My students contributed in creating these lecture notes. I appreciate there efforts but I didn't reviewed them so they are neither guaranteed to be free from errors nor to be complete.

Hamid M.Gholizadeh

Object Oriented Programing

Core java

طراحی سیستم های شی گراء با جاوا

استاد:

مهندس قلی زاده

تهیه و تنظیم:

امیر مشهدی

مهدی مهری

موسسه آموزش عالی غیرانتفاعی-غیردولتی سراج تبریز

نیمسال اول ۸۹-۸۸

طراحی سیستم

موسسه آموزش عالی
غیرانتفاعی-غیر دولتی
سراج تبریز

S2Ma

فهرست عناوین

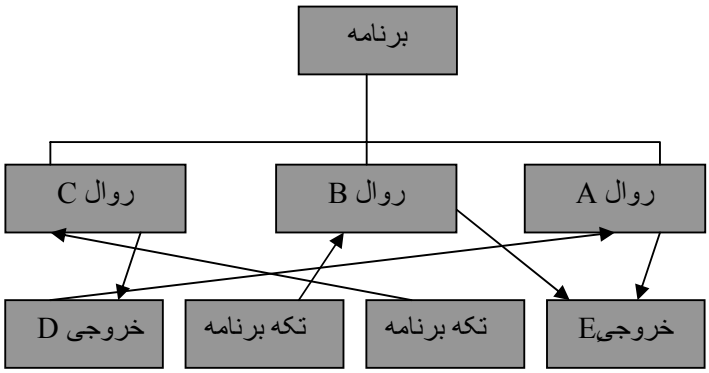
ساختار زبان های برنامه نویسی روال گونه ۵
ساختار زبان های برنامه نویسی روال گونه ۵
ساختار برنامه ها در زبانهای شی گراء: ۵
فرق جاوا با دیگر زبانها: ۵
خصوصیت زبان جاوا: ۵
مراحل ایجاد یک برنامه : ٦
خطاهای برنامه نویسی: ٦
محیط های توسعه ی یکپارچه: ٦
محیط های توسعه ی یک پارچه ی جاوا : ٦
JDK (Java Development Kit) ۷
JRE (Java Runtime Enviroment) ۷
شیوهی نام گذاری کلاسها در جاوا : ۷
تعریف متغییر در جاوا : ۷
انواع داده ها و مقدار فضای مورد نیاز آن ۸
جدول کاراکتر های خاص: ۸
نحوه تعریف مقادیر ثابت در جاوا (Constants) : ۸
قانون نامگذاری : ۸
نحوه تعریف : ۸
عملگرها (Operators) : ۹
۱ ـ عملگرهای محاسباتی ۹
۲ ـ عملگرهای منطقی ۹
عبارت شرطی : ۱۰
عملگرهای بیتی : ۱۱
عملگرهای << (شیفت به راست) و >> (شیفت به چپ) : ۱۱
توابع ریاضی و ثوابت ریاضی : ۱۱
تبدیلات بین نوع های داده ای عددی : ۱۲
تبدیل صریح و ضمنی میان انواع داده ها : ۱۲
تقدم عملگرها : ۱۳
نوع داده شمارشی : ۱۳
قانون نامگذاری : ۱٤
ادغام رشته ها : ۱٤
مقایسه دو رشته : ۱٤
کلاس string builder : ۱۵
نحوه گرفتن ورودی از کاربر : ۱۵
قالب بندی خروجی در جاوا : ۱۵
مفهوم بلوک در جاوا : ۱٦
دستورات شرطی: ۱٦
دستور if ۱٦

دستور if / else ۱۶
دستور if / else if/ else ۱۶
دستور while : ۱۶
دستور do – while: ۱۶
دستور for:..... ۱۷
ساختار switch / case:..... ۱۷
دستور break و continue : ۱۸
آرایه ها در جاوا: ۱۸
نحوه ی تعریف آرایه: ۱۸
توابع و خصوصیات مورد استفاده در ارتباط با آرایه :..... ۱۸
کلاسها و اشیاء..... ۱۹
شناسه ی شیء:object identity..... ۱۹
نحوه ی تعریف کلاس در جاوا : ۱۹
نحوه ی ساختن یک شی (نمونه) از یک کلاس: ۱۹
نحوه ی دسترسی به خصوصیات شی : ۲۰
نحوه ی دسترسی به متد شیء:..... ۲۰
توابع سازنده :cunstractors ۲۰
ارتباط aggregation (شامل بودن):..... ۲۰
نحوه ی تعریف متد ها : ۲۰
کلمه ی کلیدی static:..... ۲۱
ارث بری در جاوا inheritance ۲۱
کلمه ی کلیدی This:..... ۲۲
کلمه ی کلیدی super:..... ۲۲
تابع سازنده کلاس line ۲۲
تعیین سطوح دسترسی AccessModifiers..... ۲۳
getTerها و setTer ها در جاوا : ۲۴
دلایل استفاده از سطح دسترسی Private و کاربرد Getter ها و Setter ها ۲۴
{سربارگذاری توابع (Method Overloading) : ۲۵
Method Overriding :..... ۲۶
پکیجها (Packages) :..... ۲۷
کلمه کلیدی Import :..... ۲۸
توابع abstract : ۲۸
interface ها در جاوا :..... ۲۹
Polymorphism (چند ریختگی) :..... ۲۹
الگوهای طراحی (Design Patterns) : ۳۱
۱ ـ الگوی یگانه (Singleton Pattern):..... ۳۱
علت تعریف شدن شیء Singleton به صور استاتیک:..... ۳۲
نحوه ی نمایش interface ها در UML : ۳۳

نحوه ی نمایش Access modifier ها در UML: ۳۳
اگوی کارخانه یا Factory pattern: ۳٤
نحوه کار با واسط های گرافیکی در جاوا : ۳٤
Object JFrame : ۳٤
سلسله مراتب اشیاء در فن آوری Swing..... ۳۵
ساختار یک Jframe در جاوا : ۳۵
ترتیب قرار گرفتن object ها در Jframe به ترتیب زیر است..... ۳۵
کلاس graphic2D..... ۳۸
مدیریت رویدادها Event Handling : ٤۰
Event Listener (گوش دهنده به رویداد) : ٤۰
نحوه ی تعریف و ایجاد یک شیء بدون تعریف صریح کلاس آن: ٤۲
رویدادهای مربوط به پنجره ها (Window) ها : ٤۳

ساختار زبان های برنامه نویسی روال[1] گونه

در این نوع برنامه ها یک برنام کارکردها یا functions های خود را میان تعداد ی روال یا تابع [2] تقسیم مـس کنـد. برنامـه یـک نقطه ی آغاز دارد و سپس برای انجام وظایف برنامه روالهای گوناگون داخل برنامه فراخوانی می شود.



چرا شی گرائی؟

ساختار برنامه ها در زبانهای شی گراء۳:

در این نوع برنامه ها یک برنامه از مجموعه ای از اشیاء تشکیل شده است که این اشیاء در تامل و همکاری با یکدیگر کار کرده یا عاملیت[4] آن برنامه را انجام می دهند.

فرق جاوا با دیگر زبانها:

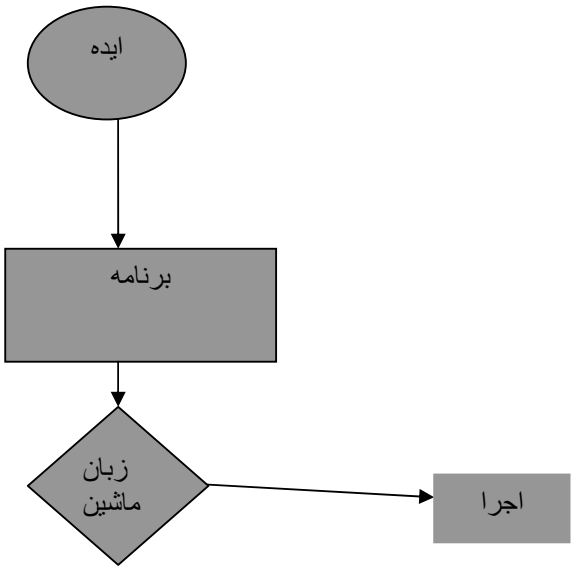
خصوصیت زبان جاوا:

یک زبان برنامه نویسی شی گراء است که ویژگی بارز آن استقلال برنامه های نوشته شده با استفاده از این زبان از بسته ی است که برنامه می خواهد بر روی آن اجرا شود. به این خصوصیت این زبان خصوصیت مـستقل بـودن از بـسته یـا platform inde pendancy گفته می شود.که دلیل آن تمامی برنامه ها پـس از compile تبـدیل بـه byte code مـی شـوند و byte code توسـط JVM۵ در هر platform به زبان ماشین آن platform تبدیل می شود

- ویژگی جاوا به گونه ای است که در تمامی platform ها قابل اجرا است.

[1] Procedured
[2] Function
[3] Object Oriented
[4] Functionality

مراحل ایجاد یک برنامه :



خطاهای برنامه نویسی:

Syntax error : خطاهای نحوی ، خطاهایی هستند که به سبب وجود ایراد در ساختار برنامه نوشته شده رخ می دهد.

مثال /. عدو وجود ; در انتهای یک دستور جاوا

Logical error : خطاهایی هستند که در عملکرد برنامه وجود دارند. یعنی برنامه به خوبی اجرا می شود ولی آن انتظاری که ما داریم بر آورد نمی شود. مثال / . برنامه می بایست میانگین نمرات گروه کامپیوتر را حساب کند به جای آن میانگین نمرات کل دانشکده را محاسبه می کند.

- عناصر مورد استفاده در برنامه نویسی:
- ۱- ویرایش گر ها (editors)
- ۲- مترجم ها (compilers)
- ۳- اشکال زدا ها (debuggers) : که به برنامه نویس امکان می دهد که هنگام اجرای برنامه همزمان خطی از برنامه را که دارد اجرا می شود را مشاهده کند. علاوه بر آن مقادیر متغییر ها و سایر پارامتر های برنامه را در هنگام اجرا ببیند.

محیط های توسعه ی یکپارچه[1]:

نرم افزار های هستند که در یک محیط ابزارهایی مانند ویرایش گر، کامپایلر، دیباگر و یا سایر ابزارهای کمکی را فراهم می کند. به عنوان مثال ابزارهای مدل سازی و رسم نمودارهای UML ابزارهای مهندسی معکوس و ...

مثال/ . ویژوال استودیو Visual Studio.net

محیط های توسعه ی یک پارچه ی جاوا :

Jbulder (Borland)
IntelJ (IBM)
NetBeans (Sun)
Jdevelop (oracle)
Eclipse

[1] Integrated Development Enviroment

JDK (Java Development Kit)

بسته ی نرم افزاری شامل کلاسهای از پیش تعریف شده ی جاوا

برای انجام برنامه نویسی با این زبان به همراه ماشین مجازی جاوا و کامپایلر و تعداد ی موارد دیگر است.

JRE (Java Runtime Enviroment)

یک بسته ی نرم افزاری شامل ابزارهایی جهت اجرای برنامه های جاوا است .

JDK-SE ویرایش استاندارد

JDE.J2EE جهت برنامه نویسی سازمانی

JME(J2E) جهت برنامه نویسی برای موبایل

نمونه ی یک برنامه ی جاوا :

```
public class MyApp1 {

      public  static void main(String arg[]){
            System.out.println("welcome to java!");
                                                }
                        }
```

که خروجی این برنامه به صورت زیر خواهد بود :

```
Welcome to Java!
```

شیوه‌ی نام گذاری کلاسها در جاوا :

۱- اگر نام کلاس یک کلمه ای بود حرف اول آن با حرف بزرگ و بقیه با حروف کوچک نوشته می شود.

۲- اگر بیش از یک کلمه بودف حرف اول آن حرف بزرگ و بقیه با حروف کوچک اما اول هر کلمه با حرف بزرگ شروع می شود.

۳- نام کلاسها می تواند دارای خط فاصله باشد.

مثال / .

```
class MathUtil {}
```

تعریف متغییر در جاوا :

```
<type >  <name>;
        و یا
<type > <name> = <value>;
```

مثال / .

```
Int a;
Int b = 4;
Float num = 4.17;
```

انواع داده ها و مقدار فضای مورد نیاز آن

ردیف	نوع داده	مقدار فضا (بایت)	توضیحات	مثال
1	byte	1	عددی	Byte a = 1;
2	Short	2	عددی	Short a = 126;
3	int	4	عددی	Int a = 3246;
4	long	8	عددی	Long a = 100000;
5	Float	4	اعشاری	Float a = 47.561
6	double	8	اعشاری	Double a = 675.4563
7	boolean	1	منطقی – true-false	True/false
8	char	1	یک کاراکتر	Char ch='a';
۹	string	-	کلاس از نوع رشته	String a = "welcome to java ";

جدول کاراکتر های خاص:

کاراکتر	عملکرد
\n	خط جدید
\r	ابتدای سطر
\t	معادل tab
\\	چاپ \
\'	چاپ '
\"	چاپ "
\B	Back space

نحوه تعریف مقادیر ثابت در جاوا (Constants) :

منظور از مقادیر ثابت متغیرهایی هستند که تمایل دارید مقادیر آنها در طول اجرای برنامه تغییر نکند (یعنی بعد از گرفتن یک مقدار دیگر به هیچ عنوان آن مقدار تغییر نکند) .

قانون نامگذاری :

تمامی حروف نام متغیرهای ثابت به صورت حروف بزرگ نوشته می شود .

نحوه تعریف :

با اضافه کردن کلمه کلیدی final در ابتدا تعریف متغیر و دادن مقدار به آن در هنگام تعریف انجام می گیرد .

```
public class Example2 {
public  static void main(String arg[]){
final float PI1=3.14f;
final double PI2=3.14;
```

```
final double PI3=3.14f;
final double PI4=Math.PI;
System.out.println(PI1);
System.out.println(PI2);
System.out.println(PI3);
System.out.println(PI4);
            }
      }
```

خروجی:

```
3.14
3.14
3.140000104904175
3.141592653589793
```

توضیحات (Comments) :

دو روش وجود دارد :

۱ ـ افزودن توضیحات تک سطحی با افزودن دو '/' متوالی

۲ ـ افزودن توضیحات بیش از یک سطح با قرار دادن بین دو عبارت /*...*/

```
/*This is a simple java program.
 call this file "Example3.java
 */
public class Example3 {
//Your program begins with a call to main().
public  static void main(String arg[]){
System.out.println("This is a simple Java program");

      }
}
```

خروجی:

```
This is a simple Java program
```

نکته : علاوه بر دو روش ذکر شده در بالا روش دیگری هم برای تولید مستندات برنامه به صورت خودکار وجود دارد که با قرار دادن بین دو عبارت /**...*/

نکته : در مقدار دهی اولیه عملوندها می توانیم از عبارات ریاضی هم استفاده کنیم .

مثال:

```
double w=x / y * z;
```

عملگرها (Operators) :

۱ ـ عملگرهای محاسباتی

```
+   -   /   %   +=   -=   *=   /=
```

۲ ـ عملگرهای منطقی

```
==   !=   >   <   >=   <=   !
```

خروجی اعمال عملگرهای منطقی یک مقدار منطقی است (true or false)

نکته :

در جاوا true ، true است و معادل غیر صفر ندارد . (false هم همینطور)

مثال:

```
if ( a || 1)   ,   if (a || 0)   ,  if (a && 4)   ,   if (1)
```

را می‌پذیرند.Booleanتمامی شرط‌های بالا اشتباه می باشند، زیرا همانطور که ذکر شد عملگرهای منطقی تنها عملوندهای

عبارت شرطی :

عبارت شرطی ? عبارت اول : عبارت دوم ;

مثال۱:

```
public  static void main(String arg[]){
      int b=5;
      b = true ? 1 : 0;
      System.out.println(b);

      }
```

خروجی:

```
      1
```

مثال۲:

```
public  static void main(String arg[]){
      int b=5;
      b = true ? 1 : 0.25;
      System.out.println(b);

      }
```

خروجی:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
      Type mismatch: cannot convert from double to int

      at Example3.main(Example3.java:5)
```

نسبت داده شده است .int به یک متغیر double*خطای نحوی (زمان کامپایل)، چون یک مقدار

مثال۳:

```
public  static void main(String arg[]){
      double b=0.5;
      b = true ? 1 : 0.25;
      System.out.println(b);

      }
```

خروجی:

```
1.0
```

*خطا ندارد چون اعداد صحیح را می توانیم به اعداد اعشاری نسبت دهیم اما برعکس آن امکان پذیر نیست

(چون تبدیل ضمنی اعداد صحیح به اعداد اعشاری به صورت خودکار صورت می گیرد اما برعکس آن صحیح نمی باشد)

مثال٤:

```
public  static void main(String arg[]){
      float b=0.5;
      b = true ? 1 : 0.25;
      System.out.println(b);

}
```

خروجی:

```
Type mismatch: cannot convert from double to float
      Type mismatch: cannot convert from double to float

      at Example3.main(Example3.java:4)
```

*خطای نحوی دارد چون اعدادی که صراحتا به صورت اعشاری در داخل برنامه تعریف می شوند از نوع double فرض می شوند . و چون نوع داده ای double بزرگتر از float است بنا براین نمی توان بدون تبدیل صریح نوع داده ای double را به یک متغیر float نسبت داد . (راه حل : اضافه کردن یک f بعد از مقدارهای اعشاری)

```
public  static void main(String arg[]){
      float b=0.5f;
```

```
b = true ? 1 : 0.25f;
System.out.println(b);

}
```

خروجی:

```
1.0
```

عملگرهای بیتی :

عملگرهای بیتی بر روی بیت ها عمل می کنند ، یعنی تفسیری که از عملوندهای خود دارند به صورت مجموعه ای از بیت ها است که هر کدام دارای مقادیر صفر یا یک هستند .

```
~   ^   |   &
```

مثال:

```
public  static void main(String arg[]){
      int a=1;
      a=~a;
      System.out.println(a);  }
```

خروجی:

```
-2
```

عملگرهای << (شیفت به راست) و >> (شیفت به چپ) :

مثال ۱:

```
public  static void main(String arg[]){
      int a=1;
      a=a<<3;
      System.out.println(a);  }
```

خروجی:

```
8
```

مثال ۲:

```
public  static void main(String arg[]){
      int a=128;
      a=a>>2;
      System.out.println(a);  }
```

خروجی:

```
32
```

مثال ۳:

```
public  static void main(String arg[]){
      int a=1;
      a=a>>32;
      System.out.println(a);

}
```

خروجی:

```
1
```

توابع ریاضی و ثوابت ریاضی :

وجود دارد که در داخل آن تعداد توابع ریاضی از پیش تعیین شده وجود دارد و برای فراخوانی mathدر جاوا کلاسی به نام آنها از حالت کلی زیر استفاده می کنیم .

Math.نام تابع ( ) ;

مثال ۱:

```
Math.pow( );
Math.sin( );
Math.cos( );
```

```
Math.tan( );
Math.atan( );
Math.atan2( );
Math.exp( );
Math.log( ) ;
```

مثال۲:

```
public  static void main(String arg[]){
      double a=14;
      double y=Math.sqrt(a);
      System.out.println(y);

      }
```

خروجی:

```
      3.7416573867739413
```

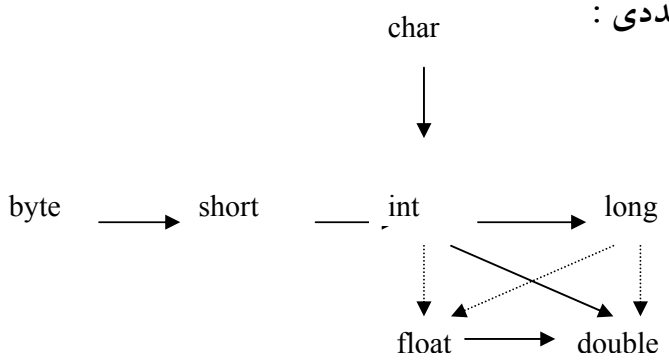
مثال۳:

```
public  static void main(String arg[]){
      double a=14;
      double y=Math.log(Math.exp(a));
      System.out.println(y);

      }
```

خروجی:

```
      14.0
```

تبدیلات بین نوع های داده ای عددی :



توضیح : در شکل بالا در صورتی که در مسیر خطوط پیوسته انجام شود در موقع تبدیل اطلاعاتی (داده ای) از بین نخواهد رفت اما در صورت تبدیل خطوطی در مسیر خط چین ممکن است دقت دادهای کم شود (داده ای از بین برود)

تبدیل صریح و ضمنی میان انواع داده ها :

منظور از تبدیل میان انواع داده ها این است که یک نوع داده را به نوع دیگر تبدیل کنیم . حال اگر این تبدیل را صراحتا به جاوا اعلام کنیم خواهیم گفت تبدیل صریح انجام داده ایم و در صورتی که این تبدیل به تشخیص خود جاوا در حین عملیات انجام شود می گوییم تبدیل ضمنی انجام شده است .

مثال :

```
            double x=9.997;
            int nx=(int)x;
            System.out.println(nx);
```

```
9
```

```
            int c=98;
            char ch=(char)c;
            System.out.println(ch);
```

```
b
            int x=12;
            double y=x;
            System.out.println(y);
      12.0
```

تقدم عملگرها :

عملگرهایی که در یک عبارت در زبان جاوا نوشته می شوند بر طبق تقدم زیر اجرا می شوند :

Operators	Associativity
[] . () (method call)	Left to right
! ~ ++ -- + (unary) - (unary) () (cast) new	Right to left
* / %	Left to right
+ -	Left to right
<< >> >>>	Left to right
< <= > >= instanceof	Left to right
== !=	Left to right
&	Left to right
^	Left to right
|	Left to right
&&	Left to right
||	Left to right
?:	Right to left
= += -= *= /= %= &= |= ^= <<= >>= >>>=	Right to left

اگر در داخل یک عبارت از پرانتز استفاده کنیم این پرانتزها تقدم عملگرها را تغیر می دهد (یعنی ابتدا عبارات داخل پرانتز اجرا می شود)

مثال :

```
            int x=10;
            double y=(10-2)*3;
            double z=10-2*3;
            System.out.println(y);
            System.out.println(z);
      24.0
      4.0
```

نوع داده شمارشی :

تعریف کنیم .enum در جاوا می توانیم نوع داده شمارشی را به کلمه کلیدی version1/4بعد از

مثال :

```
public class MyApp2 {
         enum Size{ SMALL, MEDIUM, LARGE, EXTRA_LARGE };


       public static void main(String[] args) {
                  Size k;
                  k=Size.MEDIUM;
                  System.out.println(k);              }
}
      MEDIUM
```

زمانی که یک نوع داده شمارشی تعریف می کنیم در حقیقت یک نوع داده ای جدید ایجاد می کنیم و برای آن یک نام تعیین می کنیم . (مثلا در مثال بالا نام نوع داده جدید را size گذاشتیم)

نکته : انواع داده شمارشی تنها می توانند مقادیری را به خود بگیرند که در هنگام تعریف آن نوع داده ای مشخص شده باشند .(مثلا در مثال بالا متغیرهایی که از نوع size تعریف شوند تنها چهار مقدار مشخص شده برای آن را می توانند به خود بگیرند .)

قانون نامگذاری :

نام نوع های داده ای تعریف شده با قوانین نامگذاری کلاس ها یکی است . (شروع با حرف بزرگ)

توابع مورد استفاده در ارتباط با رشته ها :

تابع substring به دو صورت تک پارامتر و دو پارامتر قابل استفاده است .

```
String s="Ali Naderi";
s=s.substring(4);
System.out.println(s);
```

```
Naderi
```

نکته : فراخوانی تابع substring توسط خود متغیر انجام می شود .

- بقیه توابع از کتاب چک شود .

ادغام رشته ها :

برای ادغام رشته ها از عملگر جمع استفاده می کنیم .

مثال :

```
String n="Ali";
String f="Naderi";
String name=n+f;
System.out.println(name);
```

```
AliNaderi
```

```
String s="Ali Naderi";
s=s.substring(4,7);
System.out.println(s);
```

```
Nad
```

مقایسه دو رشته :

برای مقایسه دو رشته نمی توانیم از علامت منطقی '==' استفاده کنیم . چرا که نتیجه دلخواه ما را نمایش نمی دهد . لذا از تابع equals() به صورت زیر استفاده می کنیم .

```
String s1="ali";
String s2="Ali";
boolean isEqual=(s1.equals(s2));
```

```
false
```

نکته : برای صرف نظر کردن از حروف بزرگ و کوچک می توانیم از تابع equals ignore case() استفاده کنیم .

```
String s1="ali";
String s2="Ali";
boolean isEqual=(s1.equalsIgnoreCase(s2));
System.out.println(isEqual);
```

```
True
```

کلاس **string builder**:

عملکرد این کلاس مانند کلاس string است . با این تفاوت که کلاس string به صورت آرایه ای ثابت از کاراکترها تعریف می شود و در موقع تغییر محتوای آن مقدار جدید برای رشته در یک حافظه جدید جای می گیرد در حالیکه کلاس string builde ساختار پیاده سازی اش شبیه لیست پیوندی است . برای همین انعطاف پذیری بیشتری برای پیاده سازی دارد .

شکل کلی تابع string builde

```
StringBuilder s=new StringBuilder();  //String s;
s.append("Ali Alizadeh");  //s="Ali Alizadeh"
System.out.println(s.length());
12
```

نحوه گرفتن ورودی از کاربر :

برای گرفتن ورودی از کاربر از کلاسی با نام scanner به صورت زیر استفاده می کنیم .

```
Scanner in = new Scanner(System.in);
System.out.print("What is your name? ");
String name = in.nextLine();
System.out.println("your name is "+name);
What is your name? mehdi
your name is mehdi
```

```
Scanner in = new Scanner(System.in);
System.out.print("Please Enter 2 number? ");
int a=in.nextInt();
int b=in.nextInt();
System.out.println((a+b)/2);
Please Enter 2 number? 15 36
25
```

نکته :برای استفاده از کلاس scanner بایستی خط زیر را به ابتدای برنامه اضافه کنیم .

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

خط فوق همانند دستور include در زبان c و using در زبان #c است . این خط بیانگر آن است که قصد داریم از کلاس های تعریف شده در داخل util که خود بسته ای در درون بسته java می باشد استفاده کنیم .

قالب بندی خروجی در جاوا :

برای قالب بندی خروجی در جاوا از تابع printf( ) به صورت زیر استفاده می کنیم .

مثال ۱/.

```
double x=33.30345678;
System.out.printf("%8.2f", x);
33.30
```

مثال ۲/.

```
System.out.println("Enter your Name?");
Scanner in = new Scanner(System.in);
String name=in.nextLine();
System.out.println("Enter your age?");
int age=in.nextInt();
age++;
System.out.printf("Hello, %s. Next year, you'll be %d", name, age);
Enter your Name?
hamid
Enter your age?
67
Hello, hamid. Next year, you'll be 68
```

مفهوم بلوک در جاوا :

یک بلوک در جاوا محدود به دو علامت {} است. یک بلوک محدوده ی دید متغییر ها را تعریف می کند یعنی متغییری که داخل یک بلوک تعریف می شود برای آن بلوک حکم متغییر محلی را دارد. بنابراین دسترسی به آن تنها داخل آن بلوک و بلوکهای زیر مجموعه ی آن امکان پذیر است.

دستورات شرطی:

دستور *if*

حالت کلی :

```
if ( conditon ){
      statement;
}
```

یک statement می تواند حالت های زیر را داشته باشد:

۱- ساده فقط اگر یک خط باشد

۲- بلوک داخل {}

دستور *if / else*

حالت کلی:

```
if ( conditon ){
      statement 1;
}else {
      statement 2;}
```

دستور *if / else if/ else*

این ساختار همان ساختار if / else است که برای ساده شدن به صورت جداگانه آن را بیان می کنیم.

```
if/ else if/else

if ( conditon 2 ){
      statement 1;
}else if ( conditon 2 ){
      statement 2;
}else {
      statement 3;   }
```

دستور *while*:

تا زمانی که شرط حلقه درست باشد بدنه ی این حلقه تکرار می شود.

```
while ( condition ) {
      Statement    }
```

دستور *do – while*.

تا زمانی که شرط حلقه درست باشد بدنه ی این حلقه تکرار می شود. با این تفاوت که حداقل یک بار اجرا می شود.

```
do {
      Statement;
} while ( condition );
```

دستور for.

تا زمان بر قراری شرط، حلقه تکرار می شود.

```
for ( int i=0 ; i<=max ; i++){
      Statement;
}
```

ساختار switch / case.

در این ساختار دستورات بعد از آن Case ای اجرا می شوند که مقدار مشخص شده ی آن case برابر مقدار متغیر choice باشد. و یا در حالت کلی م ی توان مقدار داده ای شمار شی یا قابل تبدیل به int باشد. یعنی نوع متغیر choice یا از نوع کاراکتر است یا int و یا enum.

```
switch (choice) {

      case 1:
            Statement 1;
            break;

case 2:
      Statement 2;
      break;

case 3:
      Statement 3;
      break;
…
…
…
                                  اجباری نیست و فقط برای حالات در
default:  ------------------------>  نظر گرفته نشده است
      Statement n;
      break;
      }
```

مثال ./

```
switch (score.charAt(0)) {
case 'A':
      System.out.println("Great");
      break;
case 'B':
      System.out.println("Good");
      break;
case 'C':
      System.out.println("Fair");
      break;
case 'D':
      System.out.println("Bad");          }
```

مثال / .

برنام ای بنویسید که نمره ی کار بر را گرفته ار بالای ۱۰ باشد چاپ کند pass مساوی ۱۰ باشد hardly pass و کمتر از ۱۰ باشد faild را چاپ کند.

ب) برنامه را بگونه ای تغییر دهید تا زمانی که نمره منفی وارد نشده ادامه یابد.

```
System.out.println("Enter your Score?");
Scanner in = new Scanner(System.in);
float score=in.nextFloat();
while(score>=0){
      if (score>10) {
            System.out.println("PASSED!");
      } else if (score==10){
            System.out.println("HARDLY PASSED!");
      }
      else{
```

```
                System.out.println("FAILED!!!!");
            }
            System.out.println("Enter your Score?");
            score=in.nextFloat();
        }
        System.out.println("App Finished!");
```

```
Enter your Score?
10
HARDLY PASSED!
Enter your Score?
9
FAILED!!!!
Enter your Score?
20
PASSED!
```

ب)

```
        Scanner in = new Scanner(System.in);
        float score;
        int i;
        for (i = 1;i<=10; i++) {
            System.out.println("i="+i);
            System.out.println("Enter your Score?");
            score=in.nextFloat();
            if (score<0) continue;
            if (score>10) {
                System.out.println("PASSED!");
            } else if (score==10){
                System.out.println("HARDLY PASSED!");
            }
            else{
                System.out.println("FAILED!!!!");
            }
        }
                System.out.println("i="+i);
```

دستور break و continue :

دستور brak باعث می شود که کنترل اجرای برنامه در داخل یک حلقه است بلفاصله از آن حلقه خارج شود و دستورات بعد از حلقه را اجرا کند. و در صورتی که در داخل حلقه جریان اجرا به دستور continue برسد اجرای دستورات بعد از آن صرفه نظر شده و به ابتدای حلقه باز می گردد.(قسمت ب) تمرین قبل)

آرایه ها در جاوا:

آرایه مجموعه متغییر هایی هستند که همگی دارای یک نام می باشند، هر متغیر با اندیس آن از دیگران تفکیک می شود. آرایه در جاوا از اندیس صفر شروع می شود.

نحوه ی تعریف آرایه:

```
int [] a = new int [100];
```

حالت کلی:

```
data type [ ] array name = new data type [number of element];
```

توابع و خصوصیات مورد استفاده در ارتباط با آرایه :

Length طول آرایه :

تابع copy of از کلاس array، از این تابع برای کپی کردن مقادیر یک آرایه به داخل یک آرایه ی دیگر استفاده می شود . صرفه نظر از کپی یک فضا به آرایه ی جدید اختصاص می دهد و آرایه ی قبلی را در نظر نمی گیرد.

یک بلوک ۵ تایی از نوع int تعریف می کند.

```
int [] a = {2,12,3,10,4};
int [] b = new int [5];
```

bمتغیری از نوع ارجاع که به یک آرایه از نوع int

نکته : تابع copy of صرفه نظر از اینکه آیا آرایه ای می خواه به آن کپی کند قبلا به یک آرایه ارجاع دارد یا خیر آرایه ای جدیدی با طول مشخص شده در پارامتر دوم این تابع ایجاد کرده و متغیر مشخص شده را به آن ارجاع می دهد.

کلاسها و اشیاء

یک کلایس به معنی یک قالب یا template است که از روی آن می توانیم نمونه های مختلفی که به آنها شیء یا objectb می گوییم ایجاد کنیم.

یک شی از دو قسمت عمده تشکیل می شود.

۱- خصوصیات

۲- متد ها (توابع)

که هر کدام از آنها برای یک شیء در زمان تعریف قالب شیء (همان کلاس) باید مشخص شود. مقادیری که برای هرکدام از خصوصیات یک شیء تعیین می شود حالت آن شیء را تعیین می کند.در مقابل از متد های یک شیء به عنوان رفتار نام برده می شود.

روشن است که یک کلاس دارای حالت نیست بلکه یک شیء با توجه به مقادیر و خصوصیات آن در حالت های مختلفی قرار می گیرد.

شناسه ی شیء:object identity

شناسه ی شیء در حقیقت یک شیء را به صورت یکتا در حافظه مشخص می کند. ارجاعات گاهی می توانند معادل این شناسه ه در نظر گرفته شوند.

نحوه ی تعریف کلاس در جاوا :

```
class <classname> {
      Properties
      Methods
            }
```

مثال /.

```
class shape {
      String name;
      void sayyourname ( ){
            System.out.println("My Name is "+name);;
                  }
            }
```

نحوه ی ساختن یک شی (نمونه) از یک کلاس:

1)

```
      <class name>  <object name> ;
      <object name>= new <class name ( )>;
```

2)

```
      <class name> < object name > = new <class name ( ) >;
```

مثال /.

```
      1) Shape myshape ;
```

```
        myshape = new Shape();

    2) Shape myshape = new Shape();
```

نحوه ی دسترسی به خصوصیات شی :

```
<object name >.<property>
```

مثال / .

```
myshape.name="ali";
```

نحوه ی دسترسی به متد شیء:

```
<object name >.<method>;
```

مثال/.

```
myshape.sayYourName();
```

اشیای ساخته شده از یک کلاس تمام خصوصیات آن کلاس (قالب) را خواهند داشت.

مثال /.

```
class MyBox {
      Shape s1,s2,s3;
      s1=new Shape();
      s2=new Shape();
      s3=new Shape();

      void sayYourShapesNames(){
            System.out.println(s1.name);
            System.out.println(s2.name);
            System.out.println(s3.name);
      }

}
```

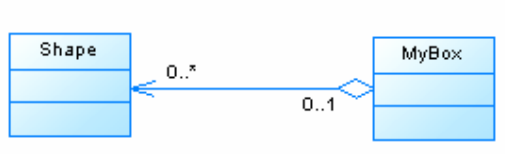
نمونه سازی یک شی از کلاس MyBox (عمل ریخته گری)

```
MyBox box = new MyBox();
box.sayYourShapesNames();
```

توابع سازنده :cunstractors

توابعی هستند هم نام کلاس که مقدار برگشتی آنها در حقیقت ارجاعی است به نمونه ی شیء ساخته شده از آن کلاس.
یک کلاس می تواند چندین توابع سازنده داشته باشد اما همگی باید هم نام با نام کلاس باشند. بسته به نمونه سازی ما در زمان استفاده از دستور new یکی از این توابع سازنده فراخوانی می شود و اگر کلاس تابع سازنده نداشته باشد یک تابع سازنده پیش فرض توسط جاوا برای آن در نظر گرفته شده و اجرا می شود.

ارتباط aggregation (شامل بودن):



نحوه ی تعریف متد ها :

```
<returning type> < method name > ( method parameters ){
                Statement;
}
void sayYourName(){
```

```
            System.out.println("My Name is "+name);
}
```

کلمه ی کلیدی static.

زمانی که یک متغییر در کلاس به صورت استاتیک تعریف می شود در زمان نمونه سازی از آن کلاس برای آن متغییر حافظه تخصیص نمی یابد. بلکه در همان موقع تعریف کلاس به آن حافظه تخصیص پیدا می کند. این باعث می شود به ازای هر شی از یک کلاس حافظه ی مجزایی برای متغییر های غیر استاتیک وجود داشته باشد. بنا بر این متغییر ها مستقل از هم وجود خواهند داشت اما برای متغییر های از نوع استاتیک تنها یک حافظه در کل برنامه اختصاص می یابد و برای همه ی نمونه ها این نوع متغییر ها مشترک هستند. بنا بر این اگر مقدار آن در جایی تغییر کند همه از این تغییر برخوردار می شوند. کلمه ی استاتیک را می توان برای توابع نیز استفاده کنیم اگر به ابتدای تعریف تابع این کلمه را اضافه کنیم به این معنی است که بدون نمونه سازی از آن کلاس می توان با پیشوند قرار دادن نام کلاس آن را فراخوانی کنیم. در صورتی که این کار برای توابع معمولی امکان پذیر نیست. طبیعتا در داخل تابع استاتیک نمی توانیم به متغییر های از نوع غیر استاتیک دسترسی داشته باشیم.

نکته : دسترسی به تابع استاتیک و متغییر استاتیک با استفاده از نام کلاس و بدون ساختن نمونه صورت می گیرد.

مثال /

```
class MyClass {
      int x;
      static int y;

      static void saySth(){
            System.out.println("y="+y);
            MyClass a=new MyClass();
            System.out.println("x="+a.x);

      }
}
```

در تابع مین به این صورت فراخوانی می کنیم و برای متغییر استاتیک همیشه آخرین مقدار نصبت داده شده باقی می اند.

```
            MyClass a = new MyClass();
            a.y=10;
            MyClass b = new MyClass();
            b.y=20;
            System.out.println("a.y = "+a.y);
            System.out.println("b.y = "+b.y);
```

```
            a.y = 20
            b.y = 20
```

ارث بری در جاوا inheritance

در هنگام تعریف یک کلاس آن کلاس می تواند خصوصیات و رفتار خود را از یک کلاس دیگر به ارث برد که در این صورت آن کلاس را اصطلاحا کلاس فرزند و کلاسی که از آن ارث برده را کلاس پدر می نامند. در جاوا یک فرزند مستقیما تنها می تواند یک پدر داشته باشد.

نکته : برای تعریف ارث بری در جاوا از کامه ی کلیدی extends بعد از نام کلاس استفاده می کنیم.

```
class  <childclassname> extends <fatherclass> {
      ……….
      ……….
}
```

نکته : اگر کلاس فرزند فاقد تابع سازنده بشد در زمان فراخوانی تابع سازنده ی کلاس پدر نیز فراخوانی می شود.

کلمه ی کلیدی This:

این کلمه در داخل یک کلاس یه نمونه ای جاری از آن کلاس که یک object یا شی ای است از نوع آن اشاره می کند.

به ودش یعنی به object ای از خودش اشاره می کند.

مثال / .

```
class Line extends Shape{
      int x2;
      int y2;
void draw(){
            System.out.println("I am drawing Line");
            System.out.println("From:"+x+","+y);
            System.out.println("To:"+x2+","+y2);
            }
}
```

کلمه ی کلیدی super:

مشابه کلمه ی کلیدی this است با این تفاوت که این کلمه به جای خود object به پدر object اشاره می کند.

```
void draw(){
      .
      .
      .
      System.out.println("From:"+super.x+","+super.y);
      }
                                                                  }
```

y در کلاس پدر تعریف شده است.

تابع سازنده کلاس line

تا زمانی که تابع سازنده ای برای کلاس فرزندان ایجاد نکرده ایم تابع سازندهی کلاس پدر فراخوانی می شود. اما با ایاد یک تابع سازند در کلاس فرزندان دیگر تابع سازنده ی پدر فراخوانی نمی شود.

```
      Line( ){
            x2=2;
            y2=12;}
```

نکته : یک تابع برای انجام همان عمل تعریف شده در کلاس shape و تابع پارامتر دار shape که می توان هم تمامی دستورات کلاس shape را دراینجا کپی کرد و هم می توان با کلمه ی کلیدی super ارتباط برقرار کرد.

فراخوانی سازنده ی پدر باید اولین دستور در کلاس فرزند باشد.

مثال / .

```
public class Shape {
      String name;
      int x;
      int y;

      Shape(){
            name="-";
            x=0;
            y=0;
      }
      Shape(String n,int a,int b){  ***  تابع سازنده  ***
            name=n;
            x=a;
            y=b;
      }
      protected void sayYourName(){
            System.out.println("My Name is "+name);}
```

نکته : از کلمه ی کلیدی super برای فراخوانی تابع سازنده ی پدر می توانیم استفاده کنیم. همچنین کلمه ی کلیدی this برای فراخوانی تابع سازنده ی دیگری از همان کلاس می تواند استفاده شود. یعنی اگر کلاس دارای ۲ یا چند سازنده باشد می تواند فراخوانی این سازنده ها را با کلمه ی کلیدی this در داخل کلاس انجام داد. اولین دستور در تابع سازنده باید باشد.

مثال / .

```
class Line2 extends Shape2{
      int x2;
      int y2;

      Line2(String name,int x,int y,int x2,int y2){
            super(name,x,y);
            this.x2=x2;
            this.y2=y2;
                  }
      Line2(int o,int p){
            this.x2=o;
            this.y2=p;
      }

      void sayYourName(String name){
            this.name=name;
            System.out.println("My Name is "+name);
      }
}
```

تعیین سطوح دسترسی AccessModifiers

سه نوع سطح دسترسی داریم:

۱- Private

اگر یک خصوصیت یا متد به صورت Private تعریف شود، امکان دسترسی فقط در داخل آن کلاس برای آنها وجود دارد.

۲- Protected

اگر یک خصوصیت یا متد به صورت Protected تعریف شود امکان دسترسی از داخل آن کلاس و کلاسهای فرزند و جود دارد.

۳- Public

اگر یک خصوصت یا ممتد به صورت public تعریف شود امکان دسترسی از خارج از کلاس و کلاسهای دیگر و فرزند وجود دارد.

مثال /.

```
class Animal {
      Color color=new Color();

      public Color getColor() {
            return color;
      }

      public void setColor(Color color) {
            this.color = color;      }
```

اگر سطح دسترسی تعریف نشود در جاوا پیش فرض public نیست و default در نظر گرفته می شود.

مثال / .

```
class Animal {
      Color color=new Color();

      Color getColor() {
            return color;
      }

      void setColor(Color color) {
            this.color = color;      }
```

getTerها و setTer ها در جاوا :

برای دسترسی خصوصیاتی که به صورت private تعریف شده اند استفاده می شود البته از لحاظ مهندسی نرم افزار این فناوری در جاوا برای یک حالت استاندارد برای دسترسی به خصوصیات object ها یا component ها در فناوری هایی مانند Java Beans است. با وجود این با این توابع می توانیم کنترل بیشتری بر روی تغییرات انجام شده یا شیوه ی دسترسی به خصوصیات یک کلاس داشته باشیم.

دلایل استفاده از سطح دسترسی Private و کاربرد Getter ها و Setter ها

۱- امکان سفارشی سازی بیشتر در دسترسی به خصوصیات را فراهم می کد. مثلا اگر بخواهیم خصوصیتی فقط خواندنی باشد ، تابع getter را برای آن تعریف می کنیم و یا تابع setter را به صورت private تعریف می کنیم.

۲- زمانی که بخواهیم در موقع خواندن و یا تغییر property ها علاوه بر تغییر مقدار، عملیات دیگری نیز انجام دهیم یا مقدار فراهم شده برای property عینا به خارج از کلای منعکس نگردد.

۳- Getterها و Setter ها یک .اسط دسترسی استاندارد به خصوصیات کلاسها فراهم می کند.

مثال ./

```
public class Color {
      private int red;
      private int green;
      private int blue;

      public Color(){
            this.red=0;
            this.green=0;
            this.blue=0;
      }

      public Color(int red,int green, int blue){
            this.red=red;
            this.green=green;
            this.blue=blue;
      }

      public int getRed() {
            return red;
      }
      public void setRed(int red) {
            this.red = red;
      }
      public int getGreen() {
            return green;
      }
      public void setGreen(int green) {
            this.green = green;
      }
      public int getBlue() {
            return blue;
      }
      public void setBlue(int blue) {
            this.blue = blue;
      }     }
```

```
class Animal2 {
      Color2 color=new Color2();

      public Color2 getColor() {
            return color;
```

```
    }

    public void setColor(Color2 color) {
        this.color = color;
    }

    public void changeToGreen(){
        color.setRed(0);
        color.setGreen(255);
        color.setBlue(0);
    }
```

{سربارگذاری توابع (Method Overloading) :

منظور از Method Overloading، این است که در داخل یک کلاس، یک تابع با یک نام دارای پیاده‌سازی‌های متعددی باشد و وجه تمایز این تابع‌ها، تعداد پارامترهای ورودی ویا نوع این پارامترها است.

نکته: توجه کنید که مقدار بازگشتی یک تابع نمی‌تواند وجه تمایزی برای دو تابع که سایر مشخصات آنها (مانند: نام، تعداد پارامترها و نوع آنها)یکسان است باشد.

مثال ۱:

```
public class OverLoading {

int add(int a, int b){
System.out.println("int add(int a, int b)is exeacuting!");
return a+b;
}

float add(float a, float b){
System.out.println("float add(float a, float b)is exeacuting!");
return a+b;
}

String add(String a, String b){
System.out.println("String add(String a, String b)is
exeacuting!");
return a+b;
}


public class Example {

public static void main(String[] args) {
OverLoading ob1 = new OverLoading();
System.out.println(ob1.add(12,14));
System.out.println(ob1.add(12.0f,14.0f));
System.out.println(ob1.add("Ali","Alizade"));
}

}
```

خروجی :

```
int add(int a, int b)is exeacuting!
26
float add(float a, float b)is exeacuting!
26.0
String add(String a, String b)is exeacuting!
AliAlizadeint add(int a, int b)is exeacuting!
26
float add(float a, float b)is exeacuting!
26.0
String add(String a, String b)is exeacuting!
AliAlizade
```

* اگر تابع float add(float,float) کلاس OverLoading را به صورت float add(int,int) تغییر دهیم، بنا به نکته بالا با خطای زیر مواجه خواهیم شد.

خروجی :

```
Duplicate method add(int, int) in type OverLoading

at OverLoading.<init>(OverLoading.java:4)
at Example.main(Example.java:5)
```

از آنجایی که سازنده‌ها (Constructors)، نیز خود به نوعی تابع هستند، آنها نیز میتوانند سربارگذاری (Overload) شوند.

مثال:

```
class Line extends Shape{
	int x2;
	int y2;

	Line(String n,int a,int b,int o,int p){
		//super(n,a,b);
		this(o,p);
		name=n;
		x=a;
		y=b;
	}
	Line(int a2,int b2){
		x2=a2;
		y2=b2;
	}
```

Method Overriding :

منظور از Override کردن یک متد، این است که متدی که در کلاس پدر تعریف شده، در کلاس فرزند مجدداً با همان امضاء تعریف شود. در این حالت اگر شیءای از نوع کلاس فرزند ایجاد شود و تابع مذکور فراخوانی گردد، تابع کلاس فرزند فراخوانی خواهد شد که اصطلاحاً گفته می‌شود این تابع، تابع کلاس پدر را Override کرده است.

مثال ٤:

```
public class Shape {
	String name;
	int x;
	int y;

	Shape(){
		name="-";
		x=0;
		y=0;
	}
	Shape(String n,int a,int b){
		name=n;
		x=a;
		y=b;
	}
	protected void sayYourName(){
		System.out.println("My Name is "+name);
	}
	void sayYourName(String s){
		this.name=s;
		System.out.println("My Name is "+s);
	}

}
```

```
class Line extends Shape{
	int x2;
	int y2;
```

```
        Line(String n,int a,int b,int o,int p){
                this(o,p);
                name=n;
                x=a;
                y=b;
        }
        Line(int a2,int b2){
                x2=a2;
                y2=b2;
        }

        void draw(){
                System.out.println("I am drawing Line");
                System.out.println("From:"+x+","+y);
                System.out.println("To:"+x2+","+y2);

        }
        void sayYourName(){
                System.out.println("I am a Line!");
                System.out.println("My Name is "+name);
        }
}
```

نکته :

اگر مقدار بازگشتی متدی را که می‌خواهیم در کلاس فرزند Override کنیم، در این کلاس تغییر دهیم آنگاه با خطا مواجه خواهیم شد.

مثال:

```
public class Shape4 {
        String name;
        int x;
        int y;

        public Shape4(){
                name="-";
                x=0;
                y=0;
        }
        Shape4(String n,int a,int b){
                name=n;
                x=a;
                y=b;
        }
        protected void sayYourName(){
                System.out.println("My Name is "+name);
        }
        void sayYourName(String s){
                this.name=s;
                System.out.println("My Name is "+s);
        }
}
```

پکیج‌ها (Packages) :

یک پکیج در جاوا مجموعه ای از کلاس ها را بسته بندی می کند، نام پکیج ها با حروف کوچک نوشته می شود . آنها از حالت سلسله مراتبی پشتیبانی می کنند. یعنی یک پکیج در داخل خود می تواند پکیج های دیگری نیز داشته باشد . با قرار دادن نام پکیج در ابتدا نام کلاس به صورت زیر مشخص می کنیم که یک کلاس متعلق به کدام پکیج است .

```
Package نام پکیج ;
```

زمانی که پکیج ها تو در تو باشند برای آدرس دهی آنها میان نام آنها از کاراکتر نقطه استفاده می کنیم .

مثال ٦:

p3 داخل p2 قرار دارد.

```
package p2.p3;
```

به ازای ساختار سلسله مراتبی پکیج‌ها ساختار سلسله مراتبی از دایرکتوری‌ها برای فایل های ایجاد شده برای کلاس های جاوا بر روی دیسک وجود دارد . یعنی به ازای کلاسی با نام a فایلی با نام a.Java و با ازای پکیجی با نام p2، دایرکتوری با نام p2 باید وجود داشته باشد.

نکته :اگر در تعریف یک کلاس Access Modifier اعضاء را protected تعریف کنیم آنگاه دسترسی به ویژگی‌ها و متدهای protected تنها در یکی از سه حالت زیر می‌تواند صورت بگیرد :

در یک کلاس

در کلاس‌های به ارث برده شده (فرزند)

در داخل پکیجی که این کلاس در آن تعریف شده است.

کلمه کلیدی Import :

زمانی که بخواهیم از کلاسهای تعریف شده در داخل یک پکیج استفاده کنیم با استفاده از عبارت زیر این مطلب را به جاوا اعلام می کنیم .

; نام کلاس.نام پکیج import

; نام کلاس.نام پکیج.نامپکیج import

; *.نام پکیج import

مثال ٨:

```
package p1;

import p2.Shape4;
```

توابع abstract :

زمانی که بخواهیم در زمان تعریف یک کلاس پیاده سازی یک متد را به کلاس های فرزند واگذار کنیم از توابع abstract استفاده می کنیم .

زمانی یک متد داخل یک کلاس به صورت abstract تعریف می شود خود آن کلاس نیز باید به صورت abstract تعریف شود .

در کلاس هایی که به صورت abstract تعریف می شود به صورت مستقیم نمی توانیم نمونه هایی ایجاد کنیم و تنها می توانیم از این کلاس ها ارث بری کنیم و در کلاس ارث برنده تابع abstract پیاده سازی کنیم .

مثال:

```
public abstract class Creature {
	private int name;

	protected abstract void saySth();

	public void die(){
		System.out.println("I am dieing!!");
	}

}
```

وقتی یک کلاس abstract به ارث می برد :

١ ـ یا بایستی تمام متدهای abstract کلاس پدر را پیاده سازی implement کند .

٢ ـ یا خود آن کلاس نیز به صورت abstract تعریف شود .

نکته : کلاس های abstract علاوه بر متدهای abstract می تواند شامل متدهای غیر abstract نیز باشد .

interface ها در جاوا :

interface در حقیقت یک واسط است . واسطی که یک چهارچوب کلی برای دسترسی به متدهای یک کلاس تعریف می کند .
نحوه تعریف interface در کلاس ها به صورت زیر است :

نحوه دسترسی interface نام واسط { }

همانند کلاس های abstract نمی توانید از interface ها نیز نمونه هایی ایجاد کنید حتی از interface ها نمی توانیم ارث بری داشته باشی چون Class (کلاس) و interface (واسط) دو مفهوم مجزا هستند
. مثال :

```
public interface Marker {
      public void write();
      public String getYourType();
}
```

نکته : برای استفده از یک interface باید آن interface را پیاده سازی کنیم .
نکته : یک کلاس می تواند یک interface را پیاده سازی کند . برای پیاده سازی یک interface از کلمه کلیدی implements استفاده می کنیم .
یک کلاس در عین حال می تواند بیش از یک interface پیاده سازی (implements) کند .
سوال : اگر کلاسی یک interface را implements کند . آیا می تواند abstract هم باشد؟
جواب : بله، چون این دو مفهوم مجزا از یکدیگرند .

نکته : می تواند کلاس abstract متد abstract نداشته باشد . به این معنی که مستقیم نمی توان از آن ارث برد .

نکته : یک interface می تواند از یک interface دیگر ارث ببرد . اما از یک کلاس نمی تواند ارث بری کند . بلکه یک کلاس یک یا چند interface را پیاده سازی می کند .

Polymorphism (چند ریختگی) :

زمانی که متغیری از نوع یک کلاس پدر تعریف می شود این متغیر (ارجاع) می تواند نگه دارنده کلاس های فرزند نیز باشد .
به عبارتی دیگر ارجاعاتی از نوع کلاس پدر می تواند به کلاسهای از نوع فرزند اشاره کند .
مثال :

```
public class A1 {
      public abstract void doSth();
System.out.println("I am Father A1!");
                                                    }
```

```
public class B11 extends A1{
      public void doSth(){
            System.out.println("I am Son B11!");
      }}
```

```
public class B12 extends A1{
	public void doSth(){
		System.out.println("I am Son B12!");
	}
}
```

```
public class MyMain {
	public static void main(String[] args){
		A1 a1 = new A1();
		a1.doSth();		//I am Father A!
		a1=new B11();
		a1.doSth();		//I am son B11!
		a1=new B12();
		a1.doSth();		//I am son B12!
}
```

نکته : ارجاعی از نوع فرزند نمی تواند به ارجاعی از نوع پدر اشاره کند (یعنی برعکس قضیه فوق الذکر صادق نیست) .

حال اگر تابعی در کلاس پدر داشته باشیم به طوریکه هر کدام از فرزندان نیز با همان امضا به شکل متفاوتی برای خودشان پیاده سازی کرده باشند و متغیری (ارجاعی) از نوع کلاس پدر نیز در برنامه موجود باشد و یک کد فراخوانی برای تابع مذکور نیز موجود باشد ، با توجه به اینکه در زمان اجرا چه نمونه ای داخل ارجاع پدر قرار می گیرد تابع مناسبی برای اجرا به صورت پویا انتخاب می شود . این پدیده را پولی مورفیسم یا چند ریختگی می گوییم .

مثال :

```
public class B13 extends A1 {
	public void doSth() {
		System.out.println("I am Son B13!");
	}
}
```

```
public class MyMain {
	public static void main(String[] args){
		A1 a1;
		Scanner in = new Scanner(System.in);
		System.out.print("B11 or B12 or B13 instance?");
		String name = in.nextLine();
		if (name.equals("B11")){
			a1=new B11();
			a1.doSth();
		}
		else if (name.equals("B12")){
			a1=new B12();
			a1.doSth();
		}
		else if (name.equals("B13")){
			a1=new B13();
			a1.doSth();
		}
		else{
			System.out.println("Invalid Input!");
		}
			a1.doSth();}
```

با توجه به مثال اخیر زمانی که فرزندی مثل **B13** دارای متد **dosth( )** نیست زمان فراخوانی متد ، طبیعتا متد کلاس پدر فراخوانی خواهد شد .

برای اینکه تمامی فرزندان را مجبور کنیم که حتما متد **dosth( )** خودشان را داشته باشند لازم است این متد را در کلاس پدر به صورت **abstract** تعریف کنیم .

کلاس پدر **(A1)** را به صورت زیر تغییر می دهیم .

```
public abstract class A1 {
	public abstract void doSth();}
```

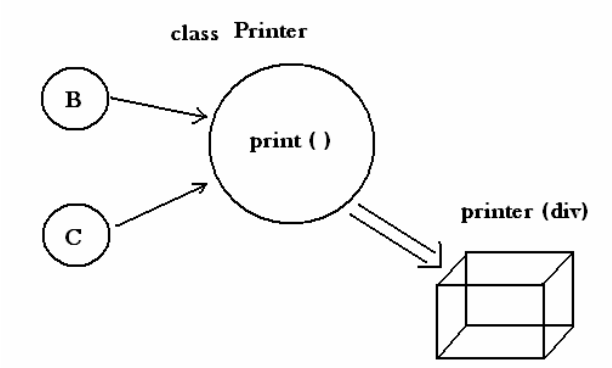
الگوهای طراحی (Design Patterns) :

الگوهای طراحی در حقیقت راه حل هایی را برای حل یک سری مسایل و مشکلات متداولی که در زمان طراحی یک سیستم شی گرا ایجاد می شوند ارایه می کند . از آنجاییکه برخورد با این مسایل به وفور در بحث سیستمهای سی گرا مطرح می شود ، راه حل های ارایه شده برای آنها به صورت سازمان یافته درآمده و برای هرکدام از مسایل و راه حل ارایه شده برای آن نامی نیز انتخاب شده است و به عنوان یک الگو شناخته می شود . به طوریکه مجموعه این مسایل و راه حل ها تحت عنوان الگوهای طراحی شی گرا مطرح می شود .

۱ ـ الگوی یگانه (Singleton Pattern):

در طراحی یک سیستم شی گرا در زمان اجرای یک برنامه گاها این نیاز ایجاد می شود که تنها و تنها یک نمونه از یک کلاس در کل زمان اجرا وجود داشته باشد . به عبارتی دیگر اگر کلاسی با نام **A** داشته باشیم می خواهیم در کل برنامه تنها و تنها یک نمونه از کلاس **A** وجود داشته باشد . مثلا اگر در داخل کلاسی مانند **B** یک نمونه از کلاس **A** ایجاد شد و سپس کلاسی مانند **C** بخواهد نمونه ای دیگر از کلاس **A** ایجاد کند این امکان برایش وجود نداشته باشد بلکه آن نمونه کلاسی که قبلا توسط کلاس **B** ایجاد شده است برای استفاده در اختیار کلاس **C** قرار بگیرد .

مثال:



مثال : برنامه مدیریت کنترل پورت‌های **Com**

به عنوان مثال اگر کلاسی مسئول یک پرینتر در سیستم باشد بایستی تنها یک نمونه از این کلاس در کل سیستم موجود باشد با توجه به این شرط که تمامی اشیاء داخل آن سیستم برای انجام کار پرینت بایستی از یک نمونه از کلاس **Printer** استفاده کند . تنها بایستی یک نمونه از کلاس موجود باشد تا در کار پرینت کردن هرج و مرج اتفاق نیافتد .

به عنوان مثال دیگر می توان کلاسی به عنوان مسئول پورتهای ورودی و خروجی یا پورتهای شبکه (پورتهای **Com** و یا پورتهای منطقی شبکه را در نظر بگیریم) . که در این صورت نیز تنها یک نمونه از این کلاس ها بایستی وجود داشته باشد .

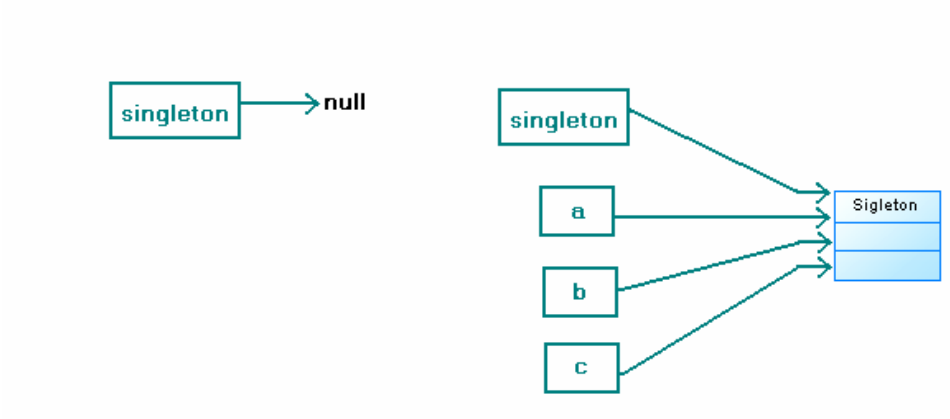
مثال :

```
public class Singleton {
	private String name="";
	private static Singleton singleton;
	private Singleton(String name){
		this.name=name;
	}
	public void whatIsYourName(){
		System.out.println(this.name);
	}
	public static Singleton getInstance(String name){
		if (singleton==null){
			singleton=new Singleton(name);
		}
		return singleton;
	}
}
```

علت تعریف شدن شیء Singleton به صور استاتیک:

ما بایستی همواره یک ارجاع به کلاسی که می خواهیم تنها یک نمونه از آن داشته باشیم، داشته باشیم.

بهترین مکان براد داشتن چنین ارجاعی داخل خود همان کلاس است. و از آنجایی که همیشه یک نمونه وجود دارد، بنابراین خصوصیت مذکور باید به صورت استاتیک باشد.




```
            Singleton c=Singleton.getInstance("Hasan");
            Singleton a=Singleton.getInstance("Ali");
            Singleton b=Singleton.getInstance("Sara");
            a.whatIsYourName();
            b.whatIsYourName();
            c.whatIsYourName();

            a.setName("Nader");
            a.whatIsYourName();
            b.whatIsYourName();
        c.whatIsYourName();
```

خروجی :

```
Hasan
Hasan
Hasan
Nader
Nader
Nader
```

سوال :

اگر بخواهیم مشخصه ی نام را تغییر دهیم، آنگاه چکار باید کرد؟

از انجایی که تغییر نام ارتباطی با نمونه سازی ندارد، و تنها با تغییر یک ویژگی (Name)، به نام جدید این امکان فراهم می شود.

چون name از نوع دسترسی private است در داخل تابع main نمی توانیم آن را تغییر دهیم و باید از توابع getter و setter استفاده کنیم.

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

مثال/.

می خواهیم کلاسی به نام Spool تعریف کنیم که این کلاس مسئول ارسال مستندات به یک پرینتر است. کلاس تابعی برای پرینت دارد که یک رشته را به عنوان پارامتر ورودی گرفته و قرار است آن را چاپ کند. رشته های گرفته شده داخل یک آرایه جهت انجام عمل چاپ قرار می گیرد و در زمان مناسب چاپ می شود. همواره در کل سیستم بایستی یک نمونه از کلاس Spoolوجود داشته باشد.

کلاس مذکور را تعریف کنید به طوری که شرایط اخیر را داشته باشد.

```
public class Spool {
```

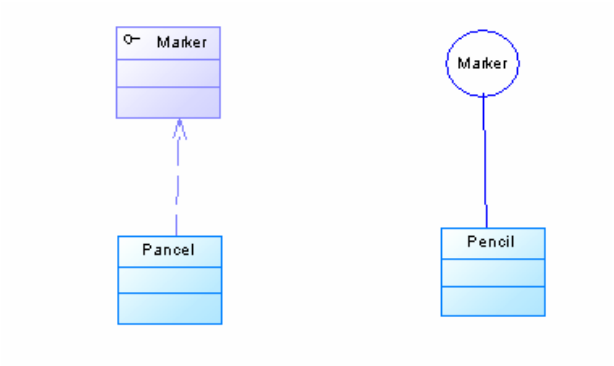
```
        private String []name= new String [5];
        private static Spool a;
        private int i=0;
        private Spool (){

        }
        public void print (String m){
                name[i++]=m;
                        }
        public static Spool getInstance()
        {
                if (a==null){
                        a=new Spool();
                }
                return a;
        }
}
```

زمانی که یک کلاس یک interface را پیاده سازی می کند ارجاعاتی به آن interface می تواند به نمونه های ایجاد شده از آن کلاس نیز ارجاع کند.

به عنوان مثال اگر marker یک interface باشد و pancel کلاسی باشد که آن را پیاده سازی کرده کد زیر در رابطه با انها صحیح است.

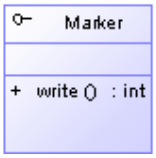
```
public class Pencil implements Marker{


        public String getYourType() {
                return "Pencil";
        }
        public void write() {
                System.out.println("I am writting with Pencil!");
        }
                                                                        }
// in main ()

                Marker k;
                k=new Pencil();
```

نحوه ی نمایش interface ها در UML :



نحوه ی نمایش Access modifier ها در UML:

public → +
private → -
protected → #



اگوی کارخانه یا Factory pattern:

الگوی factory(کارخانه) در مواقعی مطرح می شود که بخواهیم یک نمونه از یک کلاس را ایجاد کنیم . در یک سیستم ممکن است وضعیتی ایجاد شود که با توجه به شرایط سیستم نمونه ای که می خواهد ایجاد شود بایستی متفاوت شود. به صورت معمول در زمان نمونه سازی از سازنده ی یک کلاس برای ساختن نمونه ای از آن کلاس استفاده می کنیم. اما در وضعیت توصیف شده ساختن یک نمونه مستقیما از طریق سازنده ی کلاس مشکل را برطرف نمی کند.

مثال ./

```
public class MarkerFactory {

      private static int penCount=2;
      private static int pencilCount=1;
      }

      //------- interface -----

public interface Marker {
      public void write();
      public String getYourType();
}}
```

نحوه کار با واسط های گرافیکی در جاوا :

در جاوا فن آوری به نام **Swing** وجود دارد که با استفاده از آن می توانیم واسط گرافیکی ایجاد کنیم . منظور از واسط های گرافیکی عناصری مانند فرم ، دکمه ، جعبه متن ، منو و هر عنصر دیگری است که می تواند در داخل پنجره های ویندوز ظاهر شود .

Object JFrame :

Object JFrame امکان ایجاد یک پنجره **Window** در جاوا را می دهد . می توانیم از این شی ارث برده و پنجره خود را سفارشی کنیم (یعنی عناصری مثل دکمه را به آن اضافه کنیم .)

نکته : تمامی کلاس های مربوط به **Swing** داخل `javax.swing.JFrame` قرار دارد .

مثال :

```
import javax.swing.JFrame;

public class MyFrame extends JFrame{
      public MyFrame(){
            this.setSize(400,300);
      }
```

}

در داخل تابع **main( )** :

```
MyFrame f=new MyFrame();
            f.setVisible(true);
            f.setTitle("THis is F1!");
            f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            f.setLocation(500,300);
            f.getContentPane().setBackground(new Color(130,140,150));

            MyFrame f1=new MyFrame();
            f1.setVisible(true);
```

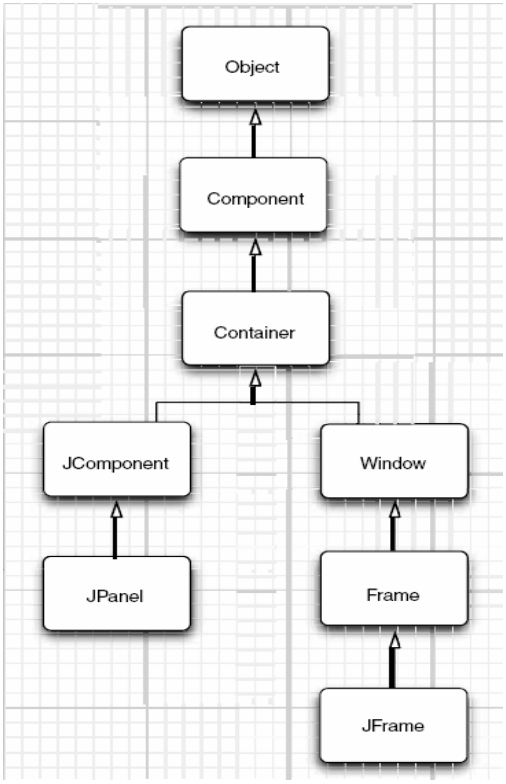
```
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

باعث می شود در زمان کلید کردن بر روی کلید **close** آن فرم ، کل برنامه به اتمام برسد .

```
f.setLocation(500,300);
```

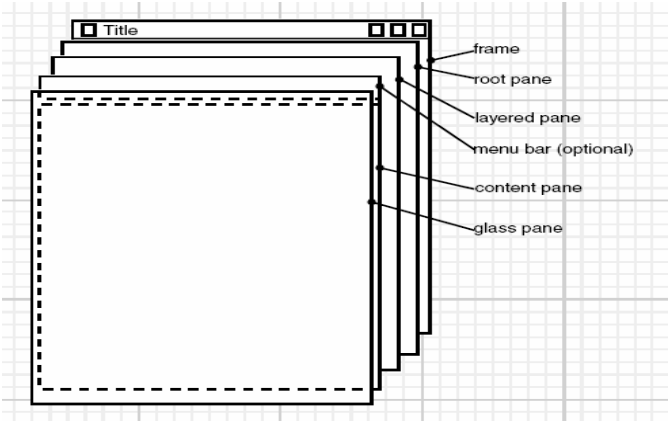
نقطه بالای سمت چپ فرم را تعیین می کند ، یعنی محلی را که فرم قرار است در آن نقطه ظاهر شود .

سلسله مراتب اشیاء در فن آوری Swing



کلاسهایی که از کلاس container به ارث برده اند می توانند شامل عناصر دیگری در داخل خود باشند. به عنوان مثال Jpanel و Jframe خود می تواند شامل object هایی از نوع Button و Textbox و ... باشد.پ

ساختار یک Jframe در جاوا :



ترتیب قرار گرفتن object ها در Jframe به ترتیب زیر است.

۱-object frame
۲-Object root pane
۳-Object layer Pane
٤-Menu Bar (Optional)

Content pane-٥

Glass pane-٦

در ترتیب بالا هر Object بر روی Object دیگری قرار میگیرد.

Object Menu Bar برای افزودن یک منو به داخل Jframe است و Content pane جایی است که معمولا عناصر داخل صفحه را برای نمایش داخل Jframe به آن اضافه می کند.

ترسیم : نحوه ی ترسیم اشکال در داخل Jframe و سایر (Components) Container ها متدی به نام Paint Component با امضای زیر است :

```
public void paintComponent(Graphics g){    }
```

درداخل کلاس Component وجود دارد که با Override کردن آن می توانیم نحوه ی ترسیم کامپوننت را تغییر دهیم . هر زمانی که لازم باشد یک Componentبه هر علتی بر روی صفحه مجددا ترسیم شود این متد به صورت خودکار توسط جاوا برای رسم کامپوننت فراخوانی می شود. زمانی که لازم باشد برنامه نویس این متد را فراخوانی کند بهتر است به جای فراخوانی مستقیم متد آن از متد repaint () با امضای زیر استفاده کند.

```
1- void repaint()
2- public void repaint(int x, int y, int width, int height)
```

پارامترهای دوم قسمتی از کامپوننت را که بایستی مجددا ترسیم شود را مشخص می کند.

مثال / .

```
package p1;

import java.awt.Graphics;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class NotHelloWorldFrame extends JFrame {
        public NotHelloWorldFrame()
        {
             setTitle("NotHelloWorld");
             setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
             // add panel to frame
             NotHelloWorldPanel panel = new NotHelloWorldPanel();
             setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
             add(panel); //this.getContentPane().add(panel);
         }
        public static final int DEFAULT_WIDTH = 300;
        public static final int DEFAULT_HEIGHT = 200;
        public static void main(String[] args){
              NotHelloWorldFrame myform=new NotHelloWorldFrame();
              myform.setVisible(true);
        }
}

class NotHelloWorldPanel extends JPanel {
  public void paintComponent(Graphics g)
  {

     g.drawString("Not a Hello, World program", MESSAGE_X, MESSAGE_Y);
  }
  public static final int MESSAGE_X = 75;
  public static final int MESSAGE_Y = 100;
}
```

خروجی:

```
public NotHelloWorldFrame()
```

تابع سازنده

```
setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
```

ثابت های تعریف شده برای تنظیم اندازه ی فرم

```
setTitle("NotHelloWorld");
```

عنوان فرم را تنظیم می کند.

```
NotHelloWorldPanel panel = new NotHelloWorldPanel();
```

یک شی ء از نوع inner class انتهای برنامه می سازد.

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

هر نمونه از کلاس ساخته شده را با زدن کلید (انتخاب کلید) close خاتمه می دهد.

```
      public static final int DEFAULT_WIDTH = 300;
      public static final int DEFAULT_HEIGHT = 200;
```

محل تعریف ثابت های برنامه (طول و عرض)

```
add(panel);
```

باعث می شود شی ء panel به داخل Object جاری اضافه می شود

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

نقطه ی آغازی برنامه

```
myform.setVisible(true);
```

برای نمایش نمونه ی ساخته شده می باشد

تا قبل از ورژن ۱,٤ به جای `add(panel);` می نوشتیم

this.getContentPane().add(panel);

که Panel را به لایه ی content pane اضافه می کند.add(panel) نیز همین کار را می کند اما برای ورژن ۱,٤ به بعد .

در JDK 1.4 و قبل از آن، برای افزودن یم کامپوننت بر یک فرم بایستی ابتدا لایه ی content pane آن را بدست آورده (Getcontent)سپس متد add آن را فراخوانی کنیم.

```
class NotHelloWorldPanel extends JPanel {
  public void paintComponent(Graphics g)
```

Paint component متد paintcomponent مربوط به کلاس component را override می کند.

```
public void paint Component(Graphics g)
```

g شی و از نوع گرافیک که توسط جاوا به داخل این متد ارسال می شود.

```
     g.drawString("Not a Hello, World program", MESSAGE_X, MESSAGE_Y);
```

یک متد از کلاس گرافیک است که با استفاده از آن می توانیم یک رشته را نقاشی کنیم.

```
  public static final int MESSAGE_X = 75;
  public static final int MESSAGE_Y = 100;
```

ثابت های محل رسم رشته.

کلاس graphic2D

این کلاس امکانات بسیار بیشتری نسبت به کلاس گرافیک برای ترسیم روی کامپوننت ها فراهم می کند. به صورت معمول object که به داخل paintcomponent ارسال می کند object ای از این کلاس است. اما از آنجایی که ارجاعات به کلاس پدر می تواند نگه دارنده ی نمونه های کلاس فرزند باشد امضای تابه paint component در پارامتر های ورودی اش کلاس گرافیک را به عنوان پارامتر ورودی می پذیرد. اما در داخل این تابع با کسب کردن یا تبدیل کردن صریح شی ء ای از توع graphics به شی ای از نوع graphics2D با استفاده از کد زیر می توانیم از امکانات این object استفاده کنیم.پ

```
public void paintComponent(Graphics g)
{
Graphics2D g2 = (Graphics2D) g;
. . . }
```

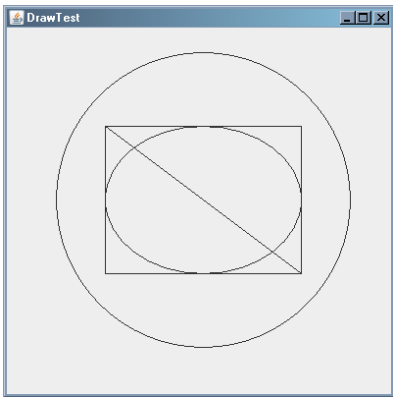
با استفاده از کلاس graphics2D می توانیم اشکالی مانند خط ، مستطیل ، بیضی و ... را رسم کنیم. علاوه بر آن امکاناتی مانند تغییر اندازه ی قلم ترسیم رنگ آن و غیره ... توسط این کلاس نیز امکان پذیر است. برای ترسیم یک شکل گرافیکی لازم است ابتدا نمونه ای از آن شکل گرافیکی را ایجاد کنیم. سپس با استفاده از متد Draw مربوط به کلاس graphics2D آن شکل را ترسیم می کنیم.

مثال ./

```
package p1;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Line2D;
import java.awt.geom.Rectangle2D;

import javax.swing.JComponent;
import javax.swing.JFrame;



      public class DrawFrame extends JFrame {
           public DrawFrame()
           {
                 setTitle("DrawTest");
                 setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
                  // add panel to frame
                  DrawComponent component = new DrawComponent();
                  add(component);

          }

        public static final int DEFAULT_WIDTH = 400;
        public static final int DEFAULT_HEIGHT = 400;
        public static void main(String[] args){
            DrawFrame myform=new DrawFrame();
            myform.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                  myform.setVisible(true);
            }
}

          /**

            * A component that displays rectangles and ellipses.

            */
class DrawComponent extends JComponent
{
    public void paintComponent(Graphics g)
    {
      Graphics2D g2 = (Graphics2D) g;
```

```
//      // draw a rectangle
      double leftX = 100;
      double topY = 100;
      double width = 200;
      double height = 150;
      Rectangle2D rect = new Rectangle2D.Double(leftX, topY, width, height);
      g2.draw(rect);

      Ellipse2D ellipse = new Ellipse2D.Double();
        ellipse.setFrame(rect);
        g2.draw(ellipse);

        // draw a diagonal line
        g2.draw(new Line2D.Double(leftX, topY, leftX + width, topY +
height));
        //draw a circle with the same center
        double centerX = rect.getCenterX();
        double centerY = rect.getCenterY();
        double radius = 150;

        Ellipse2D circle = new Ellipse2D.Double();
        circle.setFrameFromCenter(centerX, centerY, centerX + radius,
centerY + radius);
        g2.draw(circle);
     }}
```

خروجی:



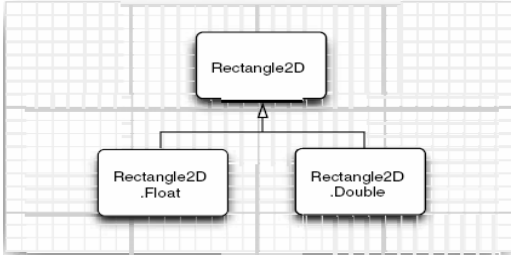

```
g2.draw(rect);
```
رسم چهارچوب (مستطیل)

```
g2.draw(ellipse);
```
رسم بیضی

```
g2.draw(new Line2D.Double(leftX, topY, leftX + width, topY + height));
```
رسم خط

```
Ellipse2D circle = new Ellipse2D.Double();
```
رسم دایره ی بزرگ

در کلاس بالا که از کلاس rectangle2D با ارث برده اند در حقیقت کلاسهای استاتیکی هستند که در داخل کلاس rectangle2D تعریف شده اند .



در جاوا کلاسی که در داخل یک کلاس دیگر تعریف شود. به آن اصطلاحا inner class یا کلاس داخلی گفته می شود.

مدیریت رویدادها Event Handling :

منظور از یک event رویدادی است که اتفاق می افتد و منظور از event handling عملیاتی است که مایل هستیم در زمان اتفاق افتادن آن رویداد انجام شود .

در بحث مدیریت رویدادها در جاوا سه عنصر اصلی زیر دخالت دارد .

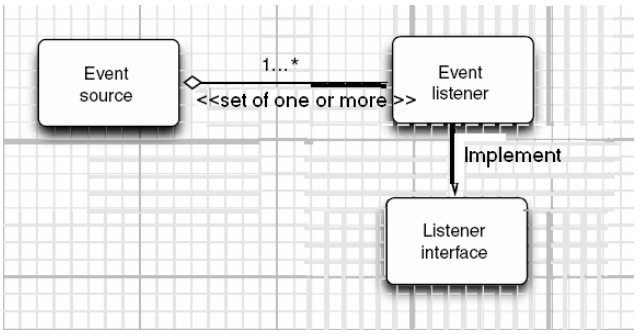
۱ ـ ایجاد کننده رویداد event source

۲ ـ مدیریت کننده رویداد event handling

۳ ـ رویداد event

رویدادهایی که در جاوا اتفاق می افتد متنوع هستند ، یک رویداد می تواند یک کلید ساده موس باشد و یا بسته شدن یک پنجره باشد و یا رویداد رسیدن به یک زمان مشخص باشد .

حتی می توانیم رویدادهایی را در برنامه ها تعریف کنیم و آنها را ایجاد یا اصطلاحا Fire نماییم .



Event Listener (گوش دهنده به رویداد) :

کسانی که مشتاقند در هنگام رخ دادن رویداد به آنها خبر داده شود .

مراحلی که برای مدیریت رویداد ها در جاوا ایجاد می شود مطابق شکل صفحه ی قبل است یعنی یک گوش دهنده به رویداد بایستی وجود داشته باشد یا به عبارتی نمونه ای از آن را ایجاد کنیم. یک ایجاد کننده ی رویداد وجود داشته باشد یا به عبارتی نمونه ای از آن را ایجاد کنیم و در مرحله ی بعدی ایجاد کننده رویداد را به گوش فرا دهنده رویداد گره بزنیم.

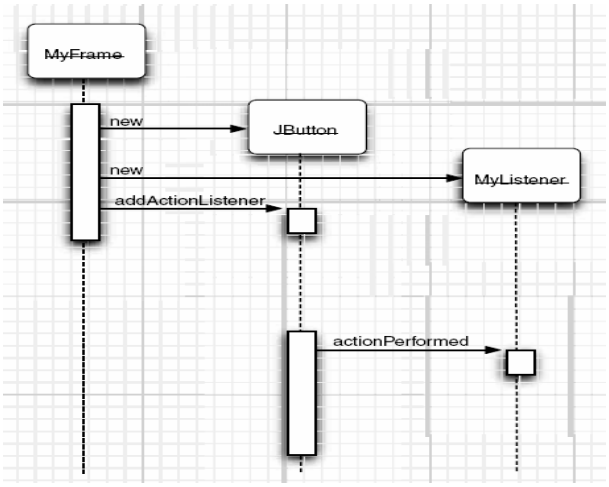
مثال

```
ActionListener listener = . . .;        کد نمونه سازی از یک event listener می باشد.
JButton button = new JButton("Ok");        event source
button.addActionListener(listener);     این سطر کار گره زدن را انجام میدهد.
```

ساختار یک event listener بایستی به صورت زیر باشد .

```
class MyListener implements ActionListener
{
        . . .
public void actionPerformed    (ActionEvent event)
{       ( عنصر سوم ) اطلاعاتی در باره ی رویدادی که اتفاق افتاده ↑
            // reaction to button click goes here
            . . . }
        ( عمل گره زدن – button محل نوشتن )
                        }}
```

تابع action performed در زمان اتفاق افتادن رویداد (فشار دادن دکمه) اجرا می شود. یا به عبارت دیگر کد نوشته شده در داخل آن پاسخی است که در قبال اتفاق افتادن یک رویداد داده می شود. (اجرا می شود .)



مثال / .

```
package b1;
؟{
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;

public class MyForm extends JFrame {
	private static int i=1;
	public MyForm(String title){
		this.setTitle(title);
		this.setSize(300, 400);
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		this.setLayout(new FlowLayout());

		ActionListener listener=new MyListener();
		JButton myButton=new JButton("Change Text");
		myButton.addActionListener(listener);

		JButton myButton2=new JButton("Change Text2");
		myButton2.addActionListener(listener);


		this.add(myButton);
		this.add(myButton2);
	}
	private class MyListener implements ActionListener {
		public void actionPerformed(ActionEvent event) {
			JButton btn=(JButton)event.getSource();
			btn.setText("Event Fired! "+String.valueOf(i));
			i++;
		}
	}
	public static void main(String[] args) {
		MyForm myform=new MyForm("Welcome to Events!");
		myform.setVisible(true);
	}
}
```

خروجی:

```
this.setLayout(new FlowLayout());
```

مسیر چیدمان فرم را تعیین می کند.

```
btn.setText("Event Fired! "+String.valueOf(i));
```

نوشته ی روی دکمه را تغییر می دهد البته بعد از cast کردن رویداد Jbutton

```
JButton btn=(JButton)event.getSource();
```

Btn یک ارجاع از نوع jbutton که ایجاد کننده ی رویداد را بر می گرداند.(my button)

مقدار بازگشتی را cast می کند در قالب Jbutton تا settext را ایراد نگیرد.

که مقدار بازگشتی از نوع object است.

```
+String.valueOf(i));
```

پارامتری که می گیرد را به پشته تبدیل می کند.

نحوه ی تعریف و ایجاد یک شیء بدون تعریف صریح کلاس آن:

زمانی که نیاز داریم از یک کلاس تعریف شده در برنامه تنها یک نمونه ساخته و استفاده کنیم ، می توانیم بدون استفاده از کلمه ی کلیدی Class ساختار کلاس را تعریف کرده و حتی بدون نام گذاری object مربوط به آن از نمونه ی ساخته شده استفاده کنیم

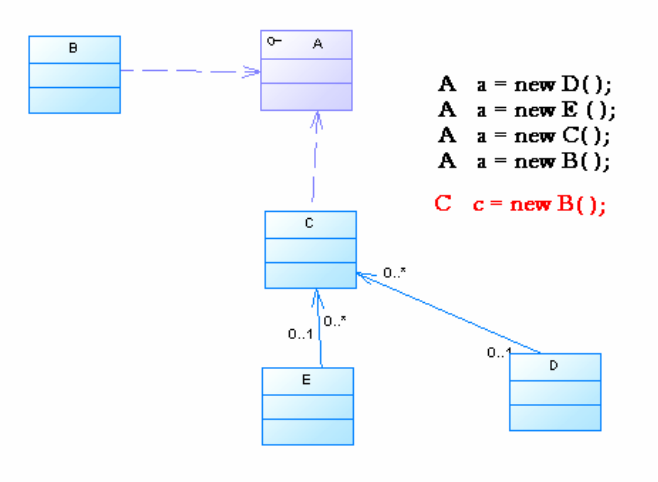
به عنوان مثال در کد زیر نمونه ای برای handle (مدیریت) کردن رویداد کلیک دکمه با شرایط توسیف شده ایجاد کرده ایم.

مثال / .

```
public void makeButton(String name, final Color backgroundColor)
{
JButton button = new JButton(name);
buttonPanel.add(button);
button.addActionListener(new ActionListener()
{
public void actionPerformed(ActionEvent event)
{
buttonPanel.setBackground(backgroundColor);
}
});
}
```

پارامتر یک تابع می تواند interface باشد ، نمونه ای که به تابع فرستاده می شود یا از نوع کلاسی است که آن interface، را پیاده سازی کرده یا از نوع کلاس هایی که یکی از پدرانشان آن را پیاده سازی کرده باشند.

یک نمونه از interface می تواند به کلاسی که آن را پیاده سازی کرده یا یکی از کلاسهایی که یکی از پدرانشان آن را پیاده سازی کرده ارجاع کند.



نکته : interface ظاهر و پوسته است، اما کلاس کامل است.

رویدادهای مربوط به پنجره ها (Window) ها :

رخ داد های متفاوتی در ارتباط با یک پنجره می تواند اتفاق بیفتد، رخدادهایی مثل بازشدن پنجره بسته شدن پنجره، Maximaize,minimaize ، فعال شدن یا غیرفعال شدن پنجره.

همه ی این رویدادها را می توانیم مدیریت کنیم،interface ای به نام windowlistener با ساختار زیر برای مدیریت رویداد ها وجود دارد.

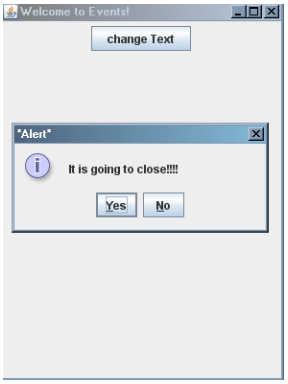
```
public interface WindowListener
{
void windowOpened(WindowEvent e);
void windowClosing(WindowEvent e);
void windowClosed(WindowEvent e);
void windowIconified(WindowEvent e);
void windowDeiconified(WindowEvent e);
void windowActivated(WindowEvent e);
void windowDeactivated(WindowEvent e);
}
```

مثال:

```
package b1;

import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class MyForms5 extends JFrame {
      //MyForms5 form;
      public MyForms5(String title){
            //form=this;
            this.setTitle(title);
            this.setSize(300, 400);
            this.setLayout(new FlowLayout());

            this.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);

            this.addWindowListener(new MyWindowListener());
            JButton myButton=new JButton("change Text");
```

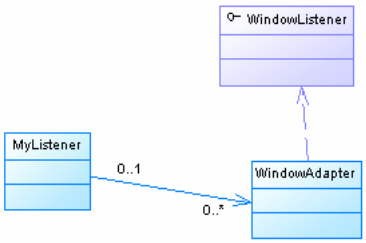
```
            this.add(myButton);
      }

      private class MyWindowListener implements  WindowListener{
            public void windowClosing(WindowEvent arg0) {
                  int r=JOptionPane.showConfirmDialog(null,"It is going to
close!!!!",

      "*Alert*",JOptionPane.YES_NO_OPTION,JOptionPane.INFORMATION_MESSAGE);
                  if (r==JOptionPane.YES_OPTION){
                        System.exit(0);
                  }
            }
            public void windowOpened(WindowEvent e) {}
            public void windowClosed(WindowEvent e) {}
            public void windowIconified(WindowEvent e) {}
            public void windowDeiconified(WindowEvent e) {}
            public void windowActivated(WindowEvent e) {}
            public void windowDeactivated(WindowEvent e) {}

      }
      public static void main(String[] args) {
            MyForms5 myform5= new MyForms5("Welcome to Events!");
            myform5.setVisible(true);
      }
}
```

خروجی:



در این کد با استفاده از کلاس JoptionPane می توانیم جعبه متن هایی مانند : جعبه متن تایید ، جعبه متن پیغام یا جعبه متن گرفتن مقدار از کاربر ایجاد کنیم.

در داخل کلاس کدهای قبل مجبور هستیم که تمامی متدهای مربوط به interface ، windowListener را در داخل کلاسی که آن را implement کرده ، بنویسیم.حتی اگر داخل این توابع خالی باشد.برای حل این مشکل در JDK کلاسهایی به نام Adapter ایجاد شده اند که interface هایی مانند windowListener را با متد هایی با بدنه ی خالی implement کرده اند. حال برای ایجاد یک مدیر رویداد کافی است به جای پیاده سازی این دو listener ، adapter مربوطه را extend کنیم و تنها برای تابع مورد نیاز خود کد بنویسیم. یا اصطلاحا آن تابع را که نیاز داریم override کنیم.



تمرینات

تمرین۱ :

برنامه ای بنویسید که یک رشته را از کاربر گرفته و با دریافت یک کاراکتر از کاربر ، محل اولین جایی که آن کاراکتر در آن قرار گرفته است را چاپ کند .

تمرین ۲ :

برنامه ای بنویسید که مقداری در مقیاس درجه گرفته و sin , cos, tang, cot آن را برای کاربر محاسبه کند .

تمرین۳ :برنامه ای بنویسید که با استفاده از حلقه ها ۲۰ عدد صحیح را بگیرد و میانگین آنها را چاپ کند.mod آنها را چاپ کند و میانه ی آنها را چاپ کند.

تمرین٤ : کلاسی با نام person تعریف کنید به طوری که دارای خصوصیاتی مثل color, name, weight, height باشد سپس سه سازنده که اولی فقط نام را بگیردف دومی نام و رنگ را و سومی هر چهار خصوصیت را به عنوان پارامتر دریافت کند. از کلمه ی کلیدی thid برای فراخوانی ستزنده ها استفاده کنید.

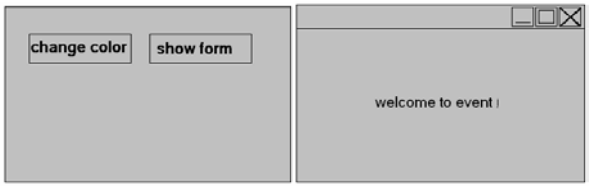
- تابعی با نام () tellyourinfo تعریف کنید که مشخصات آن را چاپ کند.
- کلاسی با نام student تعریف کنید که از person به ارث برد و دارای خصوصیت های Suni, Smajor, Sidباشد. برای این کلاس سازنده ای تعریف کنید که همه ی مشخصات دانشجو را به عنوان پارمتر دریافت کند و تابع ( ) tellyourinfo برای student علاوه بر مشخصات فردی مشخصات دانشجوی ایشان را نی چاپ کند.
- تابعی با نام ( ) tellyourstudentinfo فقط مشخصات مربوط به دانشجو بودن را چاپ کند

تمرین۵ :

دو مثال مختلف با توصیف شرایط هر کدام پیدا کنید که برای آن بتوان با ستفاده از الگوی کارخانه (Factory Pattern) راه حل ارائه داد . شرایط را کامل توصیف کرده ، راه حل آن را نیز پیاده سازی کنید .

تمرین٦:

برنامه ای بنویسید که یک فرم را به شکل زیر ایجاد کرده و در زمان زدن show form فرمی به شکل زیر نمایش دهد. و در زمان زدن change color رنگ پیش زمینه ی فرم را تغیر دهد.

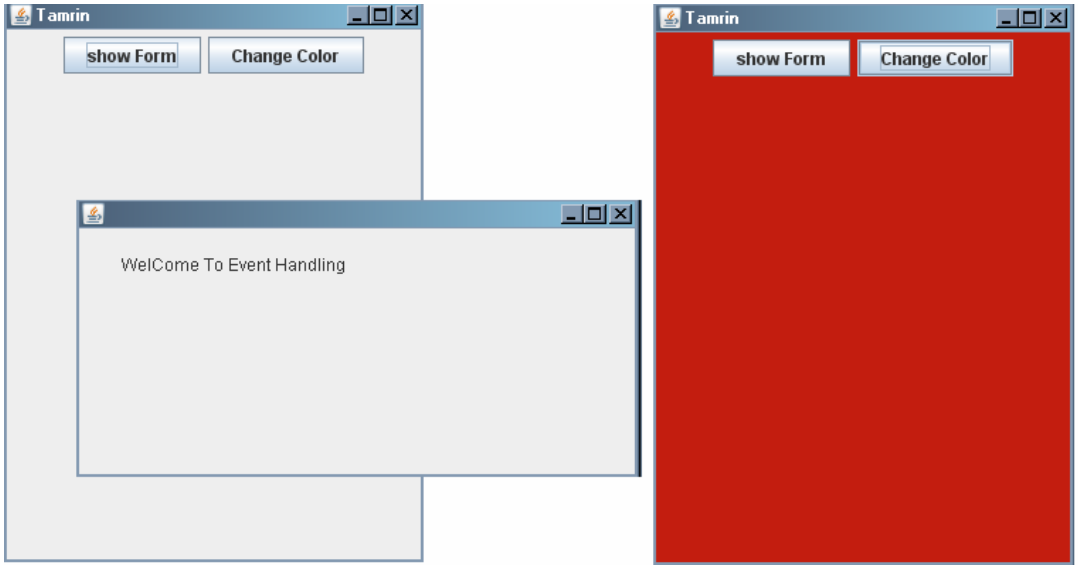


حل تمرین ٦:

```
package b1;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
```

```
import javax.swing.JFrame;
import javax.swing.JPanel;


public class FormT1 extends JFrame {
	FormT1 mform;
	private class FormT2 extends JFrame{
		public FormT2(){
			this.setSize(400, 200);
			//this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
			//this.setLayout(new FlowLayout());
			this.add(new JPanel(){
				public void paintComponent(Graphics g){
					g.drawString("WelCome To Event Handling", 30,
30);
				}
			});
		}
	}

	public FormT1(String title){
		mform=this;
		this.setTitle(title);
		this.setSize(300, 400);
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		this.setLayout(new FlowLayout());

		//-----------------
		JButton myButton1=new JButton("show Form");
		myButton1.addActionListener(new ActionListener(){
			public void actionPerformed(ActionEvent arg0) {
				FormT2 f=new FormT2();
				f.setVisible(true);

			}
		});
		this.add(myButton1);

		JButton myButton2=new JButton("Change Color");
		myButton2.addActionListener(new ActionListener(){
			public void actionPerformed(ActionEvent arg0) {
				int r=(int)(Math.random()*256);
				int g=(int)(Math.random()*256);
				int b=(int)(Math.random()*256);

				mform.getContentPane().setBackground(new
Color(r,g,b));
			}
		});
		this.add(myButton2);

	}

	public static void main(String[] args) {
		FormT1 f=new FormT1("Tamrin");
		f.setVisible(true);
	}
}
```

طراحی سیستم


Tamrin
show Form
Change Color
WelCome To Event Handling
Tamrin
show Form
Change Color