها و `import` کردن پکیج در جاوا را نمیدانید نگران نباشید در آینده به آن ها پرداخته خواهد شد)

متد `Math.max(x,y)` : وظیفه این متد همانطور که از اسمش پیداست و حدس میزنید برگرداندن عدد بزرگ تر از بین `x` و `y` وارد شده است . (x و y آرگومان های ورودی این متد هستند)

متد `Math.min(x,y)` : از بین آرگومان های ورودی عدد کوچک تر را برمیگرداند .

| متد | عملیات | مثال | |
|---|---|---|---|
| | | ورودی | خروجی |
| `Math.max(x,y)` | تشخیص عدد بزرگ تر | `Math.max(3,5)` | **5** |
| `Math.min(x,y)` | تشخیص عدد کوچک تر | `Math.min(3,5)` | **3** |
| `Math.pow(x,y)` | محاسبه توان | `Math.pow(2,5)` | **25** |
| `Math.sqrt(x)` | محاسبه جذر | `Math.sqrt(9)` | 3 |
| `Math.abs(x)` | محاسبه قدرمطلق | `Math.abs(-9.4)` | **9.4** |
| `Math.random()` | ایجاد عدد تصادفی | `Math.random()` ورودی ندارد | **0.46234223458** |

برای تولید عدد تصادفی به جز استفاده از متد Math.random() که درون بسته java.lang وجود دارد راه دیگری هم وجود دارد.

درون بسته java.util یک کلاس به نام Random وجود دارد که ما میتوانیم با ساختن شی از روی آن ، عدد تصادفی ایجاد کنیم که دارای مزیت هایی نسبت به روش اول هست که در ادامه توضیح داده میشود .

البته برای ساختن شی از روی کلاس Random باید لازم است آن را import کنیم .

(ساختن شی از روی کلاس را بعدا توضیح داده خواهد شد ، اکنون فقط این مورد خاص را به یاد داشته باشید )

برای import کردن پکیجی که کلاس مد نظر ما درون آن قرار دارد باید با استفاده از این دستور در ابتدای کد آن پکیج را import کنیم (در این مثال کلاس Random درون پکیج java.util قرار دارد به همین خاطر به این صورت آن را import میکنیم: )

```
import java.util.Random;
```

در مثال زیر به ترتیب این کار ها را انجام دادیم :

- Import کردن Random
- ساختن شی از روی Random که نام آن را rand گذاشتیم
- سپس دو متغیر تعریف کردیم به نام های randNumber1 و randNumber2 که نحوه مقدار دهی آنها با استفاده از متد nextInt() است ، این متد روی شی فراخوانی شده (برای فراخوانی و استفاده از آن لازم است شی از روی کلاس ساخته شود) و آرگومان ورودی آن یک عدد صحیح است ، این متد آرگومان ورودی مثل 1000 را دریافت میکند و سپس از بین بازه 0 تا 999 یک عدد را برمیگرداند (که در متغیر ما ذخیره میشود) .
- دو عدد صحیح تصادفی را چاپ کردیم
- دو متغیر از نوع double تعریف کردیم و سپس با استفاده از متد nextDouble() متغیر هایمان را مقدار دهی کردیم ، این متد آرگومان ورودی ندارد و یک عدد اعشاری از بین 0 تا 1 را برمیگرداند (مثل عدد 0.7288425427367139 )

```
import java.util.Random;// importin Random Class from java.util;

public class Main{

    public static void main(String args[]){
        Random rand = new Random();// create an Object of Random class

        // Generate random integers in range 0 to 999
        int randNumber1 = rand.nextInt(1000);
        int randNumber2 = rand.nextInt(1000);

        System.out.println(randNumber1);
        System.out.println(randNumber2);

        // Generate Random doubles in range 0 to 1
        double randNumber3 = rand.nextDouble();
        double randNumber4 = rand.nextDouble();

        // Print random doubles
        System.out.println(randNumber3);
```

```
        System.out.println(randNumber4);
    }
}
```

```
44
28
0.6357036872644626
0.7554176293559371
```

## پکیج ها :

پکیج ها در جاوا پوشه هایی هست که شامل یک سری از کلاس هاست ، و ما با import کردن آنها میتوانیم از آن کلاس شی بسازیم و استفاده کنیم .

برای استفاده از کلاس ها و پکیج ها با استفاده از کلمه کلیدی import این امر را ممکن میکنیم :

```
import package.name.Class;   // Import a single class
import package.name.*;   // Import the whole package
```

### دریافت ورودی در جاوا با استفاده از Scanner :

کلاس Scanner یکی از کلاس های موجود در پکیج java.util است که باید قبل از استفاده import شود .

```
import java.util.Scanner;
```

نکته : ما میتوانیم همه موارد موجود در یک پکیج را به یک باره import کنیم با استفاده از کاراکتر * ، برای مثال به این صورت :

```
import java.util.*;
```

**نحوه استفاده از کلاس Scanner :**

- یک نمونه از کلاس میسازیم با نام دلخواه .(مثل input )
- یکی از متد که در ادامه گفته میشود را بر روی شئ که ساختیم فراخوانی میکنیم و آن مقدار را در متغیر یا هر جا که نیاز است استفاده میکنیم .

```
import java.util.Scanner;  // Import the Scanner class
public class Main {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);// Create a Scanner object as
input
    System.out.println("Enter username :");

    String userName = input.nextLine();  // Read user input
    System.out.println("Username is: " + userName);  // Output user input
  }
}
```

```
Enter username :
User input
Username is: User input
```

در کد بالا وقتی روی شی input متد ()nextLine فراخوانی میشود ، یک متن را از ورودی میخواند و در متغیر username قرار میدهیم .

برای خواندن انواع داده ای مختلف از متد های مختلف استفاده میکنیم :

| متد | توضیحات |
|---|---|
| nextBoolean() | مقدار هایی از نوع داده‌ای boolean را از کاربر دریافت میکند |
| nextByte() | مقدار هایی از نوع داده‌ای byte را از کاربر دریافت میکند |
| nextDouble() | مقدار هایی از نوع داده‌ای double را از کاربر دریافت میکند |
| nextFloat() | مقدار هایی از نوع داده‌ای float را از کاربر دریافت میکند |
| nextInt() | مقدار هایی از نوع داده‌ای int را از کاربر دریافت میکند |
| nextLine() | مقدار هایی از نوع داده‌ای String را از کاربر دریافت میکند |
| nextLong() | مقدار هایی از نوع داده‌ای long را از کاربر دریافت میکند |
| nextShort() | مقدار هایی از نوع داده‌ای short را از کاربر دریافت میکند |

```java
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    System.out.println("Enter name, age and salary:");

    // String input
    String name = input.nextLine();

    // Numerical input
    int age = input.nextInt();
    double salary = input.nextDouble();

    // Output input by user
    System.out.println("Name: " + name);
    System.out.println("Age: " + age);
    System.out.println("Salary: " + salary);
  }
}
```

```
Enter name, age and salary:
User name input
User age input
User salary input
Name: User name input
Age: User age input
Salary: User salary input
```

## if ,else if,else :

### if :

یک دستور شرطی ساده است با این حالت که یک شرط را به آن میدهیم و در صورتی که ارزش آن شرط برابر با true باشد یک بلوک کد را اجرا میکند .

```
if (condition) {
      // block of code to be executed if the condition is true
    }
```

یک مثال خیلی ساده :

```
if (20 > 18) {
      System.out.println("20 is greater than 18");
    }
```

```
20 is greater than 18
```

همین مثال با استفاده از متغیر به این صورت است .

### else :

از else وقتی استفاده میکنیم که میخواهیم در صورتی که شرط if نادرست باشد (و بلوک کد مربوط به if اجرا نشود ) یک بلوک دیگر کد که به else مربوط است را اجرا کند .

```
if (condition) {
      // block of code to be executed if the condition is true
    }
    else {
      // block of code to be executed if the condition is false
    }
```

مثال از else و if همراه با هم :

```
int time = 20;
      if (time < 18) {
        System.out.println("Good day.");
      }
      else {
        System.out.println("Good evening.");
      }
      // Outputs "Good evening."
```

```
Good evening
```

در این مثال چون time از 18 بزرگ تر است ( 18 < time ارزش نادرستی دارد) پس if اجرا نمیشود و else اجرا میشود و پیغام Good evening را چاپ میکند .

## else if :

همانطور که گفتیم else در صورتی اجرا میشود که if اجرا نشود (شرط if نادرست باشد) .

اکنون میخواهیم دستور else if را برسی کنیم که اگر if اجرا نشد ، قبل از اجرا کردن else شرط دیگری را چک کند و یک دستور را اجرا کند در صورت درست بودن.

مثلا برنامه ای که اگر رتبه ورزشکار یک باشد در خروجی چاپ کند (شما مدال طلا بردید) ، اگر نفر دوم شود در خروجی چاپ کند (شما مدال نقره را بردید) و اگر نفر سوم باشد در خروجی چاپ کند (شما مدال برنز را بردید ) و در صورتی که هیچکدام از این سه حالت نباشد چاپ کند شما ( شما هیچ جایزه ای دریافت نکردید).

```
int position= 3;
if (position == 1) {
  System.out.println("You won the gold medal");
}
else if (position == 2) {
  System.out.println("You won the silver medal");
  }
else if (position == 3) {
  System.out.println("You won the bronze medal");
  }
else {
  System.out.println("You didn't get a medal");
  }
```

```
You won the gold medal
```

در این مثال چون شخص ورزشکار نفر سوم شده است در خروجی این متن را چاپ میکند.

در واقع چون position بازیکن برابر 1 نیست if اجرا نمیشود و همچنین چون برابر 2 هم نیست else if اول (شرط دوم) اجرا نمیشود و چون برابر 3 است و شرط لازم برای اجرای else if دوم است بلوک کد آن را اجرا میکند .

نکته میتوانیم دستوراتی که از یک if و else ساخته شدند را به صورت مختصر تر بنویسیم (Short-hand if...else) :

```
variable = (condition) ? expressionTrue :  expressionFalse;
```

مثال اول else را میتوانیم به این صورت بنویسیم :

```
int time = 20;
String result = (time < 18) ? "Good day." : "Good evening.";
System.out.println(result);
```

```
Good evening
```

گاهی اوقات تعداد شرط هایی که باید برسی کنیم زیاد میشود و نوشتن `if` و `else if` ها میتواند سخت باشد ، در این مواقع از `switch` استفاده میکنیم.

## `switch` :

نحوه کارکرد `switch` به این صورت است که یک مقدار را دریافت میکند و در بدنه کد با هر یک از عبارت های `case` مقایسه میشود و اگر مقداری که `switch` در ابتدا دریافت کرده بود با هر یک از مقادیر موجود در `case` برابر بود ، `case` مربوطه اجرا میشود .

ساختار کلی :

```
switch(expression) {
        case x:
          // code block
          break;
        case y:
          // code block
          break;
        default:
          // code block
}
```

**توجه :** کلید واژه های `break` و `default` اختیاری هستند ، که در ادامه با آن ها آشنا میشوید.

```
      int day = 4;
      switch (day) {
        case 1:
          System.out.println("Saturday");
          break;
        case 2:
          System.out.println("Sunday");
          break;
        case 3:
          System.out.println("Monday");
          break;
```

```
    case 4:
      System.out.println("Tuesday");
      break;
    case 5:
      System.out.println("Wednesday");
      break;
    case 6:
      System.out.println("Thursday");
      break;
    case 7:
      System.out.println("Friday");
      break;
  }
  // Outputs "Tuesday" (day 4)
```

```
Tuesday
```

در این مثال یک متغیر به نام day داریم و با توجه به اینکه مقدار آن چند است با استفاده از switch مشخص میکنیم که آن روز ، روز چندم هفته است .

برای مثال اگر day = 1 باشد خروجی روز شنبه خواهد بود

اگر از break استفاده نکنیم در مثال بالا چه اتفاق می افتد؟ در مثال بعد مشاهده میکنید :

```
  int day = 4;
  switch (day) {
    case 1:
      System.out.println("Saturday");
    case 2:
      System.out.println("Sunday");
    case 3:
      System.out.println("Monday");
    case 4:
      System.out.println("Tuesday");
    case 5:
      System.out.println("Wednesday");
    case 6:
      System.out.println("Thursday");
    case 7:
      System.out.println("Friday");
  }
  // Outputs "Tuesday" (day 4)
```

```
Tuesday
Wednesday
Thursday
Friday
```

دیدید که در صورتی که `break` استفاده نکنیم از آن نقطه که مقدار `case` مربوطه با مقدار ورودی اصلی `switch` همخوانی داشته باشد به بعد ، تمام `case` ها بدون توجه به مقدار `case` اجرا میشوند .

نقش `default` در این کد این است که اگر ورودی `switch` با هیچکدام از `case` ها همخوانی نداشت ، کد های مربوط به `default` اجرا میشوند .

## حلقه ها :

حلقه ها میتوانند یک بلوک کد را تا زمانی که به یک شرط مشخص رسیده‌اند اجرا کنند .

### `while` :

تا زمانی که شرط آن صحیح باشد تمام کد های درون بلوک خود را اجرا میکند .

```
while (condition) {
// code block to be executed
}
```

برای مثال کد زیر تا زمانی که متغیر `i` کوچکتر از `5` باشد حلقه اجرا خواهد شد :

```
int i = 0;
while(i < 5){
    System.out.println(i);
    i++;
   }
```

```
0
1
2
3
4
```

**توجه داشته باشید متغیر های مورد استفاده برای شرط را طوری قرار دهید که حلقه پایان پذیر باشد.**

### `do/while` :

این حلقه بلوک کد را اجرا میکند و سپس شرط را چک میکند و در صورت درست بودن دوباره و دوباره بلوک کد ها را اجرا میکند تا زمانی که شرط نادرست شود .

```
do {
      // code block to be executed
    }
    while (condition);
```

مثال قبل را با استفاده از do/while پیاده میکنیم .

```
int i = 0;
do {
    System.out.println(i);
    i++;
}
while (i < 5);
```

## for :

حلقه for برای زمانی است که دقیقا میدانید قرار است چند بار یک بلوک کد را اجرا کنید .

```
for (statement 1; statement 2; statement 3) {
      // code block to be executed
}
```

statement 1 : دستوری که در اینجا قرار میگیرد فقط یک بار (دقت کنید ، در مجموع فقط یک بار) در مجموع قبل از اجرای بلوک کد ها اجرا میشود ، که ما معمولا در اینجا متغیر شمارنده را تعریف میکنیم (int i = 0) .

statement 2 : این قسمت شرایط اجرای بلوک کد ها را بیان میکنیم .

statement 3 : این قسمت هر بار پس از اجرای بلوک کد ها اجرا میشود (دقت کنید ، هر بار) که ما معمولا برای تغییر دادن متغیر شمارنده از آن استفاده میکنیم (i++) .

```
for (int i = 0; i < 5; i++) {
      System.out.println(i);
}
```

```
0
1
2
3
4
```

این قطعه کد در ابتدا یک متغیر به نام i میسازد و مقدار 0 را به آن اختصاص میدهد .

شرط را چک میکند و اگر درست بود بلوک کدی را اجرا میکند که آن بلوک i را چاپ میکند

در آخر هم یک واحد به i اضافه میکند (i++) .

مثال : مقادیر زوج را از 0 تا 10 چاپ کنید .

```
for (int i = 0; i <= 10; i = i + 2) {
      System.out.println(i);
}
```

## حلقه های تو در تو :

امکان قرار دادن یک حلقه for درون یک حلقه for دیگر وجود دارد ، با مثال این موضوع را برسی میکنیم :

```
int n = 5;
for(int  i = 0; i <= n; i++) {
    for(int  j = 0;  j < i; j++) {
        System.out.print("*");
    }
    System.out.println("");
}
```

```
*
**
***
****
*****
```

For-Each : (بهتر است این قسمت را بعد از یاد گرفتن آرایه مطالعه کنید)

```
int[] number = {10 ,20 ,30 ,40 ,50};
for(int element : number){
    System.out.println(element);
}
```

```
10
20
30
40
50
```

در این نوع حلقه for نیازی به تعریف متغیر برای شمارنده نیست و خود حلقه اعضا را پیمایش میکند و بلوک کد را اجرا میکند ،

## break/continue :

### break :

همانطور که در switch از آن استفاده کردیم و از نامش هم مشخص است برای پرش از حلقه از آن استفاده میکنیم .

```
for (int i = 0; i < 10; i++) {
      if (i == 4) {
        break;
      }
      System.out.println(i);
    }
```

```
0
1
2
3
```

break را میتوانیم در حلقه while هم استفاده کنیم :

```
    int i = 0;
while (i < 10) {
  System.out.println(i);
  i++;
  if (i == 4) {
    break;
      }
    }
```

```
0
1
2
3
4
```

در این مثال انتظار میرود که اعداد یک تا 9 چاپ شوند ، اما یک دستور شرطی if قرار داده ایم که در هر مرحله که بلوک کد اجرا میشود ، اگر i برابر با 4 باشد بلوک کد مربوط به if را اجرا میکند ، که درون آن بلوک کد دستور break قرار دارد و این دستور باعث میشود از حلقه for به بیرون از حلقه پرش کند و حلقه را در همان مرحله پایان دهد .

`continue` : این دستور باعث میشود که فقط از آن مرحله پرش کنیم و به مرحله بعدی حلقه برویم .

برای مثال این کد از 4 پرش میکند و آن را اجرا نمیکند .

```
for(int i = 0; i < 10; i++) {
    if(i == 4){
        continue;
    }
    System.out.println(i);
    }
```

```
0
1
2
3
5
6
7
8
9
```

`continue` را میتوانیم در حلقه `while` هم استفاده کنیم :

```
int i = 0;
while(i < 10) {
    if(i == 4) {
        i++;
        continue;
    }
     System.out.println(i);
     i++;
    }
```

```
0
1
2
3
5
6
7
8
9
```

تا اینجا با دو دستور `print` و `println` آشنا شدیم و از آنها بار ها استفاده کردیم ، اکنون برای قسمت های نیاز است با `printf` هم آشنا شوید .

از متد ()`printf` می توان برای قالب بندی رشته ها و اعداد استفاده کرد.

این متد بر اساس یک **الگو** و با استفاده از کاراکترهای خاصی که در جدول زیر آمده اند، قالب بندی را انجام می دهد.

الگوی کلی قالب بندی رشته ها و اعداد به صورت زیر است :

```
%[flags] [width] [.precision] conversion-character
```

در الگوی بالا اجزایی که در داخل کروشه هستند اختیاری می باشند. در حالت عادی الگو با علامت % شروع می شود و بعد از آن یکی از کاراکترهای جدول زیر می آید:

| عملکرد | کاراکترconversion-character |
|---|---|
| **چاپ کاراکتر** | %c |
| **چاپ اعداد double** | %d |
| **چاپ اعداد float** | %f |
| **چاپ رشته** | %s |
| **چاپ کاراکتر %** | %% |
| **چاپ یک خط** | %n |

مثال :

```
System.out.printf("%f",34.789456);
```

```
34.789456
```

حال نحوه قرارگیری اجزای درون کروشه ها را برسی میکنیم .

ابتدا flags :

| کاربرد | نشانه |
|---|---|
| با اضافه کردن فاصله به سمت راست یک عدد یا رشته آن را به سمت چپ می کشد. | - |
| با اضافه کردن فاصله به سمت چپ یک عدد یا رشته آن را به سمت راست می کشد. | + |
| تعدادی صفر که خودمان تعیین کرده ایم به سمت راست یا چپ نوشته یا عدد اضافه می کند | 0 |
| تعدادی فاصله که خودمان تعیین کرده ایم به سمت راست یا چپ نوشته یا عدد اضافه می کند | فاصله |

به کار بردن نشانه های بالا به تنهایی و بدون اینکه تعیین کنیم چه تعداد فاصله یا صفر می خواهیم به ابتدا یا انتهای عدد یا رشته اضافه کنیم بی معنی می باشد.

در این صورت باید از جزء بعدی که width یا پهنا هست استفاده کنیم [.precision] .هم در صورتی که متغیر از نوع اعداد اعشاری باشد، برای تعیین تعداد ارقام اعشار، و اگر از نوع رشته باشد تعداد کارکترهایی که قرار است نمایش داده شوند را مشخص می کند .

حال به مثال بر می گردیم. فرض کنید که می خواهیم سه رقم از ارقام بعد از ممیز عدد اعشار مثال بالا را نشان داده و قبل از بخش صحیح آن سه عدد صفر قرار دهیم یعنی : 00034.789

```
System.out.printf("%09.3f",34.789456);
```

```
00034.789
```

همانطور که در کد بالا مشاهده می کنید الگو با علامت % شروع می شود. سپس نشانه را می نویسیم که در اینجا 0 است.

اما اینکه چرا عدد 9 را نوشته ایم دلیلش این است که 34.789 با احتساب ممیز آن برای نمایش نیاز به شش جای خالی دارد و چون قرار است که ما سه صفر هم قبل از عدد 34 قرار دهیم پس باید 9 جای خالی ایجاد کنیم.

و اما [ 3. ] هم به معنای سه رقم اعشار است و f هم که برای نمایش اعداد اعشاری به کار می رود.

فرض کنید که می خواهیم سه کاراکتر اول رشته Programming را نمایش دهیم، برای این منظور به صورت زیر عمل می کنیم :

```
System.out.printf("%.3s","Programming");
```

```
Pro
```

در زیر هم مثالهایی از نحوه استفاده از متد printf و کاراکترهای خاص آن آمده است :

## چاپ یک رشته ساده

```
System.out.printf("Hello Welcome to JAVA Programming");
```

```
Hello Welcome to JAVA Programming
```

## چاپ متغیرها

```
double sum = 2+3;
System.out.printf("Addition of two Numbers : %d",sum);
```

```
Addition of two Numbers : 5
```

### قالب بندی اعداد اعشاری

```
System.out.printf("%-12s%-12s\n","Column 1","Column 2");
System.out.printf("%-12.5f%.20f", 12.23429837482,9.10212023134);
```

```
Column 1    Column 2
12.23430    9.10212023134000000000
```

درباره کد بالا یک نکته را یادآور می شویم که اگر تعداد کاراکترهای یک رشته از تعدادی که ما برای قالب بندی آن استفاده کرده ایم کمتر باشد ، تعدادی فاصله در سمت چپ یا راست رشته قرار می گیرد.

مثلا تعداد کاراکترهای Column 1 هشت عدد می باشد و ما برای قالب بندی و ایجاد فاصله در سمت راست آن عدد 12- را به کار برده ایم.

با این کار 4 فاصله در سمت راست رشته قرار می گیرد و باعث فاصله آن با رشته بعدی می شود.

حال اگر تعداد کارکترهای رشته بیشتر از تعداد باشد که ما تعیین کرده ایم، کل رشته یا عدد بدون هیچ گونه فاصله ای در سمت چپ یا راست، نمایش داده می شود. همین نکته در مورد مثال های زیر صدق می کند .

## نمایش اعداد صحیح در قالب خاص

```
System.out.printf("%d",1234);
System.out.printf("%6d",1234);
System.out.printf("%-6d",1234);
System.out.printf("%06d",1234);
```

```
1234
  1234
1234
001234
```

## نمایش رشته ها در قالب خاص

```
String str="Programming";
System.out.printf("%s",str);
System.out.printf("%10s",str);
System.out.printf("%15s",str);
System.out.printf("%-15s",str);
System.out.printf("%15.5s",str);
System.out.printf("%-15.5s",str);
```

```
Programming
Programming
  Programming
Programming
      Progr
Progr
```

برای روشن شدن کاربرد اعداد منفی و مثبت بعد از علامت % فرض کنید که شما می خواهید بین دو رشته JAVA و Programming چهار فاصله قرار دهید. این کار به دو صورت امکان پذیر است. یا بعد از کلمه JAVA چهار فاصله قرار دهید :

```
System.out.printf("%-8s%s", "JAVA","Programming");
```

```
JAVA    Programming
```

که در مثال بالا 8- به معنای این است که 8 مکان ایجاد شود که چهار تا از آنها توسط کلمه JAVA اشغال می شود و چهار تای دیگر به خاطر علامت منفی در سمت راست کلمه JAVA قرار می گیرند.

حالت دوم هم این است که در سمت چپ کلمه Programming چهار فاصله قرار بدهیم :

```
System.out.printf("%s%15s", "JAVA","Programming");
```

```
JAVA    Programming
```

که در این صورت باید عدد 15 را بنویسیم. چونکه کلمه Programming یازده حرفی است، پس چهار مکان دیگر به خاطر مثبت بودن علامت 15 در سمت چپ آن قرار می گیرند.

## آرایه ها (Arrays):

مفهوم آرایه در زبان برنامه‌نویسی یکی از ساختار های پایه داده ای است که برای ذخیره سازی مجموعه از داده ها با نوع داده ای یکسان است . آرایه ها به ما این امکان را میدهند که داده های مرتبط را به طور منظم و کارامد در کنار یکدیگر ذخیره کنیم .

برای تعریف کردن آرایه باید به صورت زیر عمل کنیم :

```
type[] name = new type[size];
```

- `type` : نوع داده ای که میخواهیم درون آرایه ذخیره کرد
- `name` : نام آرایه
- `new` : کلمه کلیدی (در قسمت کلاس ها به آن پرداخته می‌شود )
- `size` : تعداد اعضای آرایه (عدد صحیح)

برای مثال در آرایه زیر 5 عدد با نوع داده ای `int` درون آرایه ذخیره میشوند :

```
int[] Numbers = new int[5];
Numbers = {2,4,8,6,4};
```

این کد ابتدا آرایه را تعریف کردیم و سپس مقدار دهی کردیم ، اما میتوان این کار را در یک خط انجام دهیم و به صورت مستقیم مقدار دهی کنیم.

```
 int[] Numbers = {2,4,8,6,4};
```

## دسترسی به اعضای آرایه :

هر عنصر در آرایه در یک محل حافظه قرار دارد که توسط اعداد 0 تا n شماره گذاری میشوند و آن ها را میشناسند .

برای مثال اولین عنصر درون آرایه Numbers را این‌گونه میشناسند و میتوان آن را چاپ کرد :

```
System.out.println(Numbers[0]);
```

```
2
```

در این مثال به سادگی index شماره 0 آرایه را که برابر با 2 است را چاپ کردیم

میتوانیم با استفاده از حلقه for تمامی اعضای این آرایه را چاپ کنیم :

```
int[] Numbers = {2, 4, 8, 6, 4};
for(int i = 0; i < 5; i++){
  System.out.println("Numbers["+ i +"] is: " + Numbers[i]);
}
```

```
Numbers[0] is: 2
Numbers[1] is: 4
Numbers[2] is: 8
Numbers[3] is: 6
Numbers[4] is: 4
```

گاهی ممکن است تعداد عنصر های یک آرایه را ندانیم و برای نوشتن حلقه for به آن نیاز داشته باشیم ، در این مواقع میتوانیم متد length() را روی آرایه فراخوانی کنیم و تعداد عناصر آرایه را به ما برمیگرداند .
فرض کنید در مثال بالا نمیدانستیم که آرایه دارای 5 عنصر است در این صورت سطر اول حلقه را به این صورت مینوشتیم :

```
int[] Numbers = {2, 4, 8, 6, 4};
for(int i = 0; i < Numbers.length; i++){
    System.out.println("Numbers[" + i + "] is: " + Numbers[i]);
}
```

```
Numbers[0] is: 2
Numbers[1] is: 4
Numbers[2] is: 8
Numbers[3] is: 6
Numbers[4] is: 4
```

این کار با استفاده از For-Each راحت تر انجام میشود :

```java
int[] Numbers = {2,4,8,6,4};
int index = 0;
for(int i : Numbers){
    System.out.println("Numbers["+ index +"] is: " + i );
    index++;
}
```

```
Numbers[0] is: 2
Numbers[1] is: 4
Numbers[2] is: 8
Numbers[3] is: 6
Numbers[4] is: 4
```

## آرایه های چند بعدی :

ارایه های چند بعدی شبیه به همان ماتریس ها هستند ، ما میتوانیم در یک ماتریس 5 سطر داشته باشیم که در هر سطر 3 ستون وجود داشته باشد ، شبیه به آن در جاوا ما یک ارایه دو بعدی 5 در 3 داریم .

برای ارائه یک مثال ساده یک ارایه را یک ماتریس 4 در 2 تعریف میکنیم.

ابتدا اعلان و تعریف سایز آرایه:

```java
int[][] matrix = new int[4][2];
```

سپس مقدار دهی آرایه :

```java
matrix[0][0] = 38;
matrix[0][1] = 67;
matrix[1][0] = 19;
matrix[1][1] = 17;
matrix[2][0] = 43;
matrix[2][1] = 22;
matrix[3][0] = 16;
matrix[3][1] = 37;
```

با استفاده از حلقه for راحت تر میتوانستیم آن را مقدار دهی کنیم ، برای این کار باید ورودی را از کاربر دریافت کنیم که با استفاده از کلاس Scanner این کار را انجام میدهیم .

```java
Scanner input = new Scanner(System.in);
int[][] matrix = new int[4][2];
```

```
for(int i=0;i<4;i++){
    for(int j=0;j<2;j++){
        System.out.print("matrix["+i+"]["+j+"]:");
        matrix[i][j] = input.nextInt();
    }
}
```

```
matrix[0][0]:38
matrix[0][1]:67
matrix[1][0]:19
matrix[1][1]:17
matrix[2][0]:43
matrix[2][1]:22
matrix[3][0]:16
matrix[3][1]:37
```

میتوانیم سپس ماتریس را با استفاده از `printf` و For-Each چاپ کنیم در زیر این مثال از ابتدا و به صورت کامل درج شده :

```
Scanner input = new Scanner(System.in);
int[][] matrix = new int[4][2];
for(int i = 0; i < 4; i++) {
    for(int j = 0; j < 2; j++) {
        System.out.print("matrix[" + i + "][" + j + "]:");
        matrix[i][j] = input.nextInt();
     }
}
//This code is written to print a matrix
for(int[] i : matrix) {
    for(int j : i){
        System.out.printf("%3d",j);
    }
    System.out.println();
}
```

شاید با نحوه نگارش For-Each در این قسمت مشکل داشته باشید ، بیایید با هم آن را برسی کنیم ،

```
for (int[] i : matrix){…}
```

در این قسمت ، نوع داده‌ای که درون matrix است int[] میباشد (در واقع ما درون آرایه matrix چند آرایه دیگر داریم . پس ، نوع داده‌ای آن عناصر int[] میباشد چون هر یک از آنها یک آرایه هستند ) و هر یک از آن عناصر درون این حلقه تحت عنوان i شناخته میشوند .

```
for (int j : i){…}
```

در این قسمت ما عمل پیمایش را بر روی هر یک از i ها (**آرایه های درونی** ) انجام میدهیم،
نوع داده‌ای که درون **آرایه های درونی** ذخیره میشود int است .

و هر یک از عناصر درون **آرایه های درونی** را به عنوان j میشناسیم .

و اکنون میتوانیم j ها را چاپ کنیم ؛ که با استفاده از دستور printf آنها را به همراه 3 فضای خالی برای جلوگیری از به هم ریختن اعداد چاپ میکنیم .

خودتان را بیازمایید و با استفاده از حلقه های for معمولی سعی کنید آن ها را چاپ کنید و سپس پاسخ را در زیر مقایسه کنید .

```
for(int i = 0; i < matrix.length; i++){
    for(int j = 0; j < matrix[i].length; j++){
        System.out.printf("%3d", matrix[i][j]);
    }
    System.out.println();
}
```

```
2 1
4 3
6 5
8 7
```

حال بیاید با چند مثال مطالبی که تا کنون آموخته‌اید را استفاده کنید .

در این مثال قرار است معدل دانش آموزانی از یک مدرسه ابتدایی که تعداد دانش آموزان هر کلاس به صورت زیر است گرفته شود ، سپس معدل هر کلاس محاسبه شود و بالاترین نمایش داده شود و همچنین معدل کل دانش آموزان مدرسه نیز نمایش داده شود.

```
import java.util.Arrays;
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int[][] school = new int[6][];
        school[0] = new int[21];
        school[1] = new int[24];
        school[2] = new int[17];
        school[3] = new int[15];
        school[4] = new int[19];
        school[5] = new int[26];

        for (int i = 0; i < school.length; i++){
            System.out.println("class " + i + " :");
            for (int j = 0; school[i].length > j; j++){
                System.out.print("\tST " + j + " :");
                school[i][j] = input.nextInt();
            }
        }

        float[] moadel = new float[6];

        for(int i=0;i < school.length;i++){
            for (int j=0; j < school[i].length; j++){
                switch (i) { //using enhanced switch statement (auto break)
```

```
                    case (0) -> moadel[0] += school[i][j];
                    case (1) -> moadel[1] += school[i][j];
                    case (2) -> moadel[2] += school[i][j];
                    case (3) -> moadel[3] += school[i][j];
                    case (4) -> moadel[4] += school[i][j];
                    case (5) -> moadel[5] += school[i][j];
                }
                if (j + 1 == school[i].length){
                    moadel[i] = moadel[i] / school[i].length;
                }
            }
        }
        System.out.println("bala tarin mooadel :  " + Math.max(moadel[0],
Math.max(moadel[1], Math.max(moadel[2], Math.max(moadel[3], Math.max(moadel[4],
moadel[5])))));
          /*  توضیح فارسی :عضو بزرگ تر از بین عضو 5 ام و 6 ام انتخاب و با عضو 4 ام مقایسه میشود ، سپس از بین این دو
                                                                عضو بزرگ تر انتخاب و با عضو سوم مقایسه میشود
                                                  .... و سپس از بین عضو سوم و دیگری عضو بزرگ تر انتخاب و با عضو دوم مقایسه میشود و
                                                                تا جایی که بزرگ ترین پیدا شود
                                                                                                  */
        System.out.println((moadel[0] + moadel[1] + moadel[2] + moadel[3] +
moadel[4] + moadel[5]) / 6);
    }
}
```

## خطا های برنامه نویسی:

1. **خطا های نحوی یا خطا های کامپایل (syntax error):** خطا در قوانین گرامر جاوا
   - قرار دادن `import` در داخل یا بعد از کلاس
   - فراموش کردن `import`
   - تغییر مقدار متغیر `final`
   - عدم هم خوانی نام یک کلاس و نام فایل
   - حساس به املای صحیح کلمه
   - فراموش کردن نقطه ویرگول (;)
   - فراموش کردن کلمه کلیدی
   - فراموش کردن نقل قول پایانی یک رشته (")
   - فراموش کردن بستن توضیح یک چند خطی

2. **خطا های منطقی(logical error):** یک کد نوشته میشود و وظیفه‌ی مورد نظرش را انجام ندهد.

3. **خطا های زمان اجرا(runtime error):**خطا های زمان اجرا وقتی رخ میدهد که برنامه سعی میکند عمل نامعتبری را انجام دهد (وقوع یک خطای «زمان اجرا» منجر به خاتمه پیدا کردن برنامه میشود) مانند:

- تقسیم یک عدد بر صفر
- خواندن داده از فایلی که وجود ندارد
- نوشتن یک متد استاتیک که خود را فراخوانی میکند میتواند منجر به خطای زمان اجرا شود

## متد ها :

متد ها یا همون توابع بلوکی از کد ها هستن که یک عملیاتی رو انجام میدن ، البته وقتی که فراخوانی شوند .

ما میتونیم یک سری اطلاعات که به عنوان پارامتر میشناسیمشون رو برای متد ها بفرستیم . متد ها از اون ها استفاده میکنن و یک عملی رو انجام میدن .

چرا از متد ها استفاده میکنیم ؟ برای اینکه یک الگوریتم رو یک بار طراحی کنیم و چندین بار از اون ها استفاده کنیم .

خیلی سریع بریم سراغ نحوه ساخت یک متد :

متد ها باید درون کلاس تعریف شود ، و برای تعریف متد ها ابتدا نام متد و سپس () قرار میدهیم . (جاوا دارای متد های پیش ساخته زیادی هست ؛ مثل println() )

```
public class Main {

    public void myMethod() {
      // code to be executed
    }
  }
```

### توضیح کد بالا :

- myMethod() نام متد است .
- static کلمه کلیدی در نظر بگیرید ( بعد ها درباره آن صحبت خواهد شد)
- void به این معنی است که متد چیزی را برنمیگرداند (هر متدی که از نوع void باشد به همین معنی است)

دقت کنید که ما باید متد هایمان را قبل از متد اصلی برنامه بنویسیم در غیر این صورت برنامه با خطا مواجه میشود .

(یا میتوانیم متد هایمان پس از متد main بنویسیم ولی باید به حالت static بنویسیم آن را که در فصل های بعد آن را بررسی میکنیم)

## نحوه فراخوانی متد ها :

برای فراخوانی متد ها کافیست نام متد را به همراه () بنویسیم .

برای مثال :

```
public class Main {
    public void myMethod() {
      System.out.println("I just got executed!");
    }
    public static void main(String[] args) {
      myMethod();
    }
  }
```

```
I just got executed!
```

یک متد میتواند چندین بار فراخوانی شود .

```
public class Main {

  public void myMethod() {
    System.out.println("I just got executed!");
  }

  public static void main(String[] args) {
    myMethod();
    myMethod();
    myMethod();
  }
}
```

```
I just got executed!
I just got executed!
I just got executed!
```

## پارامتر ها و آرگومان ها :

ما اطلاعات را به عنوان پارامتر ها به متد ارسال میکنیم و متد به چشم یک متغیر به آنها می نگرد .
پارامتر ها به همراه نوع داده‌ای‌شان درون پرانتزی که بعد از نام متد قرار میدهیم جای میگیردند . ما میتوانیم هر تعداد که نیاز داشته باشیم به متد ها پارامتر ارسال کنیم ، فقط باید با استفاده از « , » آنها را از هم جدا کنیم .
مثال زیر یک متد داریم که یک نام را برای آن ارسال میکنیم و به او سلام میکند .

```
public class Main {

  public void myMethod(String name) {
    System.out.println("Hello " + name + "!");
  }

  public static void main(String[] args) {
    myMethod("Ali");
```

```
    myMethod("Mohamad");
    myMethod("Narges");
  }
}
```

```
Hello Ali!
Hello Mohamad!
Hello Narges!
```

**توجه :** وقتی یک پارامتر را به متد ارسال میکنیم ، آن وقت به آن آرگومان میگوییم .

مثال :
شما میتوانید بیشتر از یک پارامتر را به متد ارسال کنید .

```
public class Main {
  public static void myMethod(String name, int age) {
      System.out.println(name + " is " + age + "years old.");
  }

  public static void main(String[] args) {
      myMethod("Setayesh", 5);
      myMethod("Sam", 8);
      myMethod("Reza", 31);
  }
}
```

```
Setayesh is 5years old.
Sam is 8years old.
Reza is 31years old.
```

**توجه :** وقتی یک تابع تعدادی پارامتر دارد ، در هنگام فراخوانی باید به همان تعداد برایش آرگومان ارسال کنیم .

متد ها به همراه if...else :
معمولا درون متد ها از if...else زیاد استفاده میکنیم
به مثال زیر توجه کنید

```
public class Main {

    public static void checkMoney(double moneyCount) {

        if (moneyCount < 100.0) {
            System.out.println("Insufficient funds - You need more money!");
```

```
        }
        else {
            System.out.println("Sufficient funds - You have enough money!");
        }
    }
    public static void main(String[] args) {
        checkMoney(250.0);
        checkMoney(50.0);
    }
}
```

```
Sufficient funds - You have enough money!
Insufficient funds - You need more money!
```

## return :

در مثال هایی که از متد ها تا کنون برسی کردیم ، متد ها خروجی نداشتند ( از نوع void ) بودند .
اکنون میخواهیم خروجی متد ها را برسی کنیم

متد ها مقدار خروجی خود را در صورت وجود به محلی که فراخوانی شده‌اند برگشت میدهند .
برای اینکه یک متد خروجی دار تعریف کنیم باید به جای void نوع داده ای که قرار است متد برگرداند را قرار دهیم ( مثل int ، double ) و سپس در انتهای متد (قبل از بسته شدن بلوک کد مربوط به متد ) با استفاده از کلمه کلیدی return مقدار بازگشتی متد را تعریف کنیم . به مثال زیر توجه کنید
یک متد که دو ورودی را دریافت میکند و حاصل را برمیگرداند .

```
public class Main {
  public static int myMethod(int x, int y) {
      return x + y;
  }

  public static void main(String[] args) {
      System.out.println(myMethod(3, 5));
  }
}
```

```
8
```

ما میتوانیم خروجی متد ها را در متغیر ها ( با نوع داده مناسب ) ذخیره کنیم و از آن استفاده کنیم .

مثال قبل را در یک متغیر به نام z ذخیره میکنیم و سپس متغیر را چاپ میکنیم .

```
public class Main {
  public static int myMethod(int x, int y) {
      return x + y;
  }
```

```
  public static void main(String[] args) {
      int z = myMethod(5, 3);
      System.out.println(z);
  }
}
```

```
8
```

## Method Overloading (سربارگیری متد ها) :

شاید بتوان گفت مهم ترین قسمتی که از متد ها شما باید بیاموزید این قسمت است .
فرض کنید کاربر در مثال قبلی که متد ما دو ورودی از نوع `int` میگرفت و سپس حاصل جمع را برمیگرداند نوع داده ای `double` را وارد کند (به جای دو عدد صحیح دو عدد اعشاری را وارد کند )

```
public class Main {
  public static int myMethod(int x) {
      return x;
  }
  public static void main(String[] args) {
    System.out.println(myMethod(3.3));
  }
}
```

```
Main.java:6: error: incompatible types: possible lossy conversion from double
to int
    System.out.println(myMethod(3.3));
                                ^
Note: Some messages have been simplified; recompile with -Xdiags:verbose to
get full output
1 error
```

برای حل کردن این چنین مشکلی ما از Method Overloading استفاده میکنیم . به این صورت که همان متد را دوباره تعریف میکنیم ولی این بار با نوع ورودی متفاوت (برای این مثال `double`)

```
public class Main {
  public static int myMethod(int x) {
      return x;
  }
  public static double myMethod(double x) {
      return x;
  }
  public static void main(String[] args) {
      System.out.println(myMethod(3.3));
```

```
  }
}
```

```
3.3
```

در این مثال که ملاحظه کردید ، جاوا با توجه به ورودی های متد ها تشخیص میدهد و آن آرگومان ها را به متد متناسب ارسال میکند .

**توجه :** ما فقط در صورتی میتوانیم دو متد هم نام داشته باشیم که امضای متد ها متفاوت باشد ( نوع ورودی متفاوتی داشته باشند.)

## محدوده (scope):

معادل فارسی اسکوپ محدوده هست که مشخص می کنه هر متغیر در این زبان در چه محدوده ای قابل استفادست.
یعنی نمیشه همه جای محیط برنامه به یه سری متغیر ها دسترسی پیدا کرد.
برنامه های جاوا در قالب کلاس ها سازماندهی می شوند و هر کلاس بخشی از یک بسته است. قوانین حوزه جاوا را می توان تحت دسته های زیر پوشش داد:

### متغیرهای کلاس(class level scope):

این متغیرها باید در داخل کلاس (خارج از هر تابع) اعلان شوند. آنها را می توان به طور مستقیم در هر نقطه در کلاس دسترسی داشت :

```
public class Test
{
    // All variables defined directly inside a class
    // are member variables
    int a;
    private String b;
    void method1() {....}
    int method2() {....}
    char c;
}
```

- ما میتوانیم متغیر های کلاس را در هر جایی از کلاس اعلام کنیم اما باید خارج از متد ها باشد.
- Access specified از متغیرهای کلاس بر دامنه آنها در کلاس تأثیر نمی گذارد.
- به متغیرهای کلاس می توان در خارج از کلاس با قوانین زیر دسترسی داشت:

| Modifier | بسته | زیرکلاس | خارج از بسته |
|---|---|---|---|
| public | yes | yes | yes |
| protected | yes | yes | no |
| Default(no Modifier) | yes | no | no |
| private | no | no | no |

### متغیر های محلی(method level scope):

زمانی که یک متغیر درون یک متد تعریف شود، دارای دامنه متد است و تنها درون همان متد معتبر خواهد بود:

```
public class MethodScopeExample {
    public void methodA() {
        Integer area = 2;
    }
    public void methodB() {
        // compiler error, area cannot be resolved to a variable
        area = area + 2;
    }
}
```

```
        area = area + 2;
        ^
  symbol:   variable area
  location: class Main
Main.java:7: error: cannot find symbol
        area = area + 2;
               ^
  symbol:   variable area
2 errors
```

در کد فوق در methodA یک متغیر متد به نام area ایجاد کردیم. به همین جهت، می‌توانیم از area درون methodA استفاده کنیم، اما امکان استفاده از این متغیر در هر جایی خارج از این متد وجود ندارد برای همین کامپایلر در methodB ارور میدهد.

### متغیر های حلقه(block scope):

اگر متغیری را درون یک حلقه اعلان کنیم، دارای دامنه حلقه بوده و تنها درون همان حلقه در دسترس ما خواهد بود:

```java
public class LoopScopeExample {
    String[] listOfNames = {"Joe", "Susan", "Pattrick"};
    public void iterationOfNames() {
        String allNames = "";
        for (String name : listOfNames) {
            allNames = allNames + " " + name;
        }
        // compiler error, name cannot be resolved to a variable
        String lastNameUsed = name;
    }
}
```

```
Main.java:9: error: cannot find symbol
        String lastNameUsed = name;
                              ^
  symbol:   variable name
1 error
```

همانطور که می‌بینیم در کد فوق متد `iterationOfNames` دارای یک متغیر متد به نام `name` است. این متغیر تنها می‌تواند درون حلقه مورد استفاده قرار گیرد و خارج از آن معتبر نیست.

## متدهای بازگشتی :

ما میتوانیم در متد بازگشتی به جای اینکه یک نوع داده را برگردانیم ، خود متد را در یک حالت دیگر فراخوانی کنیم .
شاید درک این مساله دشوار باشد ، بهتر است خودتان آن را امتحان کنید .

برای مثال من میخواهم متدی بنویسم که در صورت فراخوانی مجموع تمام اعداد صحیح کوچکتر از عددی که به متد داده شده را برگرداند

به این صورت متد را مینویسیم :

```java
public class Main {

  public static int sum(int k) {
      if(k > 1){
          return k + sum(k - 1);
      }
      return k;
  }
  public static void main(String[] args) {
      System.out.println(sum(10));
  }
}
```

```
55
```

حال یک متد بنویسیم که فاکتوریل را محاسبه کند :

```java
public class Main{
  public static int fact(int x){
      if(x > 1){
          return x * fact(x-1);
      }
      else{
          return x;
      }
  }
  public static void main(String[] args){
      System.out.println(fact(5));
      System.out.println(fact(4));
      System.out.println(fact(3));
  }
}
```

```
120
24
6
```

این متد دنباله فیبوناچی را تا عددی که کاربر وارد میکند چاپ میکند :

```java
import java.util.Scanner;
public class Main {
    public static long fibo(int a, int b,int n){
        if (n>0){
            System.out.println(a);
        return fibo(b,a+b,n-1);
        }
        return 0;
    }
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter n :");
        fibo(0,1,input.nextInt());
    }
}
```

```
Enter n : 9 example
0
1
1
```

```
2
3
5
8
13
21
```

**توجه :** همانطور که در حلقه های `while` در دام حلقه های پایان ناپذیر می افتادیم در متدهای بازگشتی هم این اتفاق می افتد و باید با ساختار های کنترلی از این اتفاق جلوگیری کنیم .

مفاهیم اولیه جاوا به پایان رسید و اکنون وقت آن است نحوه استفاده از آن را بیاموزیم ، برای اینکار مثال های پیچیده تری را برای شما انتخاب کرده ایم که با استفاده از آنها میتوانید آموخته های خود را تثبیت کنید .

**مثال اول :** یک متد که آرایه ای با نوع داده ای `int` دریافت کند و مقدار بزرگ ترین عضو آرایه را برگرداند :

```java
public class Main {

  public static int maxNum(int[] array){
      int max = array[0];
      for (int i = 0; i < ((array.length) - 1); i++){
          if (max < array[i+1]){
              max = array[i+1];
          }
      }
      return max;
  }

  public static void main(String[] args) {
      int[] myArray = {1,2,3,5,2,3,23,42,2,1,3,5,7};
      System.out.println(maxNum(myArray));
  }
}
```

```
42
```

این متد را چگونه میتوانیم بهبود ببخشیم ؟
به این فکر کنید که آرایه ما از نوع عددی باشد ولی صحیح نباشد ! آن وقت متد ما نمیتواند با این آرایه فراخوانی شود .
اکنون با استفاده از مفهوم سربارگیری متد ها یا method overloading این مشکل را برطرف میکنیم.

```java
public class Main {
  public static int maxNum(int[] array){
      int max = array[0];
      for (int i = 0; i < ((array.length) - 1); i++){
```

```
            if (max < array[i+1]){
                max = array[i+1];
            }
        }
        return max;
    }

    public static double maxNum(double[] array){
        double max = array[0];
        for (int i = 0; i < ((array.length) - 1); i++){
            if (max < array[i+1]){
                max = array[i+1];
            }
        }
        return max;
    }

    public static void main(String[] args) {
        double[] myArray = {1.65, 2.5, 3.3, 5.53, 2.4, 34.4, 23.3, 42.343,
2.43, 17.9, 3.0, 5.23, 7.3};
        System.out.println(maxNum(myArray));

    }
}
```

```
42.343
```

**مثال دوم:** یک متد بنویسیم که ضرایب یک چند جمله ای درجه 2 را دریافت کند و در صورت وجود ریشه های آن را نمایش دهد.

```
public class Main {
    public static void findRoots(double a, double b, double c) {
        double determinant = b * b - 4 * a * c;
        double root1, root2;
        if (determinant >= 0) {
            root1 = (-b + Math.sqrt(determinant)) / (2 * a);
            root2 = (-b - Math.sqrt(determinant)) / (2 * a);
            if(root2 != root1){
                System.out.printf("Roots are real and different: %.2f and
%.2f%n\n", root1, root2);
            }
            else {
                System.out.printf("Roots are real and same: %.2f\n", root1);
            }
        } else {
            System.out.println("Roots are not real");
        }
```

```
  }
  public static void main(String[] args){
      findRoots(1,1,2);
      findRoots(1,-4,4);
      findRoots(1,-7,12);
  }
}
```

```
Roots are not real
Roots are real and same: 2.00
Roots are real and different: 4.00 and 3.00
```

**مثال سوم**: یک متد برای چسباندن دو آرایه به یکدیگر:

```
public class Main{
  public static int[] MergeTwoArrays(int[] array_1, int[] array_2){
      int len1 = array_1.length;
      int len2 = array_2.length;
      int len3 = len1 + len2;
      int[] result = new int[len3];
      int j = 0;
      for(int i = 0 ; i < len1; i++){
          result[j] = array_1[i];
          j++;
      }
      for(int i = 0; i < len2; i++){
          result [j] = array _2[i];
          j++;
      }
      return result;
  }
  public static void main(String[] args){
      int[] a ={1, 2, 3};
      int[] b ={4, 5, 6, 7};
      int[] c=new int[7];
      c = MergeTwoArrays(a, b);
      for(int i = 0; i < 7; i++){
          System.out.println(c[i]);
      }
  }
}
```

```
1
2
3
4
5
```

```
6
7
```

**مثال چهارم**: یک متد برای ساخت رمز عبور قوی چند رقمی :

```java
import java.util.Random;

public class Main {

    public static String generatePassword(int length) {
        String chars =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*()";
        Random rnd = new Random();
        char[] password = new char[length];
        for (int i = 0; i < length; i++) {
            password[i] = chars.charAt(rnd.nextInt(chars.length()));
        }
        return new String(password);
    }

    public static void main(String[] args) {
        System.out.println(generatePassword(10));  // example usage
        System.out.println(generatePassword(10));  // example usage
        System.out.println(generatePassword(10));  // example usage
    }
}
```

```
t%Z9EP%Bv(
FxeXJsReXm
F2e$Q$PuZM
```

**مثال پنجم** : یک متد بنویسید که یک عدد مثل n را دریافت کند و یک آرایه n در n با اعداد تصادفی یک رقمی را چاپ کند:

```java
import java.util.Random;
import java.util.Scanner;
public class testclass{
    public static int[][] rand(int n){
        int[][] array = new int[n][n];
        Random rand=new Random();
        for(int i = 0; i < n; i++){
            for(int j=0 ; j<n ; j++){
                array[i][j] = rand.nextInt(10);
            }
```

```
        }
        return array;
    }
    public static void main(String[] args){
        Scanner input = new Scanner(System.in);
        System.out.println("Enter number like n for print a n*n matrix: ");
        int n = input.nextInt();
        int[][] matrix = new int[n][n];
        matrix = rand(n);
        for(int i=0 ; i<n ; i++){
            for(int j=0 ; j<n ; j++){
                System.out.print(matrix[i][j] + " ");
            }
            System.out.println("");
        }
    }
}
```

```
Enter number like n for print a n*n matrix:
3
2 9 7
4 2 0
0 1 3
```

**مثال ششم** : متدی که یک ماتریس n در m را میسازد و اعداد را از کاربر دریافت میکند:

```
import java.util.Scanner;
public class testclass{
    public static int[][] Matrix(int n, int m){
        int[][] array = new int[n][m];
        Scanner input = new Scanner(System.in);
        for(int i = 0; i < n; i++){
            for(int j = 0; j < m; j++){
                System.out.printf("array[%d][%d] = ", i, j);
                array[i][j] = input.nextInt();
            }
        }
        return array;
    }
    public static void main(String[] args){
        Scanner input = new Scanner(System.in);
        System.out.println("Enter row: ");
        int n = input.nextInt();
        System.out.println("Enter column: ");
        int m = input.nextInt();
        int[][] a = new int[n][m];
```

```
        a = Matrix(n, m);
        for(int i = 0; i < n; i++){
            for(int j = 0; j < m; j++){
                System.out.printf( "%-3d", a[i][j]);
            }
            System.out.println("");
        }
    }
}
```

```
Enter row:
3
Enter column:
4
array[0][0] = 1
array[0][1] = 2
array[0][2] = 3
array[0][3] = 4
array[1][0] = 5
array[1][1] = 6
array[1][2] = 7
array[1][3] = 8
array[2][0] = 9
array[2][1] = 10
array[2][2] = 11
array[2][3] = 12

1  2  3  4
5  6  7  8
9  10 11 12
```

**مثال هفتم** :یک متد که جمع ماتریسی بین دو ماتریس هم اندازه را انجام میدهد:

```
public class testclass{
    public static int[][] Matrix(int[][] a,int[][] b){
        int[][] array = new int[a.length][a[1].length];
        for(int i = 0; i < a.length; i++){
            for(int j = 0; j < a[i].length; j++){
                array[i][j] = a[i][j] + b[i][j];
            }
        }
        return array;
    }
    public static void main(String[] args){
        int[][] n = new int[2][2];
        int[][] m = new int[2][2];
        int[][] k = new int[2][2];
        for(int i = 0; i < 2; i++){
```

```
            for(int j = 0; j < 2; j++){
                n[i][j] = j + 1;
            }
        }
        for(int i = 0; i < 2; i++){
            for(int j = 0; j < 2; j++){
                System.out.printf(n[i][j] + " ");
            }
            System.out.println("");
        }
        System.out.println(" +");
        for(int i = 0; i < 2; i++){
            for(int j = 0; j < 2; j++){
                m[i][j] = i * 2;
            }
        }
        for(int i = 0; i < 2; i++){
            for(int j = 0; j < 2; j++){
                System.out.printf(m[i][j] + " ");
            }
            System.out.println("");
        }
        System.out.println(" =");
        k = Matrix(m, n);
        for(int i = 0; i < 2; i++){
            for(int j = 0; j < 2; j++){
                System.out.printf(k[i][j] + " ");
            }
            System.out.println("");
        }
    }
}
```

```
1 2
1 2
 +
0 0
2 2
 =
1 2
3 4
```

**مثال هشتم** :یک متد که بررسی میکند آیا ضرب بین دو ماتریس امکان پذیر است یا خیر ، و سپس دو ماتریس را در هم ضرب میکند:

```
public class MatrixMultiplication {
    public static void main(String[] args) {
```

```
        int[][] matrix1 = {
                {1, 2, 3},
                {4, 5, 6},
                {7, 8, 9}
        };

        int[][] matrix2 = {
                {9, 8, 7},
                {6, 5, 4},
                {3, 2, 1}
        };

        int[][] result = multiplyMatrices(matrix1, matrix2);

        printMatrix(result);
    }

    public static int[][] multiplyMatrices(int[][] mat1, int[][] mat2) {
        int rows1 = mat1.length;
        int cols1 = mat1[0].length;
        int cols2 = mat2[0].length;

        int[][] result = new int[rows1][cols2];

        for (int i = 0; i < rows1; i++) {
            for (int j = 0; j < cols2; j++) {
                for (int k = 0; k < cols1; k++) {
                    result[i][j] += mat1[i][k] * mat2[k][j];
                }
            }
        }

        return result;
    }

    public static void printMatrix(int[][] mat) {
        for (int[] row : mat) {
            for (int num : row) {
                System.out.print(num + " ");
            }
            System.out.println();
        }
    }
}
```

```
30 24 18
84 69 54
138 114 90
```

## برنامه نویسی شی گرا یا Object-Oriented Programming

****در این بخش ابتدا مفاهیم را توضیح میدهیم و سپس به نحوه پیاده سازی این مفهوم میپردازیم****

که به اختصار به آن **OOP** می‌گویند، نوعی رویکرد برنامه نویسی است که در آن طراحی کردن یک نرم افزار، عوض توابع و منطق، حول داده‌ها و شی‌ها) یا همانobject ها (می‌گردد .**شی گرایی** به توسعه دهندگان این امکان را می‌دهد تا با تعریف آبجکت‌ها یا شی‌های مختلف، سیستم‌های نرم افزاری را **مدل سازی** کنند. یک **آبجکت (Object)** می‌تواند از هر نوع داده‌ای تشکیل شده و شامل یک یا چند ویژگی باشد.

**برنامه نویسی شی گرا** یکی از معروف‌ترین و پر استفاده ترین رویکرد است که در زمینه‌های مختلف از جمله نرم افزار کامپیوتری، نرم افزار موبایل، بازی و سیستم‌های شبیه سازی مورد استفاده قرار می‌گیرد. در **برنامه نویسی با رویکرد شی گرایی**، ابتدا هدف به صورت مجموعه‌ای از **کلاس(Class)** ها ساخته می‌شود و در ادامه ی روند توسعه‌ی یک برنامه، آبجکت‌هایی از این کلاس‌ها ساخته شده و تغییر پیدا می‌کنند و مورد استفاده قرار می‌گیرند. هر **آبجکت** از تعدادی **خصوصیت یا Attribute** ساخته شده است که می‌توانند مقادیر و انواع مختلفی داشته باشند، مانند یک عدد، یک رشته و یا یک کلاس دیگر.

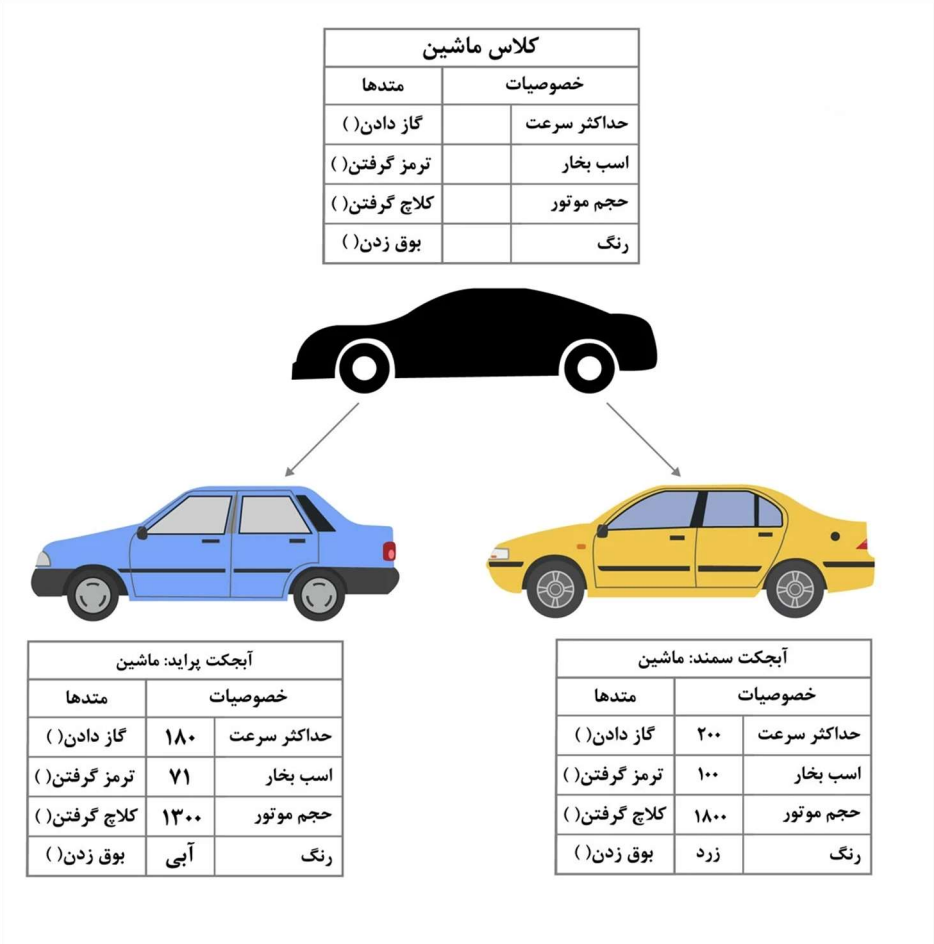
به عنوان مثال در یک بازی کامپیوتری، **اسلحه** یک کلاس است و فرضا AK-47 یک **آبجکت** که از این کلاس بوجود آمده و شامل خصوصیت‌های ویژه‌ی خود است، مثلا تعداد گلوله، تعداد شلیک در ثانیه، رنگ، وزن و خیلی از موارد دیگر. اصطلاحات و اصول مختلفی در برنامه نویسی با رویکرد **شی گرایی** وجود دارد که در ادامه به طور کامل با ذکر یک نمونه به این موارد می‌پردازیم.

### ساختار برنامه نویسی شی گرا چیست؟

به طور کلی **برنامه نویسی شی گرا** از 4 بخش اصلی زیر تشکیل شده است:


- **کلاس (Class) :** اولین بخش یک **برنامه نویسی شی گرا**، کلاس است. کلاس در واقع حکم کالبد یک **شی (Object)** را دارد. تمامی متدها، خصوصیت‌ها، تغییرات قابل اعمال، دسترسی‌هایی به شی و... در بدنه ی یک کلاس تعریف می‌شوند.
- **اشیاء (Objects) :**شی ها یا آبجکت‌ها در واقع نمونه‌هایی (Instances) از کلاس‌ها هستند که با داده‌ی خاصی ساخته شده‌اند. آبجکت‌ها می‌توانند نمایانگر یک شی در دنیای واقعی باشند، مثل یک ماشین، یک اسلحه، یک سرور و یا یک شی خیالی باشند، مثلا یک سطر از جدول نمرات دانشجو.

- **متد ها (Methods) :** متدها (که گاهی به آن **Functionیا تابع** هم می‌گویند) در واقع توابعی هستند که داخل کلاس‌ها تعریف می‌شوند تا رفتار یک آبجکت را مشخص کنند. در **برنامه نویسی شی گرا**، برنامه نویسان، یک کلاس را **کپسوله سازی (Encapsulation)** می‌کنند تا از خارج از کلاس، امکان ایجاد تغییرات در داخل کلاس وجود نداشته باشد و در عوض، هر تغییری و هر درخواست، به وسیله‌ی متدها قابل انجام باشد. یک کلاس می‌تواند از هر تعداد و نوع متدی ساخته شود. به عنوان مثال کلاس انسان شامل متد های راه رفتن، حرف زدن، خوابیدن و.. است
- **خصوصیات (Attributes) : خصوصیات** که به آن **ویژگی** هم می‌گویند در واقع متغیرهایی هستن که در بدنه‌ی **کلاس (Class)** تعریف شده و خصوصیات یک مدل در قالبAttribute ها پیاده سازی می‌شوند. به عنوان مثال کلاس انسان می‌تواند شامل خصوصیات قد، وزن، رنگ چشم، سن، جنسیت و... باشد.



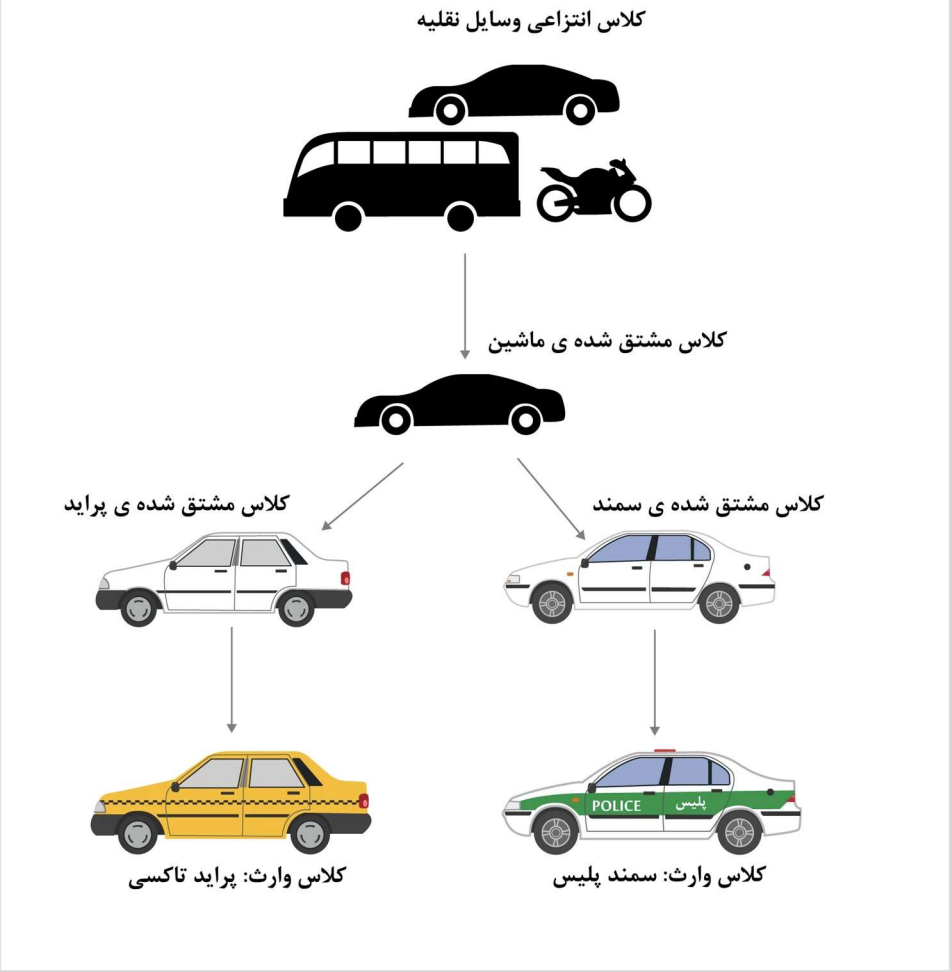

برای اینکه مفهوم موارد ذکر شده را دقیق تر متوجه شوید، تصویر بالا را در نظر بگیرید. ماشین یک **کلاس (Class)** است و سمند و پراید، یک **آبجکت (Object)** و در واقع نمونه‌هایی از کلاس ماشین هستند. حداکثر سرعت، اسب بخار، حجم موتور و رنگ، خصوصیات کلاس ماشین هستند و گاز دادن، ترمز کردن، کلاچ گرفتن و بوق زدن نیز متد های این کلاس می‌باشند.

## اصول اصلی برنامه نویسی شی گرایی چیست؟


- **کپسوله سازی (Encapsulation) :** اصل **کپسوله سازی** به این معنا است که اطلاعات و خصوصیات یک آبجکت، تنها از داخل آن آبجکت قابل تغییر باشد و هیچ آبجکتی در خارج نتواند تغییراتی در آبجکت‌های دیگر بوجود بیاورد، تنها راه خواندن و تغییر دادن داده‌ها یا خصوصیات یک آبجکت، باید به وسیله‌ی متد های تعریف شده میسر باشد. هدف از این اصل جلوگیری از بوجود آمدن خطا های منطقی و تغییرات ناخواسته است.
- **انتزاع (Abstraction) :** یکی از مفاهیم مهم در **برنامه نویسی شی گرا**، بحث انتزاع است .**مفهوم انتزاع** ممکن است کمی گنگ باشد. در برنامه نویسی، **کلاس های Abstract** برای ساخت یک اساس و مفهوم کلی ساخته می‌شوند، برخلاف مثال‌هایی از کلاس‌هایی که بالاتر ذکر کردیم، کلاس Abstract هیچ پیاده سازی‌ای ندارد و تنها تعاریف در آنها قرار گرفته‌اند. به عنوان مثال متد مربوط به گاز دادن در آن قرار دارد اما بدنه‌ی متد خالی است. به همین دلیل در کد نویسی، **نمونه سازی یا ساخت شی از کلاس های Abstract ممکن نیست** و در عوض کلاس‌ها می‌بایست از یک کلاس Abstract مشتق بشوند. در بخش بعدی ، مفهوم انتزاع را در قالب مثال نیز نمایش خواهیم داد .
- **ارث بری (Inheritance) :**کلاس‌ها می‌توانند از دیگر کلاس‌ها، خصوصیات و متد ها را به **ارث** ببرند. به این ترتیب می‌توان از کدها استفاده‌ی مجدد کرد و علاوه بر صرف جویی در زمان، باعث خوانایی کد و کاهش پیچیدگی آن نیز می‌شود .**کلاس های مشتق شده** که به آنها **Child Class** می‌گویند، از **کلاس والد یا Parent Class** ارث بری می‌کنند. کلاس های مشتق شده همچنین می‌توانند خصوصیات و متد های جدیدی نیز داشته باشند. گاهی اوقات به کلاس های والد، **Superclass** و به کلاس های فرزند **Subclass** نیز می‌گویند .
- **چند ریختی (Polymorphism) :** در بخش **ارث بری** گفتیم که کلاس‌ها می‌توانند از کلاس دیگری مشتق شوند و خصوصیات و متد های آنها را داشته باشند. اما گاهی اوقات می‌خواهیم این متد ها و خصوصیات، طور دیگری رفتار کنند یا مقادیر دیگری داشته باشند. قابلیت **چند ریختی یا پلی‌مورفیسم** به ما این اجازه را می‌دهد تا این تغییرات را به سادگی بوجود بیاوریم. بدین منظور در کلاس فرزند، متدی که قصد تغییر آن را داریم، مجددا می‌نویسیم و بدنه‌ی آن متد را در **کلاس فرزند** به صورت دلخواه بازنویسی می‌کنیم.

## مثالی جامع از یک برنامه یOOP

در این قسمت یک مثال جامع از مباحثی که تاکنون یاد گرفته اید می‌آوریم تا کل **مفاهیم شی گرایی** برای شما واضح تر شود. تصویر زیر را به دقت نگاه کنید:




در تصویر بالا، کلاس وسایل نقلیه، یک **کلاس انتزاعی یا Abstract** است که تعاریف وسایل نقلیه در آن قرار گرفته است. مانند داشتن چرخ، امکان گاز دادن، بوق زدن و... اما هیچ پیاده سازی ای صورت نگرفته است. یعنی داشتن چرخ برای یک وسیله‌ی نقلیه تعریف شده اما چه تعداد چرخ داشته باشد مشخص نشده. کلاس ماشین، یک کلاس انتزاعی مشتق شده از کلاس وسایل نقلیه است که بعضی از موارد تعریف شده در کلاس وسایل نقلیه در آن پیاده سازی شده است.

به عنوان مثال تعداد چرخ ها برابر 4 است. دو کلاس سمند و پراید، کلاس‌های مشتق شده از کلاس ماشین هستند که هرکدام ویژگی‌های خاص خودشان را دارند. این دو کلاس دیگر انتزاعی نمی‌باشند و امکان ساخت شی یا آبجکت از آن ها وجود دارد. مثلا ساخت **آبجکت (Object)**از نوع سمند مثل سمند سورن، سمند LX ، یا ساخت آبجکت پراید مانند پراید 131، 141 و ...**کلاس (Class)** پراید تاکسی یک کلاس است که از کلاس پراید **ارث بری (Inheritance)** کرده و رنگ این پراید نیز به زرد تغییر پیدا کرده است. کلاس سمند پلیس نیز یک کلاس دیگر است که از کلاس سمند ارث بری کرده است. سمند پلیس، متد بوق زدن را تغییر داده است، یعنی از خاصیت **چند ریختی (Polymorphism)** استفاده کرده. همچنین متد آژیر زدن را نیز اضافه کرده است.

**مزایا و معایب برنامه نویسی شی گرا :**


اکنون وقت آن است که به نحوه پیاده سازی این مفاهیم در جاوا بپردازیم .

هر برنامه ای در جاوا حداقل دارای یک کلاس است :
این ساختار ساده یک کلاس است که مابقی کد های مربوط به این کلاس درون این بلوک نوشته میشود .

```
public class Main {
    int x = 5;
}
```

من برای استفاده از این کلاس یک instance از روی آن میسازم (ساختن شی از روی کلاس) و از محتویات و قابلیت های کلاس استفاده میکنم

```
public class Main {
    int x = 5;

    public static void main(String[] args) {
      Main myObj = new Main();
      System.out.println(myObj.x);
    }
  }
```

```
5
```

ما میتوانیم از یک کلاس چند شی بسازیم و از همه آنها هم استفاده کنیم
برای مثال من از کلاس Main در این مثال دو شی ساخته و از آنها استفاده میکنم.

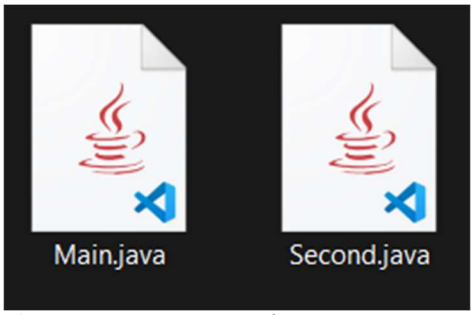
```java
public class Main {
    int x = 5;

    public static void main(String[] args) {
      Main myObj1 = new Main();  // Object 1
      Main myObj2 = new Main();  // Object 2
      System.out.println(myObj1.x);
      System.out.println(myObj2.x);
    }
  }
```

```
5
5
```

نکته : تا این جا ما از کلاس هایی که نوشتیم نمونه نساختیم ولی از قابلیت های آن استفاده کردیم . چگونه ممکن است؟ در واقع ما با `static` تعریف کردن متد `main` برنامه بدون نیاز به ساختن شی از روی کلاس متد اصلی را اجرا کردیم. (در ادامه بیشتر بررسی میکنیم این موضوع را)

علاوه بر `Main` میخواهم یک کلاس به عنوان `Second` تعریف کنم ، برای اینکار در پوشه برنامه خود یک فایل جدید به این نام و با پسوند .java میسازم




تصویری از درون پوشه برنامه که شامل دو فایل جاوا است (دو کلاس)

کد های مربوط به هر کلاس را به این صورت مینویسم :

```java
public class Main {
    int x = 5;
  }
```

```java
class Second {
  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```

```
5
```

## مشخصه های کلاس Attributes : (تحت عنوان fields هم شناخته میشوند.)

مشخصه های یک کلاس، متغیر های درون یک کلاس هستند .

این کلاس دارای دو مشخصه تحت عنوان های x و y است :

```
public class Main {
    int x = 5;
    int y = 3;
  }
```

در مثال های قبل برای اینکه به مشخصه های کلاس دسترسی پیدا کنیم دیدیم که باید از روی کلاس‌هایمان شی یا object بسازیم.

میتوانیم از این طریق مقدار متغیر های درون کلاس را مقدار دهی کنیم یا آنها را دهیم (در صورت امکان) :

```
public class Main {
    int x;
    int y=10;

    public static void main(String[] args) {
      Main myObj = new Main();
      myObj.x = 40;
      myObj.y = 30;
      System.out.println(myObj.x);
      System.out.println(myObj.y);

    }
  }
```

```
40
30
```

نکته : در صورتی که متغیر درون کلاس final تعریف شده باشد نمیتوانیم آن را تغییر دهیم :

```
public class Main {
    final int x = 10;

    public static void main(String[] args) {
      Main myObj = new Main();
      myObj.x = 25; /* will generate an error: cannot assign a value to a
                                        final variable*/
      System.out.println(myObj.x);
    }
```

```
  }
```

```
Main.java:6: error: cannot assign a value to final variable x
      myObj.x = 25; // will generate an error: cannot assign a value to a
final variable
            ^
1 error
```

تغییر دادن متغیر های مربوط به هر کدام از شئ های که از روی یک کلاس واحد ساخته شده بر دیگری تاثیر نمیگذارد . (مگر در موارد خاص که برسی خواهد شد)

```
public class Main {
    int x = 5;
    public static void main(String[] args){
        Main myObj1 = new Main();
        Main myObj2 = new Main();
        myObj1.x = 10;
        System.out.println(myObj1.x);
        System.out.println(myObj2.x);
    }
}
```

```
10
5
```

## تعدیل کننده ها Modifiers :

تا به اینجای جاوا ما از کلمه های کلیدی مثل `public` یا `static` استفاده کردیم و الان وقت آن است که چیستی آن ها را بدانیم .

### تعدیل کننده دسترسی Access Modifiers :

| Modifier | توضیحات |
|---|---|
| `public` | کد برای همه کلاس ها قابل دسترسی است |
| `private` | کد فقط در کلاسی که تعریف شده قابل دسترسی است |
| *`default`* | کد فقط در همان بسته تعریف شده قابل دسترسی است |
| `protected` | کد فقط در همان بسته و زیر کلاس هایش (درس های بعد...) |

### تعدیل کننده های غیر دسترسی Non-Access Modifiers :

| Modifier | توضیحات |
|---|---|
| `final` | متد یا مشخصه را غیرقابل تغییر میکند |
| `static` | متد یا مشخصه به کلاس وابسته هستند و نه به شی کلاس(بدون ساخت نمونه و در صورت دسترسی به کلاس میتوان از آن استفاده کرد) |
| `abstract` | روند انتزاعی ساخت برای پنهان کردن جزییات خاص که در درس های بعد به آن میپردازیم |

بیاید تفاوت `static` و `public` را در یک کد مشاهده کنیم :

```
public class Main {
    // Static method
    static void myStaticMethod() {
      System.out.println("Static methods can be called without creating
objects");
    }

    // Public method
    public void myPublicMethod() {
      System.out.println("Public methods must be called by creating
objects");
    }

    // Main method
    public static void main(String[ ] args) {
      myStaticMethod(); // Call the static method
      // myPublicMethod(); This would output an error

      Main myObj = new Main(); // Create an object of Main
      myObj.myPublicMethod(); // Call the public method
    }
  }
```

```
Static methods can be called without creating objects
Public methods must be called by creating objects
```

در این مثال همانطور که مشاهده کردید هر دو متد `myPublicMethod()` و `myStaticMethod()` درون کلاس `Main` قرار دارند و برای دسترسی به متد `myPublicMethod()` نیاز داشتیم از روی کلاس آن یک نمونه بسازیم ، اما برای متد `myStaticMethod()` اینطور نبود و ما در صورتی که درون اسکوپ و محدوده ای باشیم که کلاس مربوط به آن در دسترس باشد میتوانیم بدون ساختن شی از آن استفاده کنیم.

در مثال های قبل متوجه شدیم برای دسترسی به مشخصه های یک کلاس نیاز داریم از روی کلاس شئ بسازیم .

ما میتوانیم با `static` تعریف کردن مشخصه ها(متغیرهای کلاس) مشکل نیاز به ساخت شئ برای دسترسی را حل کنیم.

مشخصه ما در این حالت برای تمام اشیاء ساخته شده از کلاس **یکی** است (به یک خانه از حافظه اشاره میکنند ).

## متد سازنده یا Constructor :

متد سازنده ، متدی است که برای مقدار دهی اولیه استفاده میشود ، هنگامی که یک شی از کلاس ساخته میشود متد سازنده فراخوانی میشود .

```
// Create a Main class
public class Main {
    int x;  // Create a class attribute
```

```
    // Create a class constructor for the Main class
    public Main() {
      x = 5;  // Set the initial value for the class attribute x
    }

    public static void main(String[] args) {
      Main myObj = new Main(); // Create an object of class Main (This will
call the constructor)
      System.out.println(myObj.x); // Print the value of x
    }
```

**توجه :** نام متد سازنده باید هم نام کلاس باشد و متد نباید مقداری را برگرداند .

دقت کنید که اگر ما متد سازنده برای کلاس تعریف نکنیم ، جاوا یک متد پیش فرض ایجاد میکند با این حال کاری را انجام نمیدهد .

### پارامتر های متد سازنده :

متد سازنده مانند همه متد ها میتواند مقدار ورودی داشته باشد و با توجه به مقدار ورودی که ما یا کاربر برای آن میفرستیم مقدار دهی اولیه را انجام دهد .

```
public class Main {
    int actorAge;
    String actorName;

    public  Main(int year, String name) {
        actorAge = year;
        actorName = name;
    }

    public static void main(String[] args) {
        Main actor = new Main(29, "Ebrahim");
        System.out.println(actor.actorAge + " " + actor.actorName);
    }
}

// Outputs 29 Ebrahim
```

```
29 Ebrahim
```

### کلید واژه this :

گاهی ممکن است مشخصه های متد با پارامتر های ورودی متد سازنده یکی باشد ، در این مواقع برای جلوگیری از اشتباه از کلید واژه `this` به این صورت استفاده میکنیم:

```java
public class Main {
    int age;
    String name;

    public Main(String name , int age) {
        this.age = age;
        this.name = name;
    }

    public static void main(String[] args) {
        Main person = new Main("javad" , 20);
        System.out.println(person.name + " is " + person.age);
    }
}
```

```
javad is 20
```

این خط کد را بدون استفاده از this تصور کنید

```java
this.age = age;
```

به این صورت :

```java
 age = age;
```

این کد صحیح میباشد و برنامه به ما ارور نمیدهد،  اما مقدار دهی را انجام نمیدهد.

یعنی خروجی به این شکل خواهد بود:

```
javad is 0
```

این صفر مقداری پیش فرض age است

### this به شئ اشاره دارد که در آینده قرار است از روی کلاس ساخته شود .

برای تعیین مقادیر پیش فرض برای وقتی که از روی کلاس شی ساخته شود ولی آرگومانی به متد سازنده ارسال نشود ، ما میتوانیم خودمان با تعریف متد سازنده دیگری به مشخصه ها را به صورت پیش فرض و سفارشی مقدار دهی کنیم.

در نظر داشته باشید ما میتوانیم چندین متد سازنده برای حالت های مختلفی ایجاد کنیم  ، برای مثال حالتی را در نظر بگیرید که فقط آرگومان name به متد سازنده ما ارسال شده ، در این حالت ما یک متد سازنده مناسب تعریف میکنیم که name را دریافت و مقدار دهی کند و age را به صورت پیش فرض مقدار دهی کند .

یک حالت دیگر هم زمانی است که هیچ مقداری را به متد سازنده ارسال نشود که آن را هم میتوانیم تعریف کنیم .

اکنون یکی از این حالات را مینویسیم

```java
public class Main {
    int age;
    String name;
    public Main(){
```

```
        name = "EMPTY";
        age = 0;
    }
    public Main(String name , int age) {
        this.age = age;
        this.name = name;
    }

    public static void main(String[] args) {
        Main person1 = new Main("javad"  , 20);
        System.out.println(person1.name + " is " + person1.age);
        Main person2 = new Main();
        System.out.println(person2.name + " is " + person2.age);

    }
}
```

```
javad is 20
EMPTY is 0
```

این قسمت کد را میتوانیم به صورت دیگری هم بنویسیم :

```
public Main(){
  name = "EMPTY";
  age = 0;
}
```

---

```
public Main(){
   this("EMPTY",0)
}
```

**نکته : سربارگیری متد سازنده با استفاده از دستور this مانند مثال بالا باید در اولین خط کد متد باشد.**

در این حالت اتفاق که رخ میدهد به شرح زیر است :

1. یک شی از روی کلاس Main بدون مقدار دهی اولیه ساخته میشود
2. در این حالت متد سازنده ای که بدون پارامتر ورودی است فراخوانی میشود
3. کاری که متد سازنده بدون پارامتر پس از فراخوانی انجام میدهد ، فراخوانی متد سازنده **همین**( ()this ) شی است ، ولی این بار با فرستادن آرگومان به آن . که آرگومان ها همان پیش‌فرض های ما هستند .

ما معمولا متد سازنده را اینگونه سربارگیری میکنیم (method overloading) .

## کپسوله سازی (Encapsulation):

**"کپسوله سازی (Encapsulation) :** اصل **کپسوله سازی** به این معنا است که اطلاعات و خصوصیات یک آبجکت، تنها از داخل آن آبجکت قابل تغییر باشد و هیچ آبجکتی در خارج نتواند تغییراتی در آبجکت‌های دیگر بوجود بیاورد، تنها راه خواندن و تغییر دادن داده‌ها یا خصوصیات یک آبجکت، باید به وسیله‌ی متد های تعریف شده میسر باشد. هدف از این اصل جلوگیری از بوجود آمدن خطا های منطقی و تغییرات ناخواسته است. **"**

ما در کلاس ها متغیر ها را `private` تعریف میکنیم و برای مقدار دهی آنها متد های get و set استفاده میکنیم برای دسترسی و به تغییر دادن مقدار مشخصه ها ( برای مقدار دهی اولیه از متد سازنده استفاده میکنیم و برای تغییر متغیر ها و دسترسی به آنها از get و set استفاده میکنیم ، حتی میتوانیم مقدار دهی اولیه را با سازنده انجام ندهیم و بعد از ساختن شئ از set استفاده کنیم . )

```
public class Person {
    private String name; // private = restricted access

    // Getter
    public String getName() {
      return name;
    }

    // Setter
    public void setName(String name) {
      this.name = name;
    }
  }
```

اکنون در متد اصلی برنامه از روی کلاس `Person` یک شی میسازیم و سعی میکنیم مشخصه `name` را مقدار دهی کنیم .

همانطور که میدانید مشخصه `name` را به صورت `private` تعریف کردیم و نمیتوانیم آن را به این صورت مقدار دهی کنیم :

```
public class Main {
    public static void main(String[] args) {
      Person myObj = new Person();
      myObj.name = "John";  // error
      System.out.println(myObj.name); // error
    }
  }
```

```
Main.java:3: error: cannot find symbol
      Person myObj = new Person();
      ^
  symbol:   class Person
  location: class Main
Main.java:3: error: cannot find symbol
      Person myObj = new Person();
                         ^
```

```
  symbol:   class Person
  location: class Main
2 errors
```

راه درست مقدار دهی و استفاده از آن به وسیله متد های get و set صورت میگیرد :

```java
public class Main {
    public static void main(String[] args) {
      Person myObj = new Person();
      myObj.setName("Ali"); // Set the value of the name variable to "Ali"
      System.out.println(myObj.getName());
    }
  }
// Outputs "Ali"
```

```
Ali
```

درون متد اصلی برنامه از کلاس `Person` که در فایلی جدا کد آن را نوشتیم یک شئ ساختیم و با استفاده از متد های getter و setter آن را مقدار دهی کردیم .

اکنون یک با برسی یک مثال ملموس تر سعی میکنیم مطالبی را که تا کنون درباره برنامه نویسی شئ گرا مطالعه کردیم را تثبیت کنیم :

جاوا نوع داده‌ای که با آن بتوانیم با اعداد مختلط کار کنیم را دارا نمیباشد ، اکنون با استفاده از مفاهیمی که تا کنون یاد گرفته ایم یک کلاس تحت عنوان `ComplexNumber` یا `Complex` تعریف میکنیم که با ساختن شی از روی آن بتوانیم اعداد مختلط را تعریف کنیم .

**یاداوری از ریاضیات :**

$$A = a + bi\,,\ B = c + di \qquad a,b,c,d \in \mathbb{R} \qquad \begin{cases} i = \sqrt{-1} \\ i^2 = -1 \end{cases}$$

$A$ یک عدد مختلط است که بخش حقیقی (Real) آن $a$ و $c$ و بخش موهومی (Imaginary) آن $di$ و $bi$ می باشد

جمع دو عدد مختلط به این صورت است :

$$A + B = (a + bi) + (c + di) = (a + c) + (b + d)i$$

ضرب دو عدد مختلط به این صورت است :

$$A \times B = (a + bi) \times (c + di) = ac + adi + cbi + bdi^2$$

$$\longrightarrow ac + (ad + cb)i - bd = (ac - bd) + (ad + cd)i$$

تقسیم دو عدد مختلط به این صورت است :

$$\frac{A}{B} = \frac{a + bi}{c + di} = \frac{a + bi}{c + di} \times \frac{c - di}{c - di} = \frac{(ac + bd) + (cb - ad)i}{(c)^2 + (d)^2}$$

شروع به نوشتن فایل مربوط به کلاس Complex میکنیم :

```
public class Complex {
  double Real; //Real part
  double Imaginary; //Imaginary part

  //Constructor method for initializing
  public Complex(double Real,double Imaginary){
      this.Real = Real;
      this.Imaginary = Imaginary;
  }
}
```

در کلاسی دیگر متد اصلی برنامه را پیاده میکنیم :

```
public class Main {
  public static void main(String[] args){
      Complex complex1 = new Complex(2,4);
      Complex complex2 = new Complex(3,6);
  }
}
```

اکنون ما یک کلاس یا نوع داده ای برای تعریف اعداد مختلط تعریف کردیم .

برای چاپ کردن متغیر ها میتوانیم یک متد برای پرینت عدد درون کلاس مانند مثال زیر قرار دهیم (راه بهتری نیز وجود دارد ) :

همچنین متد های getter و setter را در ادامه تعریف میکنیم و کلاس را گسترش میدهیم :

```
public class Complex {
    private double Real; //Real part
    private double Imaginary; //Imaginary part

    public Complex() {
        this.Real = 0;
        this.Imaginary = 0;
    }

    //Constructor method for initializing
    public Complex(double Real, double Imaginary) {
        this.Real = Real;
        this.Imaginary = Imaginary;
    }
```

```
    public double getReal() {
        return Real;
    }
    public double getImaginary() {
        return Imaginary;
    }
    public void setReal(double real) {
        Real = real;
    }
    public void setImaginary(double imaginary) {
        Imaginary = imaginary;
    }

    public void PrintComplex(){
        if(this.Imaginary>0) {
            System.out.println(this.Real + "+" + this.Imaginary + "i");
        } else if (this.Imaginary<0){
            System.out.println(this.Real + "" + this.Imaginary + "i");
        }
        else {
            System.out.println(this.Real);
        }
    }

}
```

```
public class Main {
    public static void main(String[] args){
        Complex complex1 = new Complex(2,4);
        Complex complex2 = new Complex(3,-6);
        Complex complex3 = new Complex(-9,0);

        complex1.PrintComplex();
        complex2.PrintComplex();
        complex3.PrintComplex();

    }
}
```

```
2.0+4.0i
3.0-6.0i
-9.0
```

متد PrintComplex برای ما اعداد مختلط تعریف شده را با استفاده از ساختار های کنترلی در حالت هایی که قسمت موهومی مثبت ، منفی یا صفر باشند با فرم مناسبی چاپ میکند

اگر مستقیما شی را چاپ کنیم چه اتفاق می افتد؟

```
System.out.println(complex1);
```

```
Complex@568db2f2
```

جاوا به صورت پیش فرض درون خودش یک متد به نام toString دارد ، وقتی که یک شی را با دستور println() چاپ میکنیم این متد روی آن شی فراخوانی میشود و خروجی به فرم بالا به ما نشان میدهد

این خروجی در واقع نام کلاس به همراه آدرس آن در حافظه است

همه کلاس ها در جاوا از کلاس Object جاوا ارث بری میکند ، درباره ارث بری در ادامه بحث خواهیم کرد ، اما اینجا باید بدانیم که متد toString مربوط به آن کلاس است ، وقتی همه کلاس های ما از کلاس Object ارث بری کنند ، زمانی که ما بخواهیم شئ از کلاسمان را چاپ کنیم ، دستور println() به دنبال متد toString در کلاس ما میگردد و در صورتی که نتواند آن را در آنجا پیدا کند ، متدی که درون کلاس Object وجود دارد را فراخوانی میکند .

ما اکنون میخواهیم درون کلاسی که ساختیم این متد را بازنویسی (Override) کنیم ، تا بتوانیم خروجی را هنگام چاپ از طریق println() به صورت دلخواه تغییر دهیم .

برای Override کردن یک متد مثل toString که از قبل درون جاوا وجود دارد نیاز است خط قبل تعریف متد این را ذکر کنیم به این صورت :

```
public class Complex {
    private double Real; //Real part
    private double Imaginary; //Imaginary part

    public Complex() {
        this.Real = 0;
        this.Imaginary = 0;
    }

    //Constructor method for initializing
    public Complex(double Real, double Imaginary) {
        this.Real = Real;
        this.Imaginary = Imaginary;
    }


    public double getReal() {
        return Real;
    }
    public double getImaginary() {
```

```
        return Imaginary;
    }
    public void setReal(double real) {
        Real = real;
    }
    public void setImaginary(double imaginary) {
        Imaginary = imaginary;
    }


    @Override
    public String toString() {
        String output ;
        if(this.Imaginary>0) {
            output = (this.Real + "+" + this.Imaginary + "i");
        } else if (this.Imaginary<0){
            output = (this.Real + "" + this.Imaginary + "i");
        }
        else {
            output = String.valueOf((this.Real));
        }
        return output;
    }
}
```

```
Public class Main {
    Public static void main(String[] args){
        Complex complex1 = new Complex(2,4);
        Complex complex2 = new Complex(3,-6);
        Complex complex3 = new Complex(-9,0);

        System.out.println(complex1);
        System.out.println(complex2);
        System.out.println(complex3);

    }
}
```

```
2.0+4.0i
3.0-6.0i
-9.0
```

اکنون وقت آن است که با توجه به مطلب و فرمول هایی که یاداوری شد چند متد برای ضرب ، تقسیم و جمع اعداد مختلط پیاده کنیم.

```
public Complex add(Complex num) {

    return  new Complex(this.Real + num.getReal(), this.Imaginary +
num.getImaginary());
}

public Complex subtract(Complex num) {

    return  new Complex(this.Real - num.getReal(), this.Imaginary -
num.getImaginary());
}

public Complex multiply(Complex num) {

    return new Complex(this.Real * num.getReal() - this.Imaginary *
num.getImaginary(), this.Real * num.getImaginary() + this.Imaginary *
num.getReal());

}

public Complex divide(Complex num) {

    double denominator = num.getReal() * num.getReal() + num.getImaginary() *
num.getImaginary();

    return  new Complex((this.Real * num.getReal() + this.Imaginary *
num.getImaginary()) / denominator, (this.Imaginary * num.getReal() -
this.Real * num.getImaginary()) / denominator);

}
```

## وراثت (Inheritance) :

**ارث بری (Inheritance) :** کلاس‌ها می‌توانند از دیگر کلاس‌ها، خصوصیات و متد ها را به **ارث** ببرند. به این ترتیب می‌توان از کدها استفاده‌ی مجدد کرد و علاوه بر صرف جویی در زمان، باعث خوانایی کد و کاهش پیچیدگی آن نیز می‌شود . **کلاس های مشتق شده** که به آنها **Child Class** می‌گویند، از **کلاس والد یا Parent Class** ارث بری می‌کنند. کلاس های مشتق شده همچنین می‌توانند خصوصیات و متد های جدیدی نیز داشته باشند. گاهی اوقات به کلاس های والد، **Superclass** و به کلاس های فرزند **Subclass** نیز می‌گویند

کلمه کلیدی که برای ارث بری از آن استفاده میکنیم `extends` است .

```
public class Person {
    private String name;
    private String lastName;
    private String nationalCode;

    public Person(){
        this("empty", "empty", "0000000000");
    }

    public Person(String name, String lastName, String nationalCode){
        this.name = name;
        this.lastName = lastName;
        this.nationalCode = nationalCode;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getNationalCode() {
        return nationalCode;
    }

    public void setNationalCode(String nationalCode) {
        this.nationalCode = nationalCode;
    }

    @Override
    public String toString(){
```

```
        return "name: " + getName() + ",  lastname: " + getLastName() +
",  national code: " + getNationalCode();
    }
}
```

در اینجا ما کلاس Person را داریم ، این کلاس دارای سه مشخصه attribute می باشد و با استفاده از متد های getter و setter از دسترسی مستقیم به آن محافظت کردیم ، این کلاس دارای دو متد سازنده است. یکی برای زمانی که شی با استفاده از مقدار دهی ساخته میشود و دیگری برای زمانی که بدون مقدار دهی اولیه از آن شئ ساخته میشود .

همچنین در این کلاس متد toString را override کردیم برای زمانی که یک شی از کلاس را چاپ کنیم .

اکنون میخواهیم یک کلاس دیگر تحت عنوان Student تعریف کنیم ، هر فرد دانش آموز تمام مشخصه های کلاس Person را دارا میباشد ، پس نیازی به تعریف مجدد آنها در کلاس Student نمی باشد ، کافی است که کلاس Student را طوری تعریف کنیم که از کلاس Person ارث بری کند ، برای اینکار از کلمه کلیدی extends استفاده میکنیم .

```
class Student extends Person {
    private String grade;
    private double pointAverage;
    public Student(){
        this("empty", "empty", "0000000000", "zero", 0);
    }
    public Student(String name ,String lastName ,String natonalCode ,String
grade ,double pointAverage){
        super(name, lastName, natonalCode);
        this.grade = grade;
        this.pointAverage = pointAverage;
    }

    public String getGrade() {
        return grade;
    }

    public void setGrade(String grade) {
        this.grade = grade;
    }

    public double getPointAverage() {
        return pointAverage;
    }

    public void setPointAverage(double pointAverage) {
        if(pointAverage <= 20)
        this.pointAverage = pointAverage;
    }
    public boolean passCheck(){
        if(pointAverage >= 10){
            return true;
```

```
        }
        else{
            return false;
        }
    }
    @Override
    public String toString(){
        return super.toString() + ",  grade: " + getGrade() + ",  point
average: " + getPointAverage();
    }
}
```

این کلاس تمام مشخصه ها و متد های کلاس والدش را دارد ، علاوه بر آن مشخصه های دیگری نیز دارد .

همچنین ما در این کلاس متد `toString` را مجددا `override` کردیم ، هنگامی که یک شی از کلاس `Student` ساخته میشود و سپس تلاش میکنیم آن شی را چاپ کنیم ، متد `println()` در کلاس ما به دنبال متد `toString` میگردد ، اگر در کلاس والد به دنبال این متد میگردد ، و در نهایت این زنجیره به کلاسی به نام `Object` در جاوا منتهی میشود که همه کلاس های ما به صورت مستقیم یا غیر مستقیم از آن ارث بری میکنند و `toString` آن کلاس را فراخوانی میکند.

در اینجا به دلیل اینکه کلاس `Student` دارای مشخصه های بیشتری نسبت به کلاس والد خود بود ، و ما قصد داشتیم آنها را هنگام چاپ یک شئ از کلاس در خروجی نمایش دهیم ، متد `toString` را `override` کردیم .

همچنین برای مشخصه های جدید کلاسمان متد های getter و setter تعریف کردیم .

در یکی از این متد ها که مربوط به دریافت معدل دانشجو بود با استفاده از ساختار کنترلی معدل را کنترل کردیم ، به نحوی که کاربر نمیتواند هنگام وارد کردن معدل عددی بیشتر از `20` وارد کند .

نکته دیگر که میتوانیم آن را بیاموزیم این است که ما میتوانیم ساختار کنترلی `if` را هنگامی که یک خط کد درون آن قرار میگیرد بدون براکت بنویسیم .

```
if(pointAverage <= 20)
    this.pointAverage = pointAverage;
```

همچنین در متد سازنده این کلاس برای جلوگیری از تکرار شدن این ساختار کنترلی و نوشتن کد اضافه از متد `setPointAverage()` برای مقدار دهی کردن معدل استفاده کردیم .

```
...
public Student(String name ,String lastName ,String natonalCode ,String grade
,double pointAverage){

    super(name, lastName, natonalCode);
    this.grade = grade;
    this.setPointAverage(pointAverage);

}
...
```

اکنون در فایل مربوط به متد `main` برنامه از هر کدام برای تست کردن دو مثال میزنیم.

```java
public class Main {
    public static void main(String[] args) {

        Person person1 = new Person();
        Person person2 = new Person("Javad", "Abasi", "1234567890");
        System.out.println(person1 + "\n" + person2);

        Student student1 = new Student();
        Student student2 = new Student("Ehsan", "Moradi", "1234567890",
"fourth", 13.73);
        System.out.println(student1 + "\n" + student2 + ", pass: " +
student2.passCheck());
    }
}
```

```
name: empty,  lastname: empty,  national code: 0000000000
name: Javad,  lastname: Abasi,  national code: 1234567890
name: empty,  lastname: empty,  national code: 0000000000,  grade: zero,
point average: 0.0
name: Ehsan,  lastname: Moradi,  national code: 1234567890,  grade: fourth,
point average: 13.73, pass: true
```

در اینجا دو شخص متفاوت ، یکی با مقدار دهی اولیه و دیگری بدون مقدار دهی اولیه از روی کلاس Person ساخته شد و در خروجی هم چاپ شد ، و پایین تر از آن دو دانشجو یکی با مقدار دهی و دیگری بدون مقدار دهی از روی کلاس های مربوطه ساخته شد و در خروجی چاپ شد ، در آخر متد passCheck() را برای دانش آموز دوم فراخوانی کردیم و نتیجه را مشاهده کردیم.

متد passCheck() متدی است که ورودی ندارد و درون شئ ما برسی میکند و در صورتی که معدل دانشجو بالای 10 باشد خروجی true را برمیگرداند و در غیر این صورت false را برمیگرداند .

*پس از ارث بری مفهومی دیگر را در جاوا برسی می‌کنیم به نام*

## polymorphism یا چند ریختی :

**چند ریختی (Polymorphism) :** در بخش **ارث بری** گفتیم که کلاس‌ها می‌توانند از کلاس دیگری مشتق شوند و خصوصیات و متد های آنها را داشته باشند. اما گاهی اوقات می‌خواهیم این متد ها و خصوصیات، طور دیگری رفتار کنند یا مقادیر دیگری داشته باشند. قابلیت **چند ریختی یا پلی‌مورفیسم** به ما این اجازه را می‌دهد تا این تغییرات را به سادگی بوجود بیاوریم. بدین منظور در کلاس فرزند، متدی که قصد تغییر آن را داریم، مجددا می‌نویسیم و بدنه‌ی آن متد را در **کلاس فرزند** به صورت دلخواه بازنویسی می‌کنیم.

```java
public class X {
    public void hello() { //Base class method
        System.out.println ("hello, I'm hello method from class X");
    }
}
```

```java
public class Y extends X {
    @Override
    public void hello() { //Derived Class method
        System.out.println("hello, I'm hello method from class Y");
    }
}
```

```java
public class Z extends X {
    @Override
    public void hello() { //Derived Class method
        System.out.println ("hello, I'm hello method from class Z");
    }
}
```

```java
public class Main {
    public static void main (String args []) {
        X obj1 = new X();
        X obj2 = new Y();
        X obj3 = new Z();

        obj1.hello();
        obj2.hello();
        obj3.hello();
    }
}
```

```
hello, I'm hello method from class X
hello, I'm hello method from class Y
hello, I'm hello method from class Z
```

این یک کد ساده با 4 کلاس میباشد ، یکی از کلاس ها به نام Main که متد اصلی برنامه درون آن قرار دارد . و سه کلاس دیگر یک کلاس به نام X و دو کلاس دیگر به نام Y و Z که هر دوی این کلاس ها از X ارث بری میکنند .

در کلاس X یک متد به نام hello() وجود دارد که کارش چاپ کردن پیغام سلام است و در اخر پیغام اینکه این سلام بر اثر فراخوانی کدام متد از کدام کلاس است را مینویسد .

در دو کلاس دیگر که از این کلاس ارث بری میکنند هم متد hello وجود دارد اما با تغییرات جزیی که پس از سلام کردن نام کلاس خود را بنویسند . به این منظور ما این متد را Override میکنیم تا با تغییرات موردنظرمان تطبیق دهیم.

در واقع متد سلام کردن در هر سه کلاس وجود دارد اما به شکل های مختلفی ، آنها تفاوت هایی با هم دارند ، این تفاوت ها میتوانند خیلی جزیی مانند این مثال که برای درک این موضوع است باشند و یا میتوانند یک تفاوت محسوس تر و بزرگ تر باشد .

در مثال های اخر این فصل پس از بررسی مطالب باقی مانده همه این مواردی که یادگرفتیم را در قالب یک مثال جامع تر مرور خواهیم کرد

اما نکته دیگری که در این کد ممکن است نظر شما را به خودش جلب کن :

```
X obj1 = new X();
X obj2 = new Y();
X obj3 = new Z();
```

چرا ما از روی کلاس های Y و Z شئ ساختیم ولی آنها را درون متغیر هایی از جنس X قرار دادیم ؟؟

پاسخ این سوال این است که هر شئ از کلاس مشتق شده مانند Y و Z که از X مشتق شده باشند وجود داشته باشد ، علاوه بر اینکه یک شی از کلاس Y و Z هستند ، میتوانیم آن یک شی از کلاس X هم در نظر بگیریم.

در مثال بالا obj2 یک شئ کلاس Y است ، ولی چون Y خود کلاس مشتق شده از X می باشد میتوانیم obj2 را یک شئ از کلاس X هم در نظر بگیریم .

ولی در این صورت مشکلاتی رخ میدهند .

اکنون به توضیح آن میپردازیم :

بیاید مثال را کمی گسترش بدهیم ، من به هر کدام از کلاس ها یک متد اختصاص میدهم که مختص خود کلاس ها باشند با نام های مشخص :

```
public class X {
    public void hello() { //Base class method
        System.out.println ("hello, I'm hello method from class X");
    }

    public void xMethod() {
        System.out.println ("hello, I'm hello method from class X");
    }
}
```

```
public class Y extends X {
@Override
    public void hello() { //Derived Class method
        System.out.println ("hello, I'm hello method from class Y");
    }

    public void yMethod() {
        System.out.println ("hello, I'm hello method from class X");
    }
}
```

```
public class Z {
@Override
    public void hello() { //Derived Class method
        System.out.println ("hello, I'm hello method from class Z");
    }

    public void zMethod() {
```

```
        System.out.println ("hello, I'm hello method from class X");
    }
}
```

حال بیاید و سعی کنیم متد های منحصر به فرد کلاس ها را اجرا کنیم :

```
public class Main {
        public static void main (String args []) {
            X obj1 = new X();
            X obj2 = new Y();
            X obj3 = new Z();

            obj1.hello();
            obj2.hello();
            obj3.hello();

            obj1.xMethod();
            obj2.yMethod();
            obj3.zMethod();

        }
    }
```

```
java: cannot find symbol
 symbol:   method yMethod()
 location: variable obj2 of type X
java: cannot find symbol
 symbol:   method zMethod()
 location: variable obj3 of type X
```

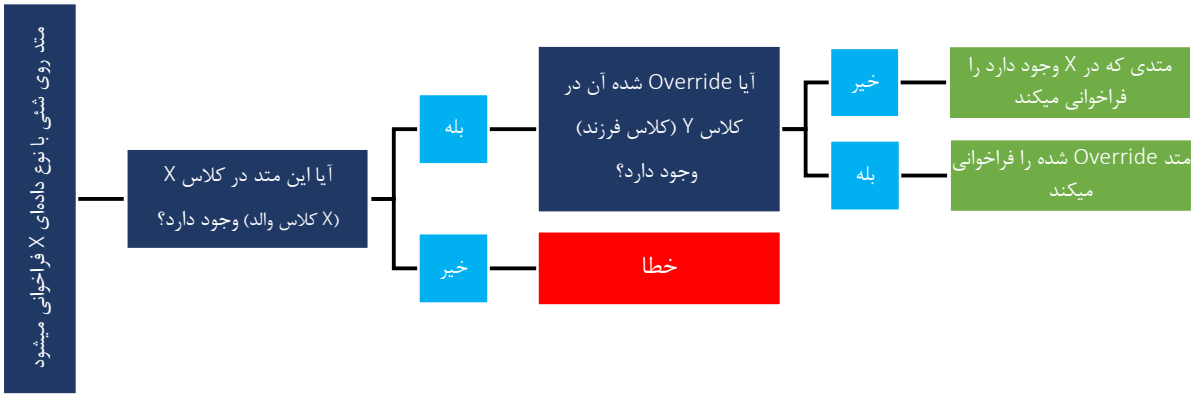
در خروجی به این صورت به ما ارور میدهد ، اما مشکل از چیست؟ دقت کنید به این قسمت

```
X obj1 = new X();
X obj2 = new Y();
X obj3 = new Z();
```

ما هنگام ساختن شی از روی کلاس ها جنس اشیا را X قرار دادیم ، و هنگامی که روی این اشیاء متدی فراخوانی میشود ، جاوا درون X چنین متد هایی جستجو میکند :

- در صورتی که وجود داشته باشد ، در مرحله بعد درون کلاس فرزند Override شده این متد را جستجو میکند  اگر وجود داشته باشد Override شده استفاده میکند و Override شده وجود نداشته باشد از متدی که در X وجود داشت استفاده میکند.

- در صورتی که وجود نداشته باشد به دنبال آن در کلاس های فرزند نمیگردد و با خطا مواجه میشویم.




**در درس بعدی که درباره انتزاع میباشد با یک مثال کاربرد این مورد را برسی خواهیم کرد.**

حال که این متد ها درون X وجود ندارند برای رفع این خطا ما باید تغییر نوع داده ای مناسبی را انجام دهیم تا اجرا شوند :

به این صورت :

```
obj1.xMethod();
((Y) obj2).yMethod();
((Z) obj3).zMethod();
```

اکنون در صورتی که از برنامه خروجی بگیریم به این صورت خواهد بود :

```
hello, I'm hello method from class X
hello, I'm hello method from class Y
hello, I'm hello method from class Z
This is a method exclusive to the X class
This is a method exclusive to the Y class
This is a method exclusive to the Z class
```

## انتزاع (Abstraction) :

**انتزاع (Abstraction) :** یکی از مفاهیم مهم در **برنامه نویسی شی گرا**، بحث انتزاع است .**مفهوم انتزاع** ممکن است کمی گنگ باشد. در برنامه نویسی، **کلاس های Abstract** برای ساخت یک اساس و مفهوم کلی ساخته می‌شوند، برخلاف مثال‌هایی از کلاس‌هایی که بالاتر ذکر کردیم، کلاس Abstract هیچ پیاده سازی‌ای ندارد و تنها تعاریف در آنها قرار گرفته‌اند. به همین دلیل **نمونه سازی یا ساخت شی از کلاس های Abstract ممکن نیست** و در عوض کلاس‌ها می‌بایست از یک کلاس Abstract مشتق بشوند.

- برای اینکه یک کلاس یا یک متد را به حالت انتزاعی در بیاوریم از کلمه کلیدی `abstract` استفاده میکنیم.
- ما نمیتوانیم از روی کلاس انتزاعی شی بسازیم .
- متد های کلاس انتزاعی میتوانند هم متد انتزاعی باشند و هم غیر انتزاعی و از آنها ارث بری خواهد شد.
- همه کلاس ها و متد های انتزاعی `public` هستند
- متد های انتزاعی را پیاده سازی نمیکنیم ، آنها باید در کلاس های فرزند پیاده سازی شوند . در اینجا فقط بدنه آن را مینویسیم
- در کلاس های انتزاعی متد سازنده **فقط** برای استفاده زیر کلاس ها تعریف میشود .

به مثال توجه کنید :

```
public abstract class Shape{
    private String color;

    public  Shape(){
        this("black");
    }
    public  Shape(String color){
        this.color = color;
    }

    public String GetColor(){
        return color;
    }

    public abstract double area();

    protected void setColor(String color) {
        this.color = color;
    }
    protected String getColor(){
        return color;
    }
    @Override
    public String toString(){

        return String.format("Shape color : %s",this.color);
    }
}
```

```
public class Circle extends Shape{
    private double radios;
    private final double PI = 3.14;
    public Circle(){
        super();
        this.radios=1;
    }
    public Circle(String color , double radios){
        super(color);
        this.radios = radios;
    }


    @Override
    public void setColor(String color){
        super.setColor(color);
    }
    @Override
```

```
    public String getColor(){
        return super.getColor();
    }

    @Override
    public double area() {
        return PI * radios * radios;
    }

    @Override
    public String toString(){
        return String.format("Radios : %.2f  |  Color : %s  | Area : %.2f
",this.radios,super.getColor(),this.area());
    }
}
```

```
class Rectangle extends Shape {
    private double length;
    private double width;

    public Rectangle() {
        super();
        this.length = 0;
        this.width = 0;
    }
    public Rectangle(String color, double length, double width) {
        super(color);
        this.length = length;
        this.width = width;
    }


    @Override
    public void setColor(String color){
        super.setColor(color);
    }
    @Override
    public String getColor(){
        return super.getColor();
    }

    @Override
    public double area() {
        return length * width;
    }

    public double perimeter() {
        return (length + width)*2;
```

```
    }

    @Override
    public String toString(){
        return String.format("Length : %.2f  | Width : %.2f | Color : %s  |
Area : %.2f ",this.length,this.width,super.getColor(),this.area());
    }
}
```

```
public class Main {
    public static void main (String[] args) {
        Shape rectangle1 = new Rectangle("red",2,4);
        Shape rectangle2 = new Rectangle("red",6,2);
        Shape circle1 = new Circle("blue",2);

        System.out.println(rectangle1);
        System.out.println(rectangle2);
        System.out.println(circle1);

        System.out.println(circle1.GetColor());
        circle1.setColor("cyan");
        System.out.println(circle1.GetColor());

        System.out.println(((Rectangle) rectangle1).perimeter());

    }
}
```

```
Length : 2.00  | Width : 4.00 | Color : red  | Area : 8.00
Length : 6.00  | Width : 2.00 | Color : red  | Area : 12.00
Radios : 2.0 | Color : blue | Area : 12.56
blue
cyan
12.0
```

در فایل اول (کلاس Shape) یک کلاس انتزاعی به نام Shape تعریف کردیم ، از این کلاس قرار نیست نمونه ای ساخته شود و فقط قرار است از بدنه و فرمت آن استفاده شود برای عدم تعریف مفاهیم تکراری .

در قسمت های بعدی کلاس Circle و Rectangle را تعریف کردیم که از کلاس Shape ارث بری میکنند .

وقتی از یک کلاس انتزاعی ارث بری میکنیم ملزم میشویم همه متد های انتزاعی آن را تعریف و Override کنیم ، ولی همچنان بقیه خواص مانند ارث بری عادی هستند .

متد های getter و setter را هم برای متد ها قرار میدهیم .(میتوانستیم متد های بیشتری را قرار دهیم ، اما هدف اینجا درک مفهوم انتزاع است)

در مابقی کد مورد جدیدی وجود ندارد و همه مواردی هستند که تا کنون برسی کردیم.

**String.format() احتمالا چیز جدیدی است برای شما ، این متد دقیقا مانند printf عمل میکند با این تفاوت که مقادیری که به آن فرستاده میشود را تبدیل برمیگرداند و چاپ نمیکند .**

به این خط دقت کنید :

```
System.out.println(((Rectangle) rectangle1).perimeter());
```

دلیل این تغییر نوع داده این است که ما روی rectangle1 که یک مستطیل است ولی از نوع داده ای Shape است متدی را فراخوانی کردیم ولی این متد درون کلاس Shape وجود ندارد و ما میدانیم اگر متد در کلاس Shape که کلاس والد محسوب میشود وجود نداشته باشد به خطا برمیخوریم .

که البته این موضوع را در قسمت های قبل برسی کردیم.

اکنون تصور کنید ما یک آرایه میسازیم و میخواهیم تعدادی مستطیل و دایره مختلف درون آن ذخیره کنیم . در این هنگام ما میتوانیم یک آرایه از نوع Shape ایجاد کنیم و این موارد را درون آن ذخیره کنیم.

پس من متد اصلی برنامه را به این صورت تغییر میدهم برای تعریف این آرایه:

```
public class Main {
    public static void main (String[] args) {
    Shape rectangle1 = new Rectangle("red",2,4);
    Shape rectangle2 = new Rectangle("red",6,2);
    Shape circle1 = new Circle("blue",2);

    Shape[] Array = {rectangle1, rectangle2, circle1};

    for(Shape i : Array){
        System.out.println(i.area());
    }
    }
}
```

```
8.0
12.0
12.56
```

ما یک آرایه تعریف کردیم که المنت های این آرایه از نوع Shape هستند سپس بر روی آنها متد ()area را فراخوانی کردیم ، و برای با توجه به مواردی که در مثال مربوط به Polymorphism در صفحه [ ] اموختیم ، چون متد ()area درون Shape وجود داشت و سپس Override شده آن درون کلاس های Rectangle و Circle وجود داشت ، پس Override شده ها فراخوانی شدند.

در ادامه میخواهیم درباره عملگر instanceof و ()equals بحث کنیم .

علمگر instanceof بین یک کلاس و یک شی قرار میگیرد :

```
(Object) instanceof (Class)
```

در این حالت در صورتی که شئ متعلق به کلاس باشد true و در غیر اینصورت false را برمیگرداند .

با توجه به مثال قبل به این کد توجه کنید :

```
public class Main {
    public static void main (String[] args) {
    Shape rectangle1 = new Rectangle("red",2,4);
    Shape rectangle2 = new Rectangle("red",6,2);
    Shape circle1 = new Circle("blue",2);

    System.out.printf("rectangle1 is instance of Shape : %b %n",rectangle1
instanceof Shape);
    System.out.printf("circle1 is instance of Rectangle : %b %n",circle1
instanceof Rectangle);
    System.out.printf("circle1 is instance of Circle : %b %n",circle1
instanceof Circle);
    System.out.printf("rectangle2 is instance of Object : %b %n",rectangle2
instanceof Circle);
    System.out.printf("circle1 instanceof Object : %b %n",rectangle1
instanceof Object);
    }
}
```

```
rectangle1 is instance of Shape : true
circle1 is instance of Rectangle : false
circle1 is instance of Circle : true
rectangle2 is instance of Object : false
circle1 instanceof Object : true
```

نکته ای که در این قسمت حائز اهمیت است این است که همه اشیائی که میسازیم یک نمونه از کلاس Object محسوب میشوند .

متد ()equals برای مقایسه بین دو شی یا نوع دادهغیرابتدایی در جاوا استفاده میشود .

استفاده از آن به این صورت است :

```
objectA.equals(objectB)
```

در صورتی که مرجع آدرس های آنها یکی باشد `true` و در غیر این صورت `false` را برمیگرداند .

```
public class Main {
    public static void main (String[] args) {

    Rectangle r1 = new Rectangle("red",2,4);
    Rectangle r2 = new Rectangle("red",2,4);

    System.out.println(r1.equals(r2));

    }
}
```

```
false
```

در این مثال با وجود اینکه هر دو شئ `r1` و `r2` دارای مقادیر ورودی یکسان هستند اما نتیجه مقایسه این دو با استفاده `equals()` در خروجی `false` است .

با وجود اینکه هر دو این اشیاء دارای مقادیر یکسان هستند اما جاوا آنها را در نقاط مختلف حافظه ذخیره میکند .

حال متغیر `r3` را از نوع `Rectangle` تعریف میکنیم و مقدار `r2` را درون آن قرار میدهیم .

```
public class Main {
    public static void main (String[] args) {

    Rectangle r1 = new Rectangle("red",2,4);
    Rectangle r2 = new Rectangle("red",2,4);
    Rectangle r3 = r2;

    System.out.println(r1.equals(r2));
    System.out.println(r3.equals(r2));

    }
}
```

```
false
true
```

چرا پاسخ `true` بود ؟ به دلیل اینکه مرجع آدرس متغیر `r3` در این کد همان مرجع آدرس `r2` است . و وقتی مرجع آدرس ها با هم برابر باشند `equals()` مقدار `true` را بر میگرداند.

به خاطر بیاورید وقتی که از عملگر == بین نوع داده ای مرجع (غیر ابتدایی) استفاده میکردیم مرجع آدرس در حافظه را مقایسه میکرد :

```
 System.out.println(r1.equals(r2));
System.out.println(r3.equals(r2));
System.out.println(r1==r2);
System.out.println(r3==r2);
```

```
false
true
false
true
```

این متد درون کلاس Object قرار دارد ، و میدانیم که همه کلاس هایی که میسازیم از کلاس Object به صورت مستقیم یا غیر مستقیم ارث بری میکند ، و این متد هم از آن کلاس به ارث رسیده .

اکنون ما آن را به نحوی Override میکنیم که به جای مرجع آدرس ، مقادیر را با هم مقایسه کند .

نکته : در کلاس String این متد Override شده است ، و به صورت پیش فرض مقادیر را با هم مقایسه میکند .

```
class Rectangle extends Shape {

    …

    @Override
    public boolean equals(Object obj) {
        if (!(obj instanceof Rectangle)){
            System.out.println("Error :input must be an object belonging to
Rectangle class");
            System.exit(0);
        }
        return (((Rectangle) obj).length == this.length) && (((Rectangle)
obj).width == this.width);
    }
    …

}
```

برای جلوگیری از شلوغی بیش از حد فقط قسمتی که به کد قبل اضافه کردیم را در این قسمت قرار دادیم .

در این کد متد equals() به این صورت عمل میکند که یک شی به عنوان پارامتر ورودی دریافت میکند و در صورتی که شئ یک نمونه از کلاس Rectangle نباشد پیغام اخطار برای کاربر نمایش میدهد و سپس برنامه را به پایان میرساند .

در غیر این صورت در خروجی طول و عرض ها را با هم مقایسه میکند و در صورتی که دو به دو با هم برابر باشند `true` و در غیر این صورت `false` را نمایش میدهد .

```
Rectangle r1 = new Rectangle("red",2,4);
Rectangle r2 = new Rectangle("red",2,4);
Rectangle r3 = r2;

System.out.println(r1.equals(r2));
System.out.println(r3.equals(r2));
```

```
true
true
```

اکنون خروجی مقادیر اشیاء را با هم مقایسه کرد و خروجی به این صورت شد .

**نکته** : ساختن تعداد زیاد متغیر ها و نمونه ها باعث اشغال شدن حافظه و ممکن است روی عملکرد کلی برنامه ما تاثیر منفی بگذارد .

برای خالی کردن حافظه از متغیر هایی که به آنها نیاز نداریم کافیست متغیر ها را برابر با `null` قرار بدهیم :

```
public class Main {
    public static void main (String[] args) {

    Rectangle rec = new Rectangle("red",2,4);
    String str = "hello";

    rec = null;
    str = null;

    }
}
```

**دقت کنید که این روش فقط برای حذف کردن داده های مرجع (غیر ابتدایی) کار میکند .**

## یکم پیشرفته تر :

### ارث بری سلسله مراتبی و ارث بری چندگانه

**ارث بری سلسله مراتبی :** در ارث بری سلسله مراتبی چند کلاس به صورت متوالی از یکدیگر ارث بری میکند .

به شکل توجه کنید :


در این مورد کلاس C از کلاس B ارث بری کرده و کلاس B از کلاس A

کلاس C تمام خصوصیات کلاس B و A را دارد ، ممکن است برخی از آنها را تغییر داده باشد و Override شده باشد .

کلاس B تمام خصوصیات A را دارد ، ممکن است برخی از آنها را تغییر داده باشد و Override شده باشد.

در این مثال هر شی از کلاس C را میتوانیم به این دو صورت تعریف کنیم

```
A object = new C();
B object = new C();
```

به این دلیل که هم A و هم B کلاس والد C محسوب میشوند .

نکته : برعکس مورد بالا امکان پذیر نیست ، یعنی نمیتوانیم یک شی از کلاس A بسازیم با نوع B یا C :

```
C object = new A();
B object = new A();
```

**ارث بری چندگانه :**

جاوا از ارث بری چند گانه استفاده نمیکند ، اما مفهوم interfaces به ما در این راه کمک میکند .

interface ها شبیه به کلاس های انتزاعی هستند ، نمیتوان از روی آنها شئ ساخت ، به همین خاطر متد سازنده ندارند .

تفاوت آنها با کلاس های انتزاعی این است که میتوانیم با استفاده از رابط ها جایگزینی برای ارث بری چند گانه پیدا کنیم ،

هستند و کلاس‌هایی که آن را پیاده سازی میکنیم باید public در رابط ها همه متدهایی که تعریف میشوند به صورت پیش‌فرض
هستند .final و static و public حتما آنها را پیاده سازی کنیم . مشخصه ها و متغیر ها در رابط ها همزمان

```
public interface Interface_1 {
    public static final double PI=3.14;
    public void hello();
    public void bye();
}
```

```
public class MyClass implements Interface_1{

    @Override
    public void hello() {
```

```
        System.out.println("hello");
    }
    @Override
    public void bye() {
        System.out.println("bye");
    }


}
```

```
public class Main {
    public static void main(String[] args) {

        MyClass test = new MyClass();

        System.out.println(test.PI);
        test.hello();
        test.bye();

    }
}
```

```
3.14
hello
bye
```

در این مثال ما یک رابط تعریف کردیم با دو متد . در رابط ها همه متد ها انتزاعی هستند ، مگر اینکه به صورت `default` تعریف شوند. اگر در این حالت تعریف شوند نیازی به تعریف آن در کلاس هایی که از این رابط استفاده کردند نیست ، با این حال میتوانیم آن را `Override` کنیم .

```
public interface Interface_1 {
    public static final double PI=3.14;
    public default void hello(){
        System.out.println("hello from interface_1");
    };
    public void bye();
}
```

```
public class MyClass implements Interface_1{


    @Override
```

```
    public void bye() {
        System.out.println("bye");
    }


}
```

```
public class Main {
    public static void main(String[] args) {

        MyClass test = new MyClass();

        System.out.println(test.PI);
        test.hello();
        test.bye();

    }
}
```

```
3.14
hello from interface_1
bye
```

ما میتوانیم چند رابط تعریف کنیم و کلاس در کلاس هایمان آن را پیاده کنیم .

توجه :در کلاس ها و ارث بری از اصطلاح ارث بردن استفاده میکردیم ، در بخش رابط ها از اصطلاح پیاده کردن استفاده میکنیم.

```
public interface Interface_1 {
    public void in1();
}
```

```
public interface Interface_2 {
    public void in2();
}
```

```
public class MyClass implements Interface_1,Interface_2{

    @Override
    public void in1() {
        System.out.println("this is from Interface 1");
    }

    @Override
    public void in2() {
```

```
        System.out.println("this is from Interface 2");


    }
}
```

```
public class Main {
    public static void main(String[] args) {

        MyClass test = new MyClass();

        test.in1();
        test.in2();

    }
}
```

```
this is from Interface 1
this is from Interface 2
```

## Composition یا ترکیب بندی :

استفاده کردن اشیاء یک کلاس در کلاس دیگر را Composition مینامیم ، اینکار را به کرات انجام داده ایم تا به اینجا ، برای مثال وقتی که درون یک کلاس یک مشخصه را با نوع داده ای `String` تعریف میکنیم ، در واقع ما از یک شئ کلاس `String` در کلاسمان استفاده کردیم .

در این مثال یک کلاس به نام `NameAndAddress` داریم که درون خود نام ، آدرس و کد پستی یک مکان را نگهداری میکند .

سپس درون یک فایل کلاس دیگر به نام School دو مشخصه تعریف میکنیم ، یکی از مشخصه ها ، مشخصات مدرسه که همان نام ، آدرس و کد پستی است میباشد و دیگری تعداد ثبت نام های این مدرسه است .

و در بدنه متد اصلی برنامه یک شئ از کلاس `School` ساختیم که به عنوان آرگومان باید نام ، آدرس ، کد پستی و تعداد ثبت نامی های مدرسه را به کلاس برای ساخت شی جدید ارسال کنیم . کلاس تعداد ثبت نامی ها را در یک متغیر ثبت میکند و مشخصات مدرسه را در یک متغیر دیگر (که این متغیر از نوع `NameAndAddress` می باشد و متد سازنده کلاس `School` با این سه مشخصه یک شئ جدید از کلاس `NameAndAddress` میسازد و در خودش ذخیره میکند.)

```
public class NameAndAddress {
    private String name;
    private String address;
    private String zipCode;
    public NameAndAddress(String name , String address,String zipCode){
        this.name = name;
        this.address = address;
        this.zipCode = zipCode;
    }

    public void display(){
```

```
        System.out.printf("Name : %s | Address : %s | Zipcode :
%s",this.name,this.address,this.zipCode);
    }

}
```

```
public class School {
    private NameAndAddress nameAdd;
    private int enrollment;
    public School(String name,String address,String zipCode,int enrollment){
        this.nameAdd = new NameAndAddress(name,address,zipCode);
        this.enrollment = enrollment;
    }

    public void display(){
        System.out.print("This school information");
        nameAdd.display();
        System.out.println(" | Enrollment is : "+enrollment);

    }

}
```

```
public class SchoolDemo {
    public static void main(String[] args) {
        School mySchool = new School("name","address","60618",350);
        mySchool.display();
    }
}
```

```
This school informationName : name | Address : address | Zipcode : 60618 |
Enrollment is : 350
```