برنامه نویسی جاوا

امیر حسین کاشفی

مرور مطالب

- جلسه اول – آشنایی با زبان جاوا
  - چرا برنامه نویسی جاوا ؟
  - تاریخچه
  - معماری هسته جاوا
  - مراحل نصب
  - انواع داده
  - دستورات ورودی و خروجی
  - یک برنامه نمونه ساده

چرا برنامه نویسی ؟

- Why programming? Need to tell computer what to do by Program
- Machine languages is Tedious and error-prone.
- Natural languages is Ambiguous and hard for computer to parse.
- High-level programming languages Acceptable tradeoff.

چرا جاوا ؟

- تولید انواع برنامه‌ها، در انواع کاربردها
- زبانی مستقل از پلت فرم
- API و کلاس های کمکی بسیار غنی
- یادگیری ساده
- متن باز بودن
- زبانی شیءگرا
- گستردگی استفاده و انجمن‌های غنی
- اشکال‌زدایی قوی

تاریخچه جاوا

- James gosling، mike sheridan، patric naughton در شرکت SUN پروژه Green در ۱۹۹۱

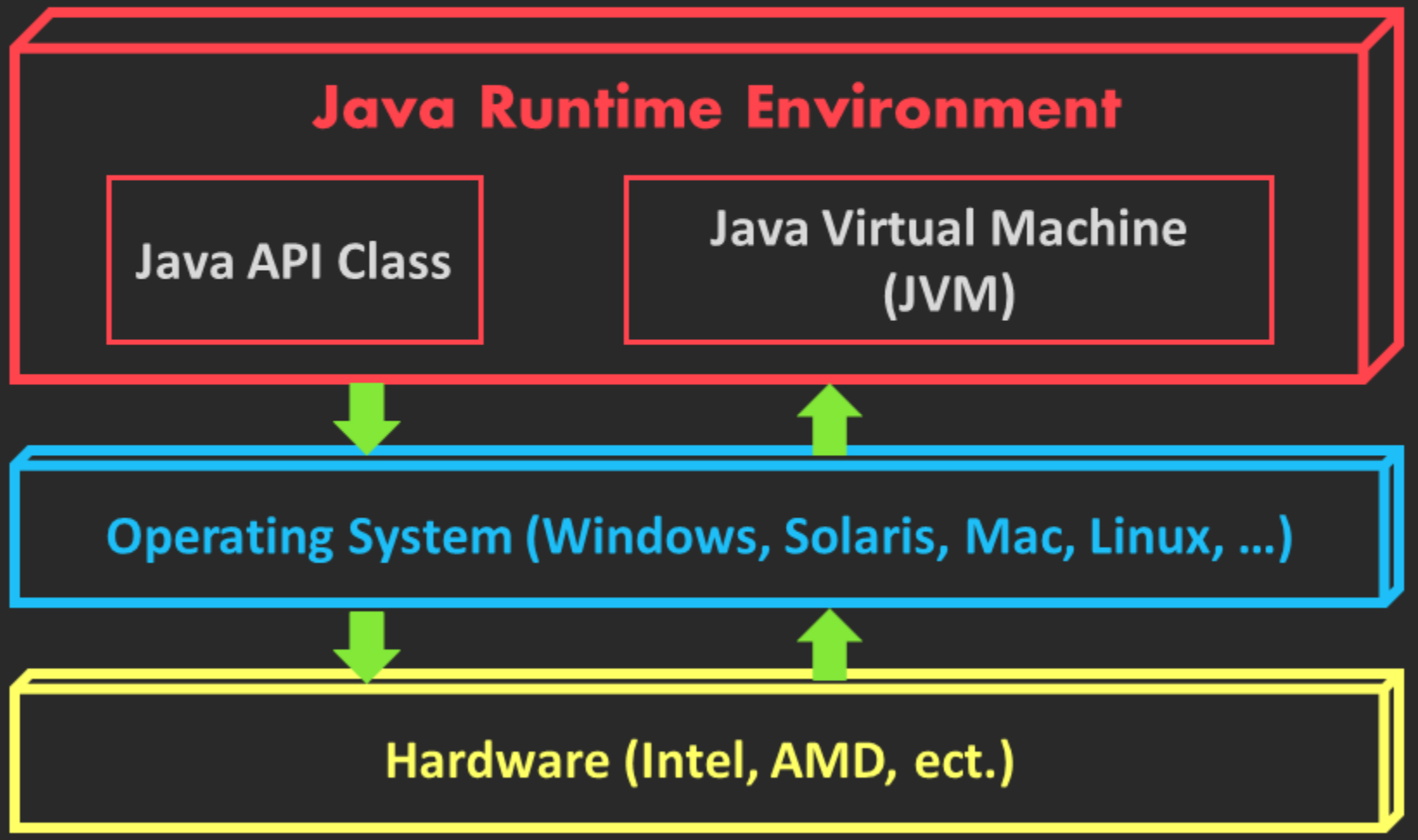
oak

C++

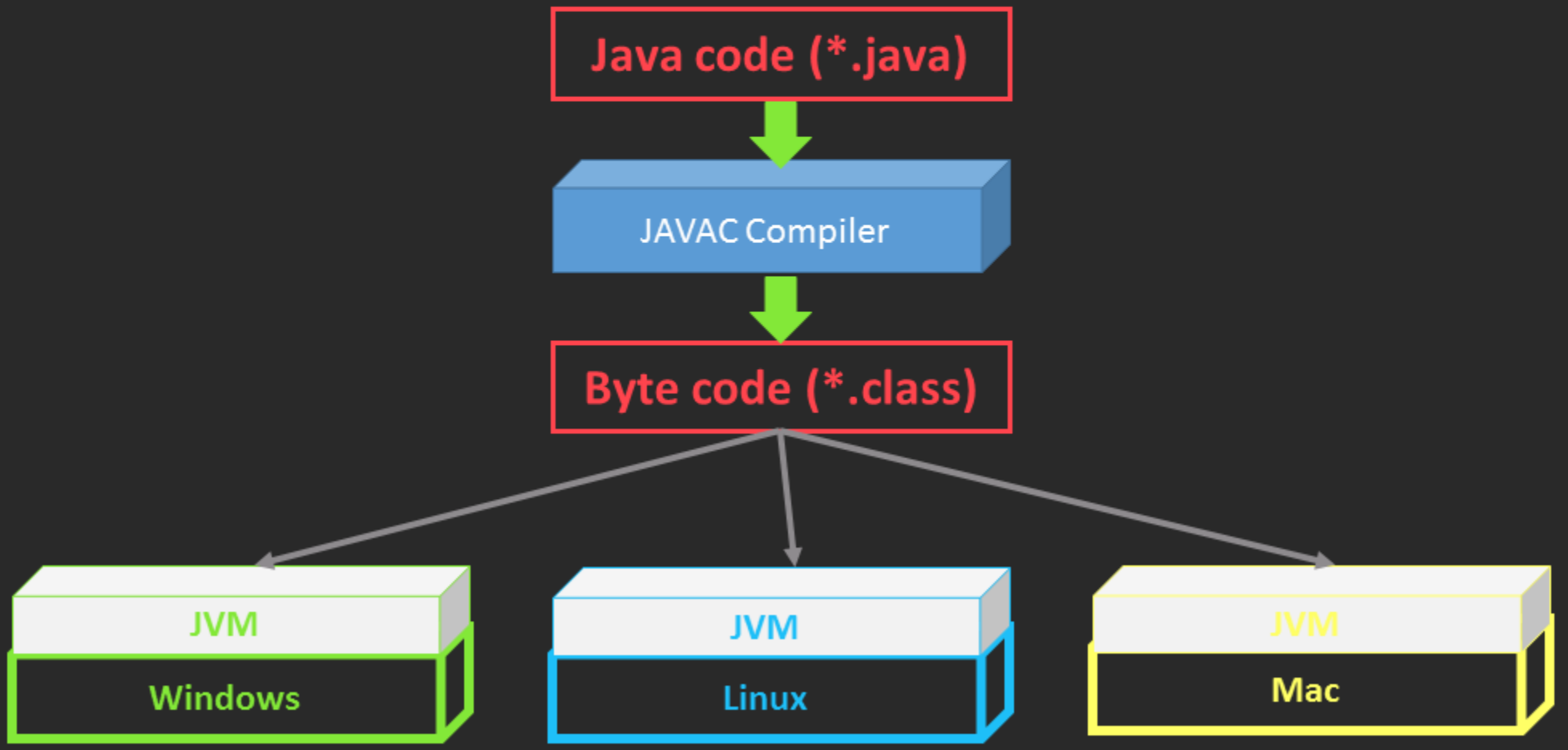
- در سال ۱۹۹۵ این زبان به java تغییر نام پیدا کرد.

Write Once, Run any Where" (WORA)

WORA ??!!


Java Runtime Environment
Java API Class
Java Virtual Machine
(JVM)
Operating System (Windows, Solaris, Mac, Linux, ...)
Hardware (Intel, AMD, ect.)

Java Runtime Environment (JRE)



بسته‌های آموزشی جاوا

- Java Standard Edition (J2SE)

ویژه برنامه نویسی در desktop

- Java Enterprise Edition (J2EE)

ویژه برنامه نویسی در وب

- Java Micro Edition (J2ME)

ویژه برنامه نویسی در تلفن همراه

Java Integrated Development Environment (IDE)

IDE	Logo	License	Written in Java	Windows	Linux	OS X	GUI builder
BlueJ		GPL2+GNU	✓	✓	✓	✓	✗
DrJava	-	Permissive	✓	✓	✓	✓	✗
Eclipse		EPL	✓	✓	✓	✓	✓
IntelliJ IDEA		Community Edition: Apache License v 2.0	✓	✓	✓	✓	✓
JBuilder	-	Proprietary	✓	✓	✓	✓	✓
NetBeans		CDDL, GPL2	✓	✓	✓	✓	✓

مقدمات شروع کار با جاوا

۱ : نصب JRE

۲ : نصب JDK

۳ : نصب IDE

موارد پرکاربرد در جاوا

Built-In Types	
int	double
long	String
char	boolean

System
System.out.println()
System.out.print()
System.out.printf()

Math Library	
Math.sin()	Math.cos()
Math.log()	Math.exp()
Math.sqrt()	Math.pow()
Math.min()	Math.max()
Math.abs()	Math.PI

Flow Control	
if	else
for	while

Parsing
Integer.parseInt()
Double.parseDouble()

Primitive Numeric Types		
+	-	*
/	%	++
--	>	<
<=	>=	==
!=		

Boolean	
true	false
||	&&
!	

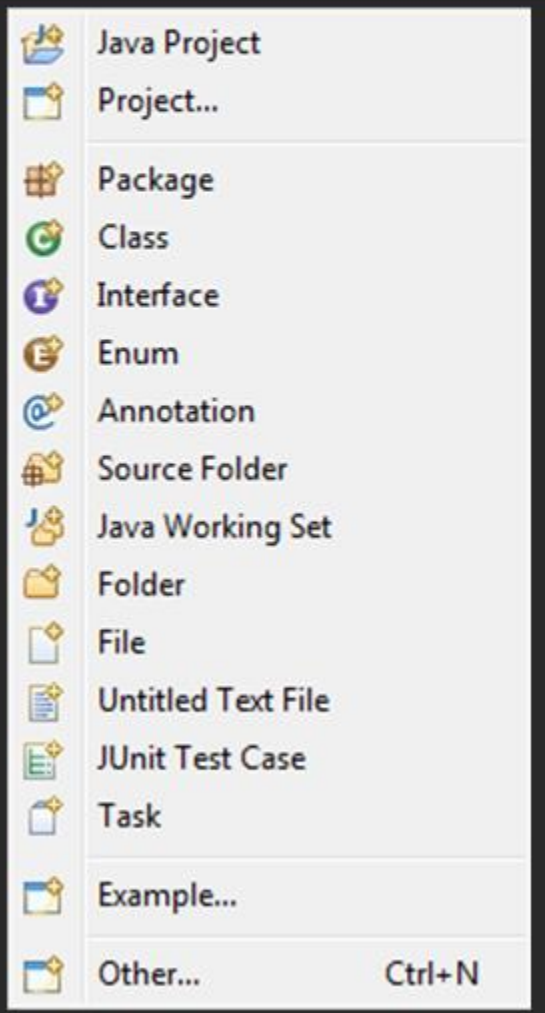
Punctuation	
{	}
(	)
,	;

Assignment
=

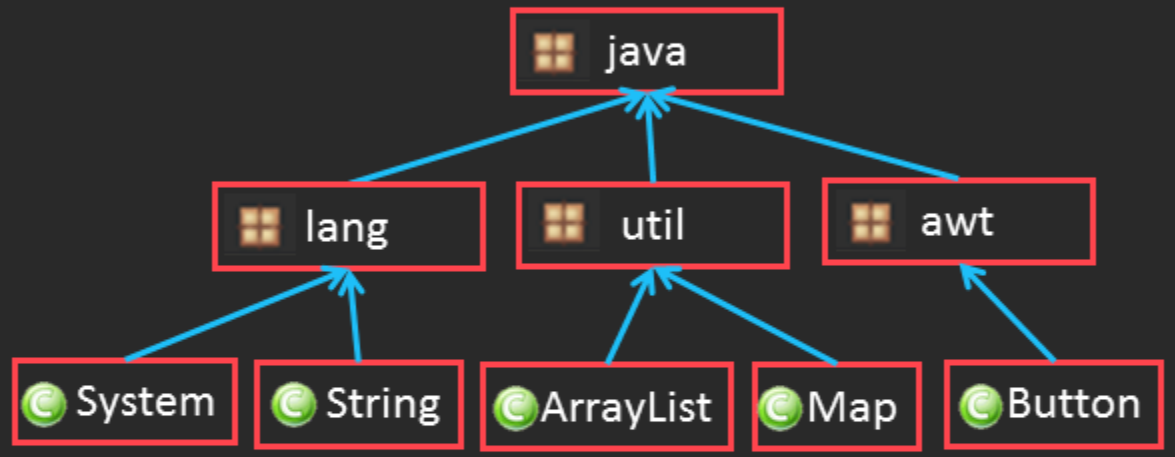
String	
+	""
length()	compareTo()
charAt()	matches()

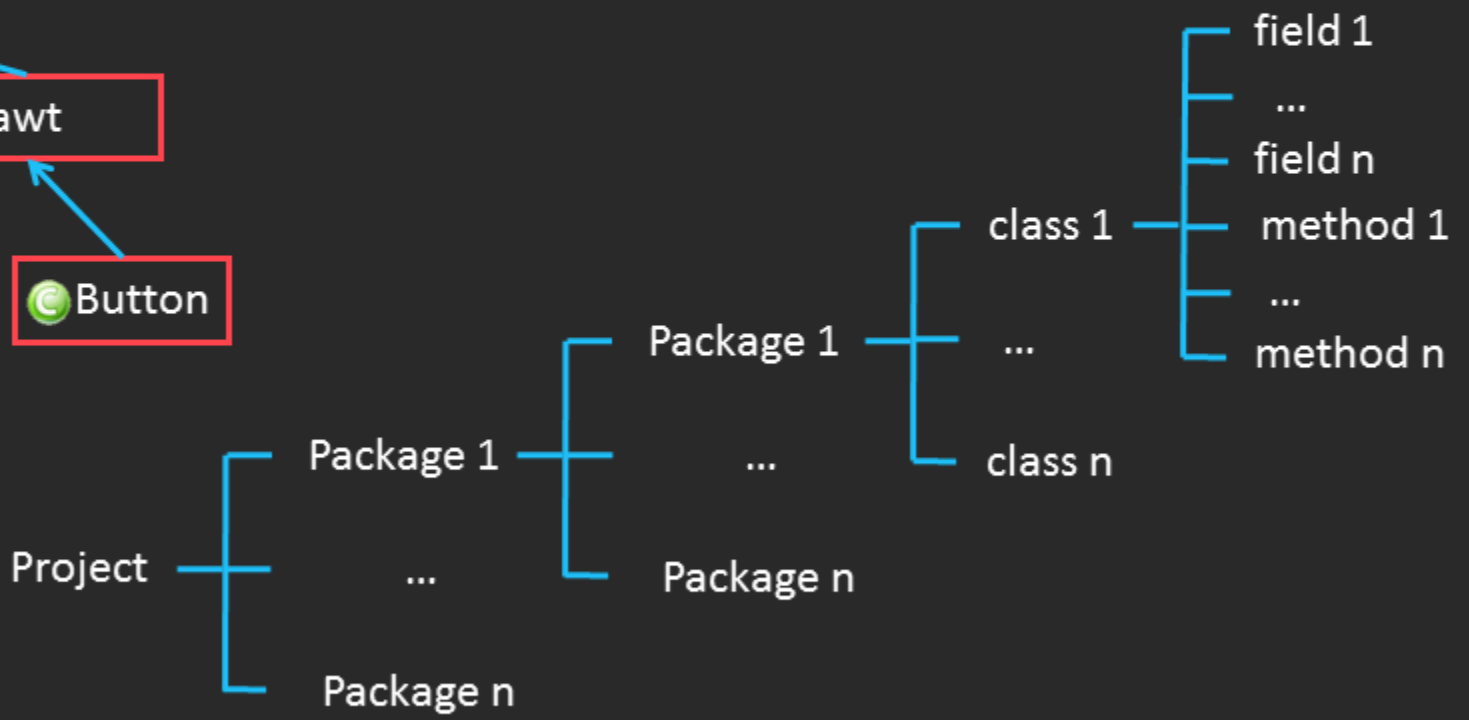
Arrays
a[i]
new
a.length

Objects	
class	static
public	private
final	toString()
new	main()

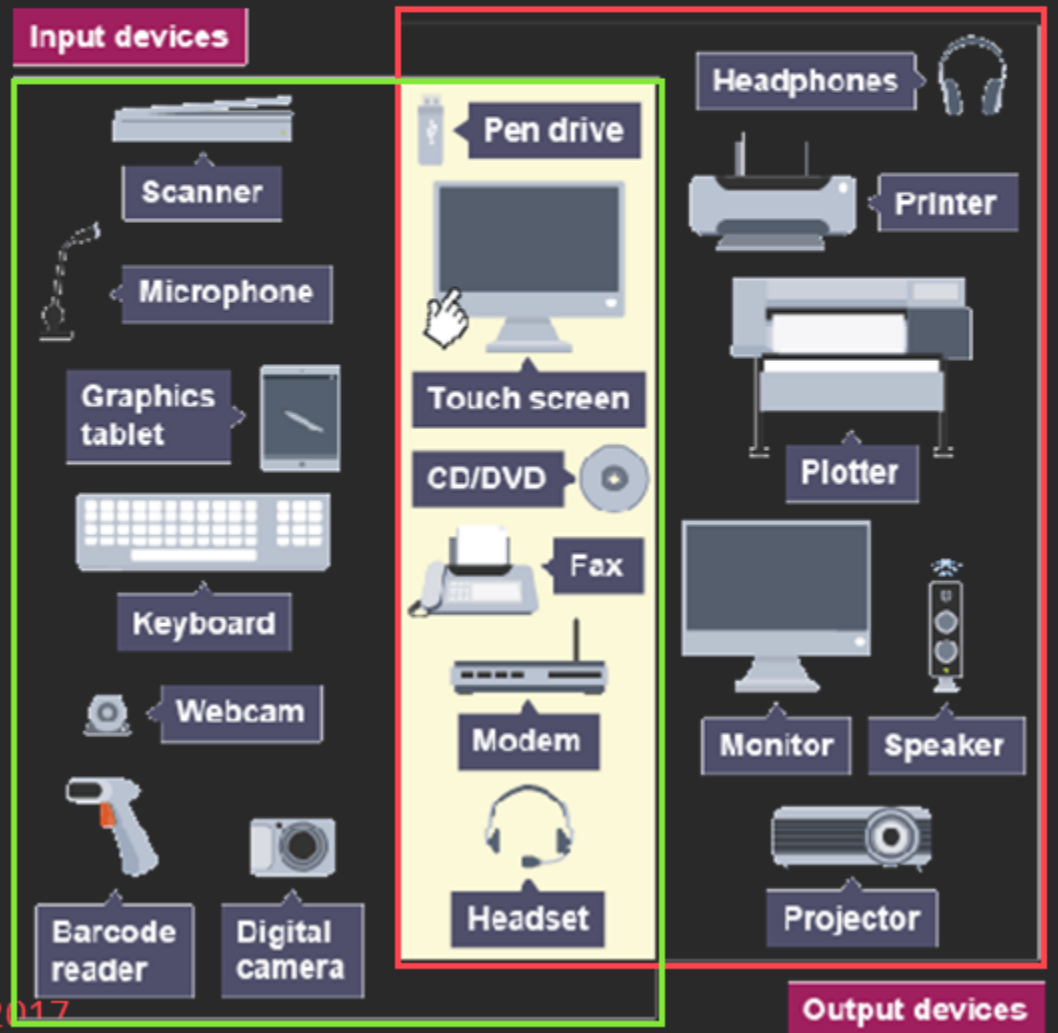


سلسله مراتب کدنویسی در جاوا





ورودی و خروجی



- برای درج دستورات ورودی و خروجی از کتابخانه‌های جاوا استفاده می‌شود.
- رایجترین ورودی و خروجی صفحه کلید و مانیتور هستند:

نمایش در صفحه مانیتور

```
System.out.println (هر مقدار دلخواه);
```

ورودی از صفحه کلید

```
import java.util.Scanner;

Scanner scanner = new Scanner (System.in);
int n = scanner.nextInt();
```

فرمت خروجی

- خروجی را میتوان به‌صورت قالب‌بندی شده با متد format یا printf نمایش داد:

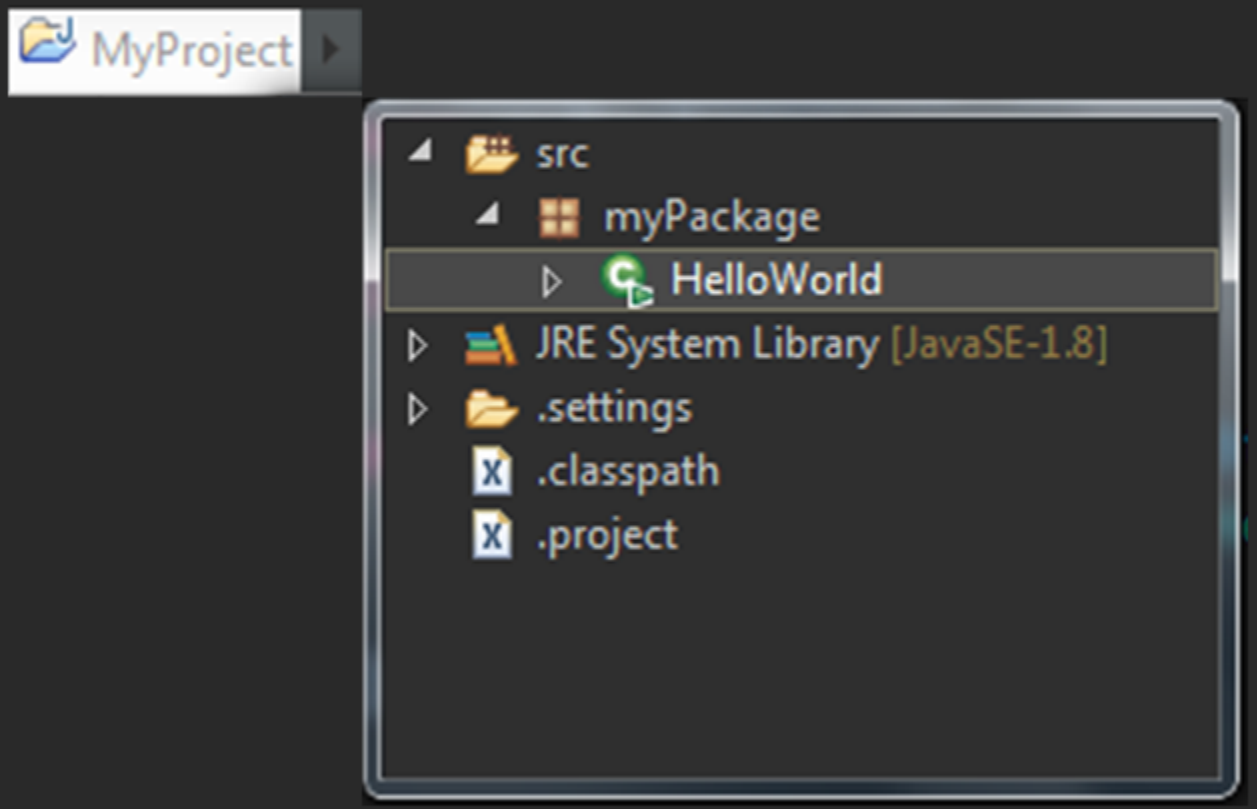
```
System.out.format("%d",c);
System.out.printf("%d",c);
```

%d → Decimals اعداد ده‌دهی
%f → Floats اعداد اعشاری
%s → Strings رشته‌ها

- در این متن دلخواه می‌توان از کاراکترهای ویژه‌ای استفاده نمود که هرکدام مفاهیم خاص خود را دارند:

کاراکتر	مفهوم	کاراکتر	مفهوم
\n	New line	\”	Double Quote
\r	return	\’	single Quote
\t	tab	\b	Backspace
\\	Backslash		

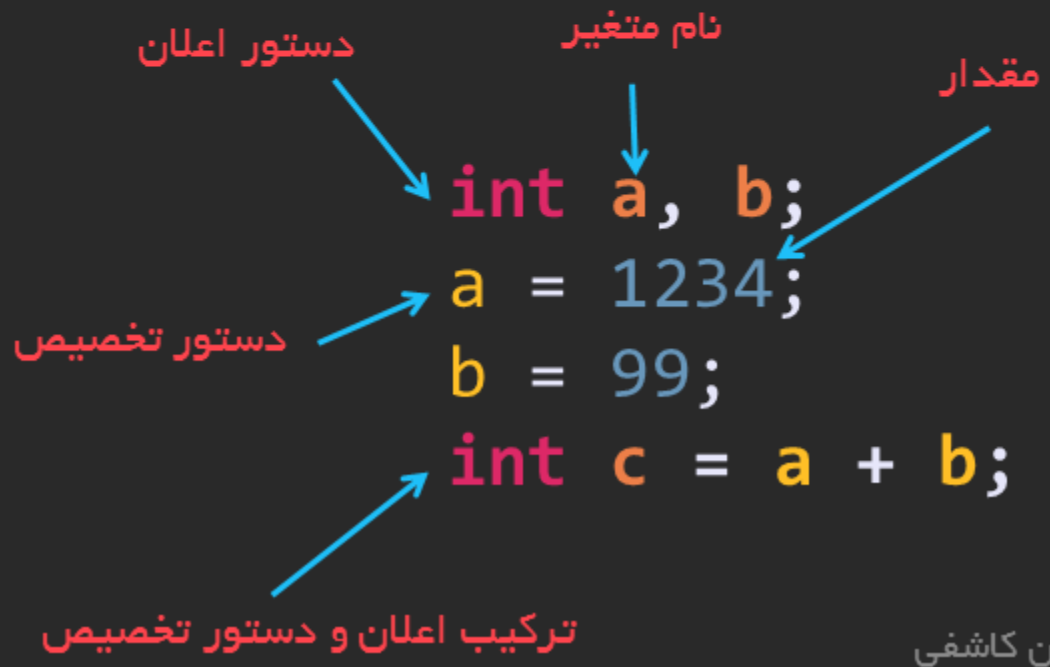
برنامه Hello World




```java
package myPackage;

public class HelloWorld {

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

  }

}
```

تعاریف پایه

- زبان جاوا یک زبان case sensitive بوده و شیوه comment گذاری در آن مشابه زبان C است.
- متغیر (variable): نامی است که به یک مقدار ر جوع می کند.
- دستور تخصیص (assignment statement): ارتباط یک مقدار با یک متغیر



انواع داده

- انواع داده (Data Type): یک مجموعه از مقادیر و عملگرهای تعریف شده روی این مقادیر

نوع داده	مجموعه مقادیر	نمونه مقدار	عملگرها
char	کارکترها	‘a’	مقایسه
String	یک -توالی از کاراکترها	“hello”	+ یا الحاق
int	مقادیر صحیح	17	+, -, *, %, /
double	مقادیر اعداد اعشاری	3.56	+, -, *, %, /
boolean	مقادیر درستی	True / false	&&, || , !

دامنه انواع داده

نوع اولیه	اندازه	کمترین مقدار	بیشترین مقدار	کلاس Wrapper
boolean	-	-		Boolean
char	16 bit	Unicode 0	Unicode 2^{16}-1	Character
byte	8 bit	-128	+127	Byte
short	16 bit	-2^{15}	$+2^{15}$-1	Short
int	32 bit	-2^{31}	$+2^{31}$-1	Integer
long	64 bit	-2^{63}	$+2^{63}$-1	Long
float	32 bit	IEEE754	IEEE754	Float
double	64 bit	IEEE754	IEEE754	Double
void	-	-	-	Void

قابلیت‌های نوع داده رشته‌ها

```
public static void main(String[] args) {
	String str1 = "wellcome to java programming!";
	String str2 = "    I Love Java!    ";
	String str3 = "wellcome to Java ProGramming!";
	System.out.println(str1); // wellcome to java programming!
	System.out.println(str1.length());  // 29
	System.out.println(str1.equals(str2));  // false
	System.out.println(str1.equalsIgnoreCase(str3));  // true
	System.out.println(str1.startsWith("prog"));  // false
	System.out.println(str1.endsWith("!"));  // true
	System.out.println(str1.indexOf("j"));  // 12
	System.out.println(str1.lastIndexOf("o"));  // 19
	System.out.println(str1.replace("java", "C++"));
	//wellcome to C++ programming!
	System.out.println(str1.concat("??"));
	//wellcome to java programming!??
	System.out.println(str1.toUpperCase());
	WELLCOME TO JAVA PROGRAMMING!
	System.out.println(str2.trim());  // I Love Java!
	System.out.println(str1.isEmpty());  // false
2017}
```

تبدیل نوع داده

- تبدیل نوع داده: یک نوع داده را به نوع داده دیگری تبدیل می کند.
- ضمنی دقیق: بدون از دست دادن دقت، بطور ضمنی انجام می‌شود.

“33” + 99 → “3399”

- ضمنی نا دقیق: با از دست دادن دقت، بطور ضمنی انجام می‌شود.

Long → float / double , int → float

- صریح (Casting): با استفاده از متدها صریحا انجام می‌شود.

(int) 4.5 * 5 → 20 , Integer.parseInt(“2”) → 2

- غیرممکن: تبدیل نوع ممکن نیست.

bolean ←→ any data type

یک مثال ساده

```java
package myPackage;

public class Add{

    public static void main(String[] args) {
        int a, b;
        a = 1234;
        b = 99;
        int c = a + b;
        System.out.println(c);
    }

}
```

عملگرهای مهم در جاوا

- ریاضی: عملوندهای عددی را گرفته و یک مقدار عددی بر می گرداند:

  + , – , * , % , / , ++ , -- , - , += , -= , %= , *= , /=

- مقایسه‌ای: عملوندهای بولی را گرفته و یک مقدار بولی بر می گرداند:

  == , != , < , > , <= , >=

- منطقی: عملوندهای بولی را گرفته و یک مقدار بولی بر می گرداند:

  ! , && , ||

```
(b*b – 4.0*a*c) >= 0.0
( year % 100) == 0
( month >= 1 ) && ( month <= 12 )
```

اولویت عملگرها

عملگر	اولیت چندتا از آنها پشت هم
++, -- (post)	راست به چپ
++, -- (pre), +, -, !	راست به چپ
*, /, %	چپ به راست
+, -	چپ به راست
<, <=, >, >=	چپ به راست
==, !=	چپ به راست
&&	چپ به راست
||	چپ به راست
?:	راست به چپ
=, +=, -=, *=, /=, %=	راست به چپ

برنامه نویسی جاوا

امیر حسین کاشفی

مرور مطالب

- جلسه دوم – ساختارهای تصمیم، حلقه‌ها و مدیریت Exception
  - ساختار تصمیم if … then
  - ساختار تصمیم switch … case
  - ساختار تکرار while
  - ساختار تکرار for
  - ساختار try-catch-finally

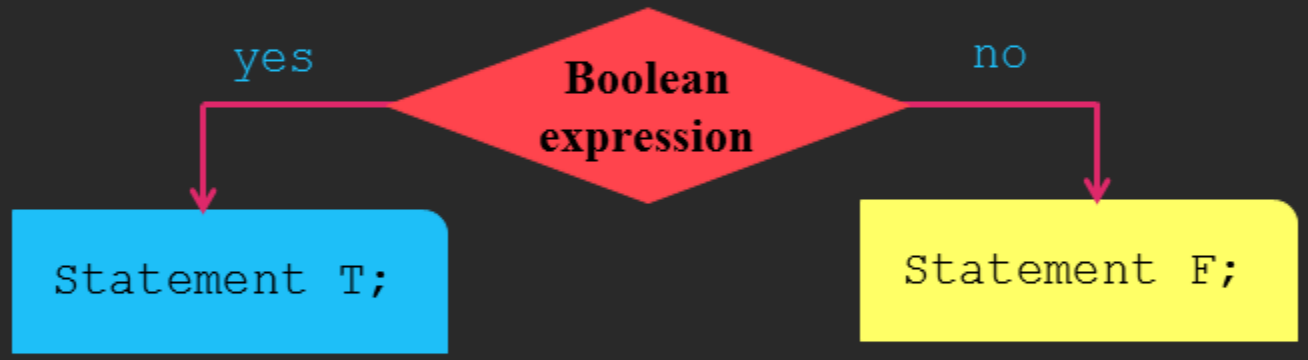
ساختار تصمیم

- ساختار تصمیم if …else
- ساختار تصمیم ? :
- ساختار تصمیم if …else تو در تو
- ساختار تصمیم switch … case

ساختار تصمیم if ... then

- جهت بررسی و تصمیم‌گیری مبتنی بر یک شرط از ساختار تصمیم استفاده می‌شود:

```
if (boolean expression){
      Statement T;
} else {
      Statement F;
}
```



- در صورت درستی شرط Statement T و در صورت نادرستی شرط Statement F اجرا می شود.
- ساختار if می تواند بدون else نیز باشد.

```
if (x%2==0){
  y++;
} else {
  z++;
}
```

ساختار : ?

- اگر چه در صورت درستی شرط و در صورت نادرستی شرط یک متغیر مقدار دهی شود، یا یک مقدار نتیجه شود، می توان ساختار if … then را با استفاده از : ? اجرا نمود.

```
(boolean expression)? Statement T:Statement F;
```

```
if (x%2==0){
    y=3;
} else {
    y=4;
}
```

```
y=(x%2==0)?3:4;
```

```
if (x%2==0){
    System.out.println(3);
} else {
    System.out.println(4);
}
```

```
System.out.println((x%2==0)?3:4);
```

مثال از ساختار تصمیم

```
class Programmer {

public static void main(String args[]) {
   Scanner s = new Scanner(System.in);
   System.out.println("enter a number less than 1000");
   int a = s.nextInt();
   if (a < 500)
      System.out.println("SHIR");
   else
      System.out.println("KHAT");
   }
}
```

ساختار تصمیم تو در تو

- در مواردی که لازم باشد، بعد از else، می توان از if-else دیگر استفاده نمود. این امر می تواند بارها تکرار شود.

```
if (boolean expression1){
      Statement T1;
} else if (boolean expression2){
      Statement T2;
} else if (boolean expression3){
      Statement T3;
}
                  ...
  else
      Statement F;
}
```

```
class Programmer {

   public static void main(String args[]) {
      Scanner s = new Scanner(System.in);
      int a = s.nextInt();
      if (a > 0)
         System.out.println(">0");
      else if (a == 0)
         System.out.println("=0");
      else
         System.out.println("<0");
   }
}
```

ساختار تصمیم Switch ... case

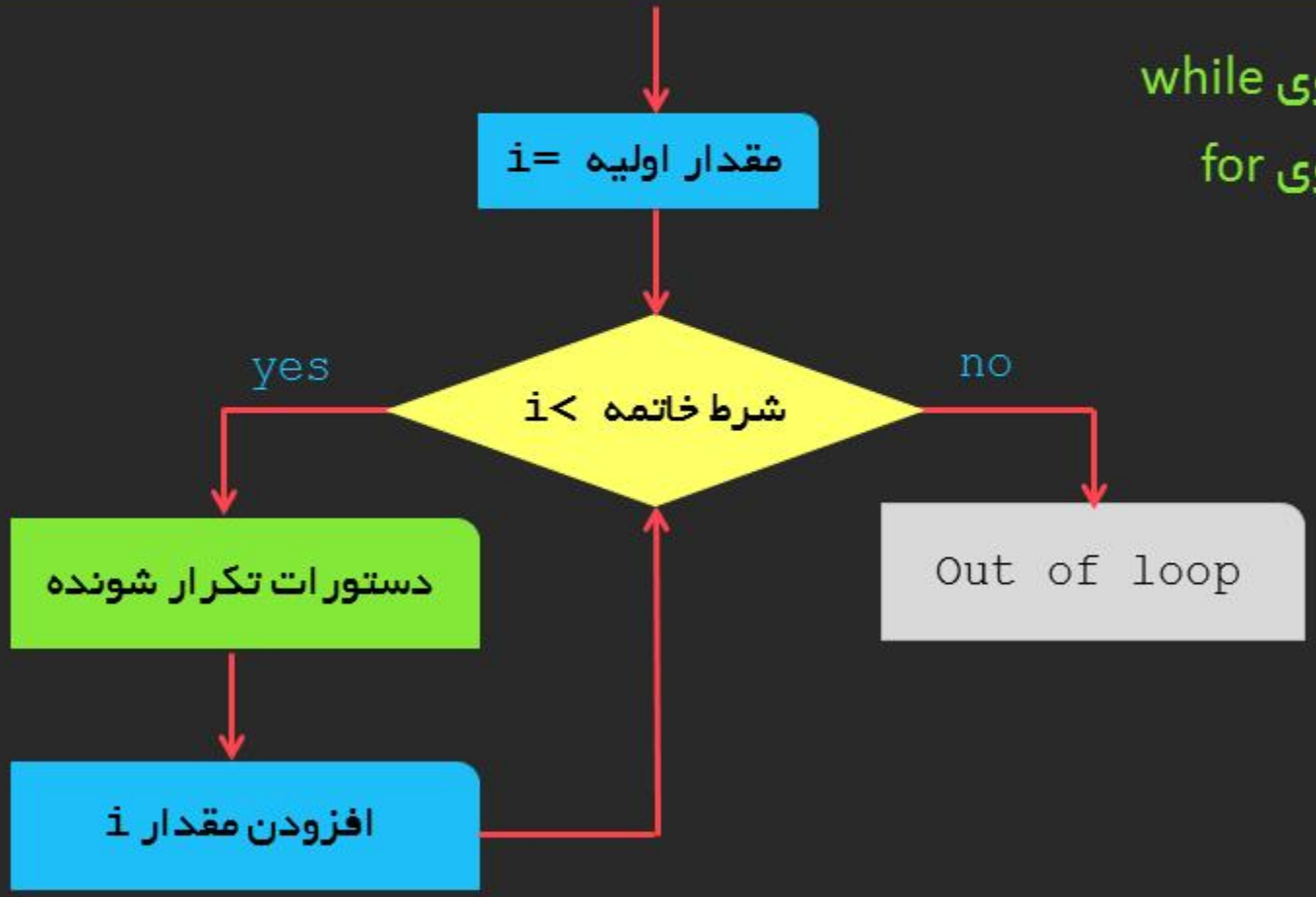
- ساختار تصمیم تو در تو را می توان با ساختار switch … case نیز نوشت:

```
switch (variable){
case value1:
        statement1;
        break;
case value2:
        statement2;
        break;
            …
default:
        statement0;
}
```

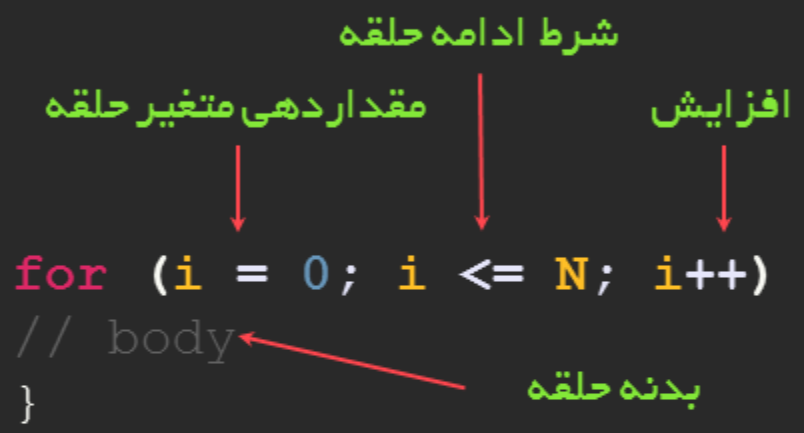
```
switch (a) {
case 1:
        System.out.println("sun");
        break;
case 2:
        System.out.println("mon");
        break;
case 3:
case 4:
        System.out.println("thu or wed");
        break;
default:
        System.out.println("others");
}
```

ساختار حلقه

- الگوی while
- الگوی for



الگوی for



- الگوی for

```
for (init; boolean expression; increment){
      statement1;
      statement2;
}
```

مثال از حلقه for

- چاپ تمامی توان‌های 2 کمتر از n

```
class Programmer {

    public static void main(String[] args) {

        Scanner s = new Scanner(System.in);
        int N = s.nextInt();
        for (int i = 1; i < N; i *= 2) {
            System.out.printf("%d, ", i);
        }
    }
}
```

الگوی while و do-while

- الگوی while

```
while (boolean expression){
        statement1;
        statement2;
}
```

- الگوی do-while

```
do {
        statement1;
        statement2;
} while (boolean expression)
```

مثال از حلقه while

- چاپ تمامی توان‌های 2 کمتر از n

```
class Programmer {

    public static void main(String[] args) {

        Scanner s = new Scanner(System.in);
        int N = s.nextInt();
        int i = 1;
        while (i < N) {
            System.out.printf("%d, ", i);
            i*=2;
        }
    }
}
```

دستورات break و continue

- در هر بلوک از حلقه‌های تکرار زمانیکه لازم باشد از حلقه خارج شویم از دستور break استفاده می شود.

```
while(true){
    i = s.nextInt();
    if (i==0)
        break;
}
```

- در هر بلوک از حلقه‌های تکرار زمانیکه لازم باشد ادامه بدنه اجرا نشده و از تکرار بعدی شروع نماییم، از دستور continue استفاده می شود.

```
for (int i = 1; i < 10; i ++) {
    if (i==4)
      continue;
    System.out.printf("%d, ", i);
}
```

ساختار try-catch-finally

- عدم اجرای صحیح برخی دستورات، منجر به بروز خطا خواهد شد. به کد زیر توجه کنید:

```
public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    int a = s.nextInt();
    System.out.println(a);
    s.close();
}
```

- اگر کاربر بجای وارد کردن عدد در ورودی کاراکتر وارد کند، Exception رخ خواهد داد:

```
u
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Unknown Source)
        at java.util.Scanner.next(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at amir1.Newton.main(Newton.java:27)
```

- هر رویدادی که ساختار اجرایی عادی برنامه را تغییر دهد، یک Exception است.

ساختار try-catch-finally

- جهت مدیریت Exception، می توان دستوراتی که احتمال بروز Exception در آنها وجود دارد را درون ساختار try-catch-finally قرار داد:

```
try{
// do something
}catch (ExceptionType et1){
// do something
}catch (ExceptionType et2){
// do something
}finally {
// cleanup
}
```

دستورات مستعد به بروز Exception

اگر Exception ای از نوع et1 رخ داد

اگر Exception ای از نوع et2 رخ داد

در انتهای ساختار چه Exception رخ دهد چه ندهد

ساختار try-catch-finally

- در مثال زیر، Exception کنترل شده است.

```
public static void main(String[] args) {
    Scanner s = new Scanner( System.in );
    int a = 0;
    try {
        a = s.nextInt();
        System.out.print( "\n you enter: " + a );
        s.close();
    }
    catch ( InputMismatchException ex ) {
        System.out.println( "you must enter a integer number!" );
    }
    catch ( Exception ex ){
        System.out.println( "error: " + ex.toString() );
    }
}
```

برنامه نویسی جاوا

امیر حسین کاشفی

مرور مطالب

- جلسه سوم – آرایه‌ها و متدها در جاوا
  - ساختار آرایه یک بعدی و چند بعدی
  - ساختار یک متد
  - حوزه و دوره حیات
  - ورودی آرایه برای متد
  - اشکال‌زدایی (debug)

آرایه

```java
int N = 5;
// declare
int[] a;
int b[];
// create
a = new int [N];
b = new int [N];
// declare + create
int[] c = new int [N];
int d[] = new int [N];
```

- برای ذخیره سازی حجم زیادی از data از آرایه‌ها استفاده می شود.
- آرایه دنباله‌ایست شاخص‌دار از مقادیر هم نوع
- تعریف آرایه‌ها در جاوا شامل دو مرحله است: اعلان declare و ایجاد create
- مقادیر پیش‌فرض بعد از تعریف آرایه 0 خواهد بود.

آرایه

- مقداردهی به عناصر آرایه می تواند به‌صورت ایستا باشد.

```
String[] months = {
        "Jan", "Feb", "Mar", "Apr", "May", "Jun",
        "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" };
int[] array = {1, 2, 3, 4, 5};
```

- برای دسترسی به عنصر i ام آرایه از array[i] استفاده می‌شود.

array	0	1	2	3	4
	1	2	3	4	5

- طول یا اندازه آرایه به‌صورت array.length مشخص می گردد.

ساختار for ویژه مرور آرایه‌ها

- به مقادیر آرایه می‌توان به‌صورت زیر دسترسی داشت.

```
int[] x = new int[10];
for (int i = 0; i < x.length; i++)
    System.out.println(x[i]);
```

- به مقادیر آرایه می‌توان به‌صورت زیر نیز دسترسی داشت.

```
int[] x= new int [10];
for (int i:x)
System.out.println(i);
```

- دقت شود در این روش تغییرات روی آرایه اعمال نخواهد شد.

آرایه‌های چندبعدی

- آرایه‌های دوبعدی در ریاضیات به‌عنوان ماتریس و در جاوا به‌عنوان آرایه 2d شناخته می شود.

```
int row = 3;
int col = 5;
// declare 2d array
double[][] a;
// create 2d array
a = new double[row][col];
```

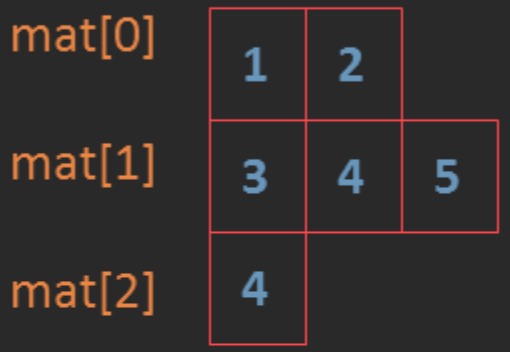
a	0	1	2	3	4
0	0,0	0,1	0,2	0,3	0,4
1	1,0	1,1	1,2	1,3	1,4
2	2,0	2,1	2,2	2,3	2,4

- a[r][c] به معنای سطر r ام و ستون c ام است.

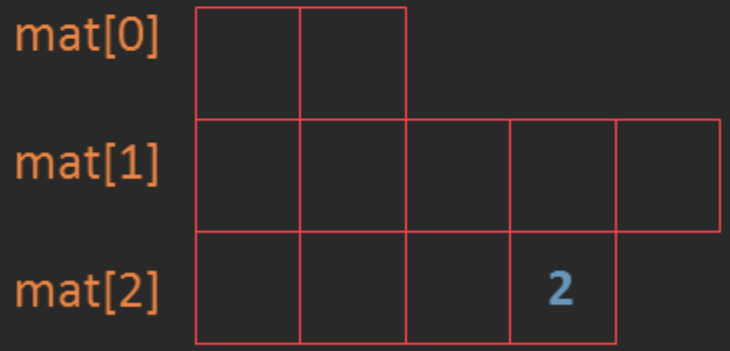
```
// elements are indexed from 0 to n-1
for (int r = 0; r < a.length; r++)
    for (int c = 0; c < a[r].length; c++)
        a[r][c] = 0.0;
```

آرایه‌های دوبعدی نامتوازن

```
int [][] mat = {{1,2},{3,4,5},{4}};
```




```
int [][] mat = new int[3][];
mat[0]=new int[2];
mat[0]=new int[5];
mat[0]=new int[4];
mat[2][3]=2;
```



کلاس آرایه

- یک کلاس مهم در کتابخانه استاندارد جاوا است. که با **import** کلاس **Array** را در برنامه قابل استفاده است:

```
import java.util.Arrays;
```

- برای استفاده از آن، کلاس Arrays را نوشته و از متدهای استاتیک آن استفاده می کنیم:

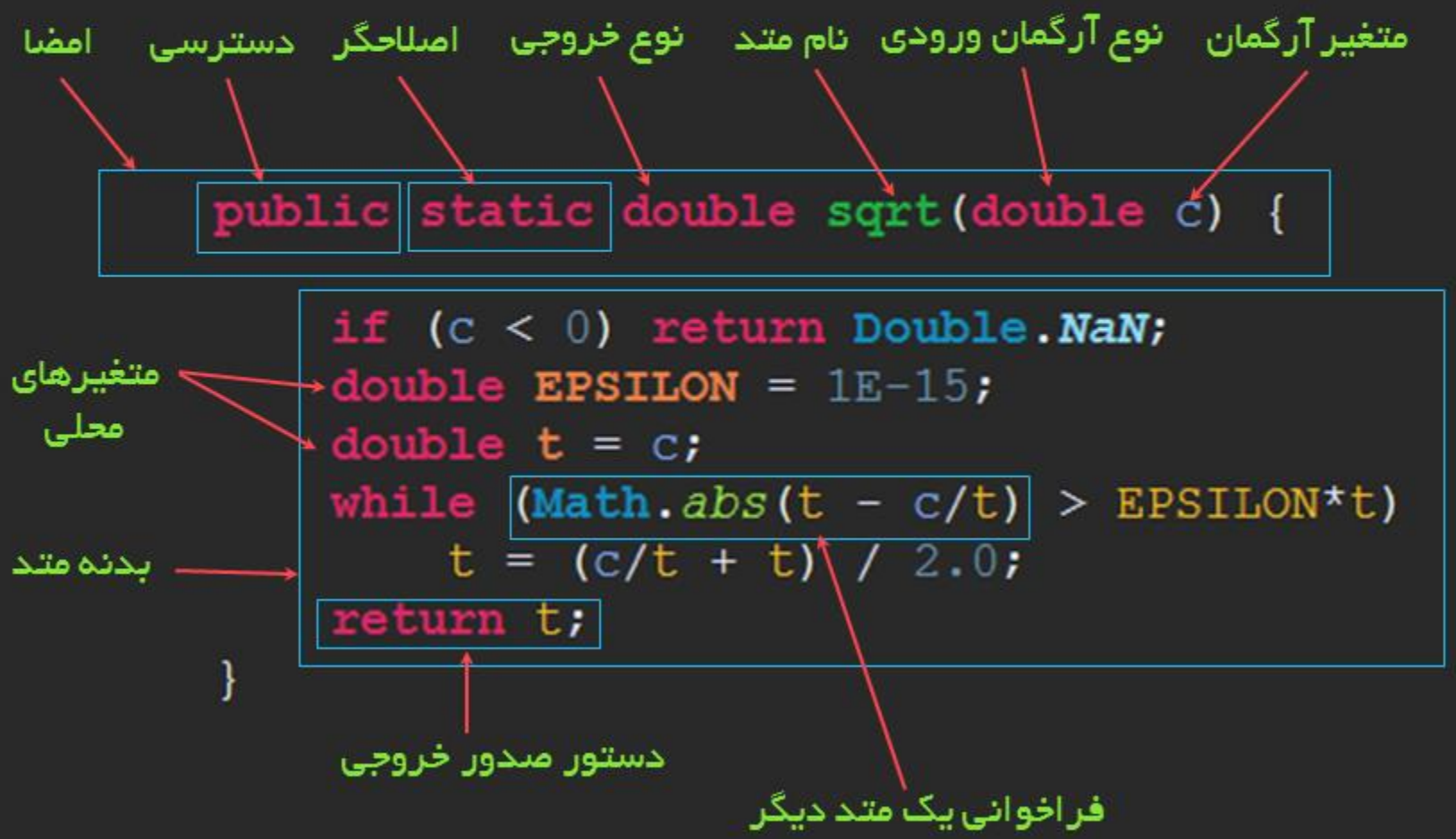
```
int array[] = {2, 5, 78, 89, 4, 24};
```

متد	مفهوم
`arrays.binarySearch(array, item)`	جستجوی item در آرایه مرتب و برگشت جایگاهش
`copyOf(array, length)`	جهت کپی گرفتن از آرایه به اندازه length
`equals(array, array2)`	بررسی برابر بودن دو آرایه
`fill (array, value)`	پرنمودن آرایه با یک مقدار
`sort (array)`	مرتب‌سازی یک آرایه
`toString (array)`	تبدیل عناصر آرایه بطور رشته پشت سرهم

تابع <— متد

- در جاوا توابع دارای 0 یا بیشتر ورودی و یک یا 0 مقدار خروجی هستند.
- برنامه‌نویسان از توابع برای ساخت برنامه ماژولار و محاسبه فرمول‌ها استفاده می کنند.
- Math.random()، Integer.parseInt() نمونه‌هایی از توابع از پیش نوشته شده‌اند.
- main() نمونه‌ای از توابع نوشته شده توسط کاربر است.
- به تابعی که در یک کلاس تعریف می‌شود، متد آن کلاس می گویند.

ساختار یک متد



جریان کنترلی

- توابع می‌توانند جریان کنترلی یک برنامه را تغییر دهند.

```
public class Newton {
    public static double sqrt(double c) {
        if (c < 0) return Double.NaN;
        double EPSILON = 1E-15;
        double t = c;
        while (Math.abs(t - c/t) > EPSILON*t)
            t = (c/t + t) / 2.0;
        return t;
    }
    public static void main(String[] args) {
        double[] a = new double[args.length];
        for (int i = 0; i < args.length; i++)
            a[i] = Double.parseDouble(args[i]);
        for (int i = 0; i < a.length; i++) {
            double x = sqrt(a[i]);
            System.out.println(x);
        }
    }
}
```

Scope and life time

```
public class Newton {
    int a = 7;

    public static int function(int c) {
        int a = 8;
        a++;
        return a;
    }

    public static void main(String[] args) {
        int a = 9;
        for (int i = 0; i < args.length; i++)
            a++;
        int x = function(a);
        System.out.println(x);
    }
}
```

Scope of a, c

Scope of i

Scope of a, x

Scope of a

- life time هر متغیر بازه زمانی است که آن متغیر در حافظه قرار دارد.
- Scope هر متغیر بلوکی است که در آن تعریف شده تنها در آن حوزه قابل شناسایی است.

متدهای هم‌نام

- می‌توان چندین متد با یک نام و امضا های متفاوت (انواع/تعداد ورودی) در یک کلاس تعریف نمود که به این امر method overloading گویند.

```
public class Newton {

    public static int function(int a) {
        return ++a;
    }
    public static int function(int a, int b) {
        return a+b;
    }
    public static int function(double a) {
        return (int)a;
    }
}
```

- در جاوا امکان تعریف متدهای هم نام با خروجی‌های متفاوت وجود ندارد.

تابع با ورودی آرایه

- ورودی توابع می تواند آرایه باشد.

```
public static int max(int[] a) {
    int max = a[0];
    for (int i = 1; i < a.length; i++)
        if (a[i] > max)
            max = a[i];
    return max;
}

public static void main(String[] args) {
    int x[] = { 2, 5, 78, 89, 4, 24 };
    int m = max(x);
    System.out.println(m);
}
```

تابع با ورودی متغیر

- می‌توان یک لیست متغیر را به عنوان ورودی به تابع داد.

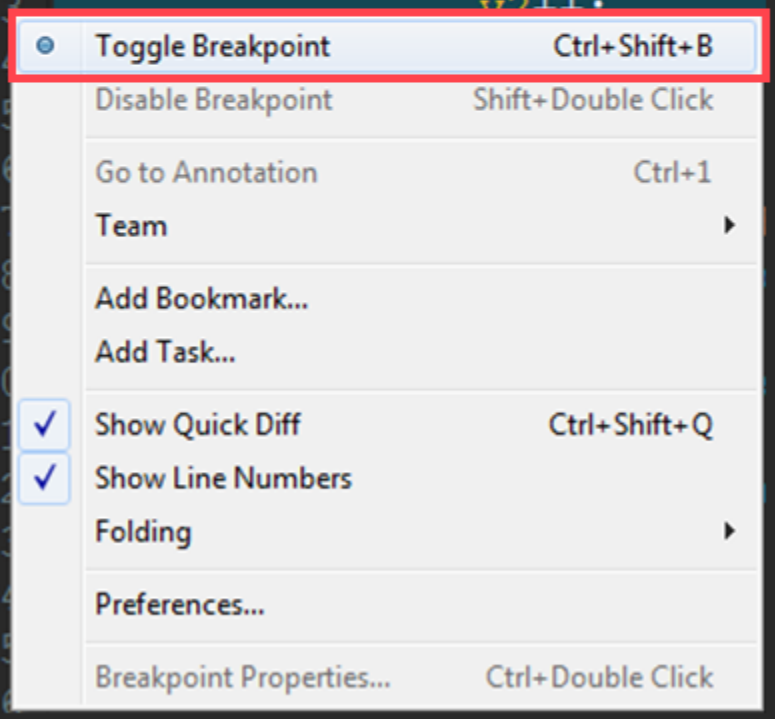
```
public static int max(int... a) {
    int max = a[0];
    for (int i = 1; i < a.length; i++)
        if (a[i] > max)
            max = a[i];
    return max;
}

public static void main(String[] args) {
    int m = max(2, 5, 78, 89, 4, 24);
    System.out.println(m);
}
```

ارجاعات در تابع

- در مورد انواع primitive مانند انواع int، String، float و ... جاوا به صورت call by value عمل می کند.
- در مورد غیر primitiveها مانند آرایه‌ها، چون Objectهای جاوا ارجاع به خانه‌های حافظه هستند، با ارسال یک Object، مقدار آدرس حافظه است که درون پارامتر تابع کپی می شود بنابراین به صورت call by reference عمل می کند.
- الگوریتم و قواعد JVM برای Garbage Collection است که مانع گرفتن Reference / Offset می‌شود.

عیب یابی (debug)

```
109         int s2=45;
110         int y2=90;
111         s2=s2+y2;
112         s2--;
```



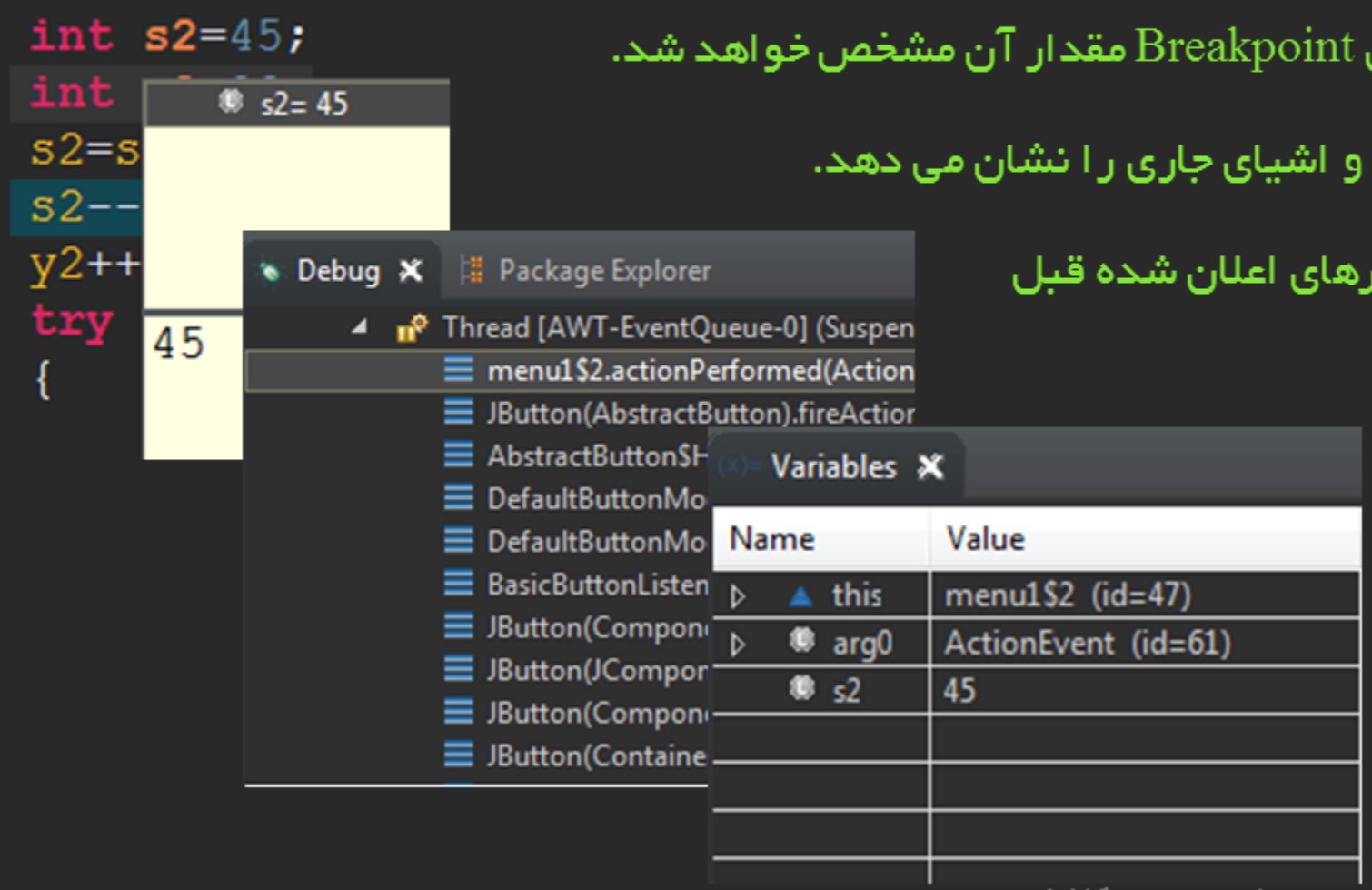
- برای این منظور از نقطه شکست ضامن (toggle breakpoint) استفاده می کنیم.
- روی خطی از برنامه جهت عیب‌یابی کلیک و Toggle Breakpoint را انتخاب می کنیم:
- سپس بجای Run As از Debug As استفاده می کنیم.
- برای ردگیری از دکمه‌های زیر استفاده می‌شود:

Step Into (F5) 

Step Over (F6) 

Step Return (F7) 

عیب یابی (debug)



- با نگهداشتن نشانگر موس روی متغیر های قبل Breakpoint مقدار آن مشخص خواهد شد.
- پنجره Debug نشان دهنده پشته متدها، تردها و اشیای جاری را نشان می دهد.
- پنجره Variables نشان دهنده مقدار تمام متغیرهای اعلان شده قبل Breakpoint است.

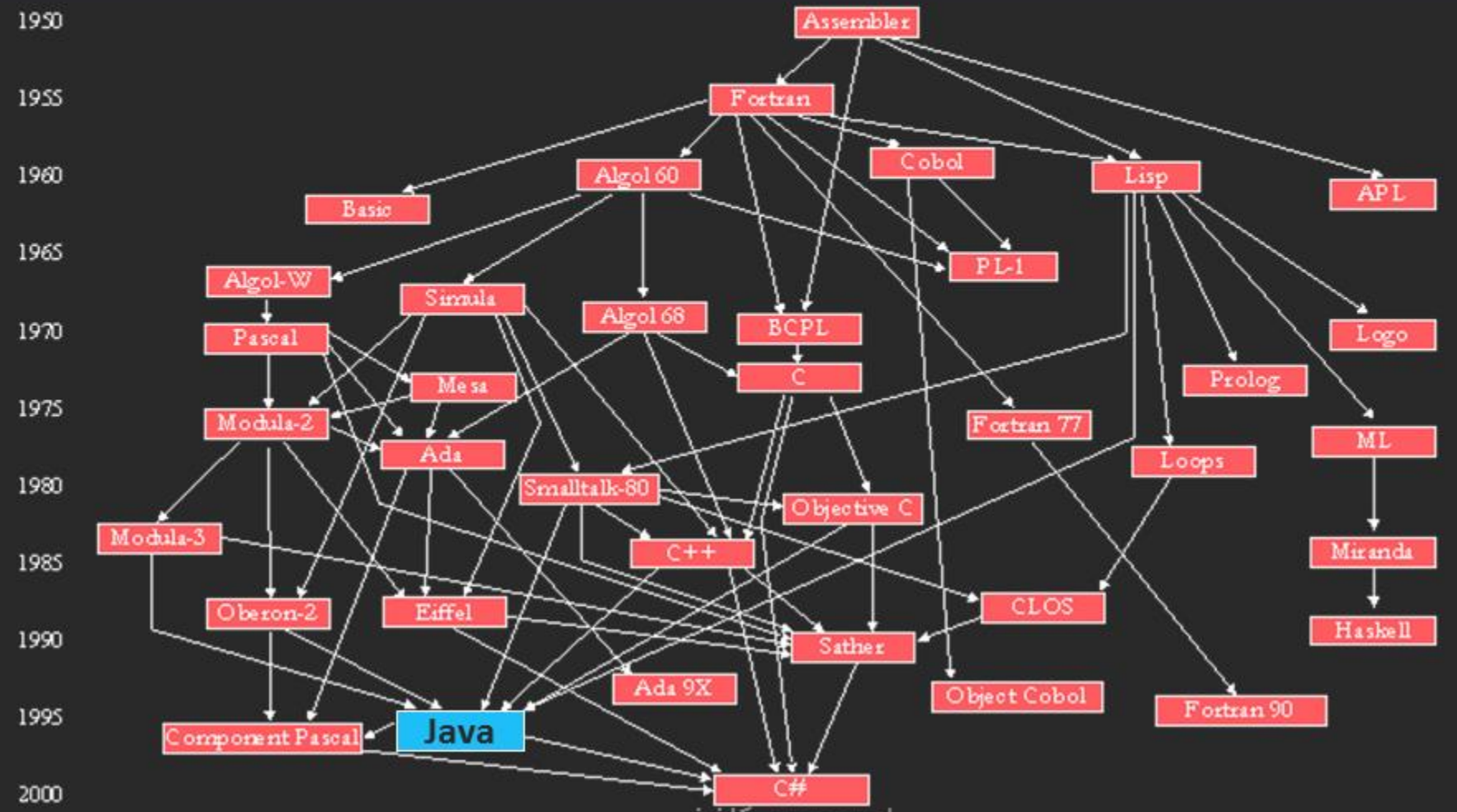
برنامه نویسی جاوا

امیر حسین کاشفی

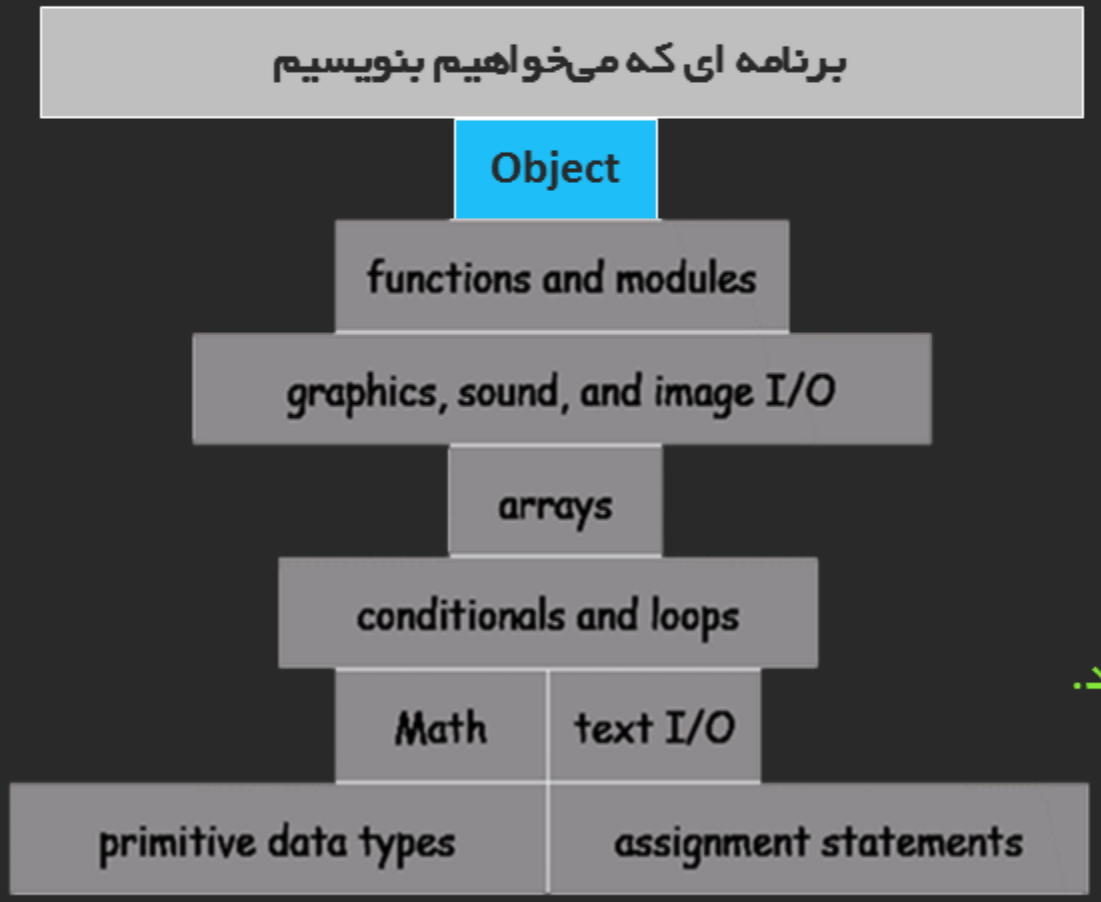
مرور مطالب

- جلسه چهارم – اشیاء، کلاس و شیء گرایی
  - ساختار یک کلاس (class)
  - تعریف یک شیء (object)
  - اصلاحگرها (modifiers)
  - سازنده کلاس (constructor)
  - اشاره گر this

شجره‌نامه زبان‌های برنامه نویسی



شیءگرایی



- سوال اصلی در رویکرد Procedural:

  What does this program do?

- سوال اصلی در رویکرد Object Oriented:

  What real world objects am I modeling?

- در برنامه نویسی شیء گرا، شیء بالاترین سطح انتزاع است.

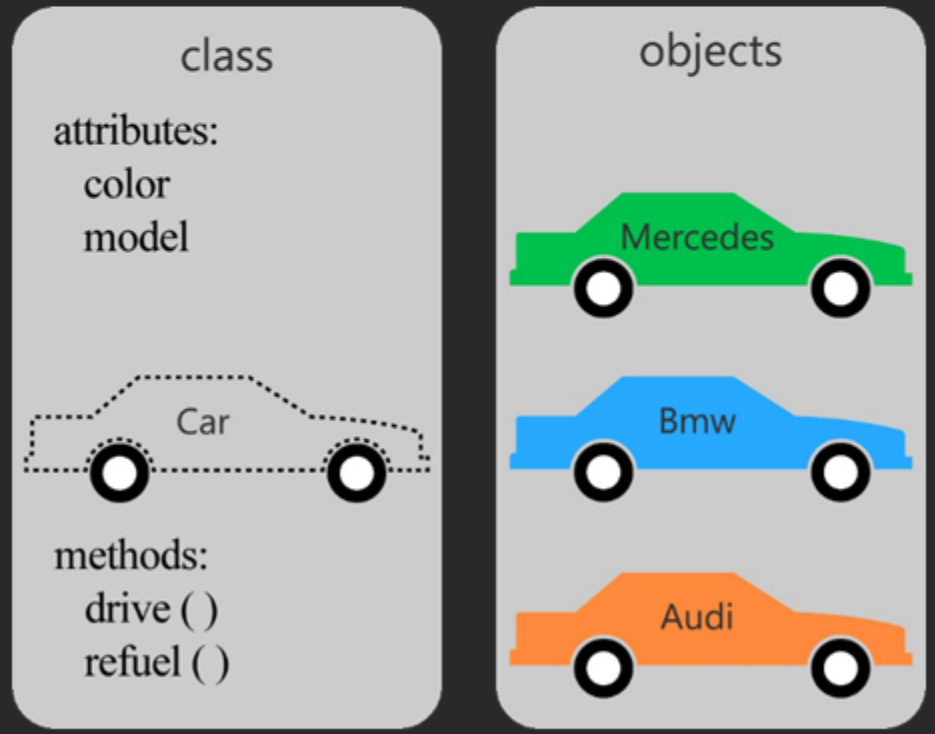
- برنامه با استفاده از اشیاء و تعامل آن‌ها با یکدیگر مدل می شود.

شیء

- یک برنامه مجموعه ای از اشیاء است که با ارسال پیام به هم، به یکدیگر می‌گویند چه کاری انجام دهند.
- هر آنچه در دنیای واقعی دارای چندین ویژگی و رفتار است، در برنامه‌نویسی شیءگرا، شیء یا Object به شمار می‌آید.
- هر شیء، حافظه اختصاص یافته به خود را دارد و دارای یک نوع است.
- همه اشیای هم‌نوع می‌توانند پیام های مشابه دریافت کنند.
- به عنوان مثال، تلفن همراه شما یک شیء است که دارای نوع « تلفن همراه » است.
  - ویژگی‌های این شئ : مدل، رنگ، جنس، وزن و ...
  - رفتارهای این شیء : پخش موزیک، برقراری تماس، ارسال پیامک و ...

کلاس

- اشیا را می‌توان با توجه به مشخصات و رفتار آن‌ها دسته بندی کرد که به این دسته‌ها کلاس یا Class می‌گویند.
- کلاس نوع داده پیشرفته در java به حساب می‌آید که مانند Arrays از پیش تعریف شده یا می‌توانند توسط برنامه‌نویس تعریف شوند.



- کلاس، نوع یک شیء را مشخص می کند

و دارای الگویی برای تعدادی ویژگی و رفتار است.

- به نمونه‌های ایجاد شده از هر کلاس شیء گفته می‌شود.

کلاس و شیء

- اشیای تعریف شده از یک کلاس می توانند به ویژگی‌های تعریف شده در کلاس مقدار دهند و از رفتارهای تعریف شده در کلاس استفاده کنند.
- مثلا کلاس حیوان با ویژگی‌های (قد / وزن) و رفتارهای (راه رفتن / غذاخوردن)
- شی زرافه (قد=۲۵۰ / وزن=۴۵۰) با قابلیت های رفتاری (راه رفتن / غذاخوردن)
- هر ویژگی در یک کلاس را « صفت » یا Attribute می‌نامند و برای پیاده سازی آن‌ها در کلاس از « متغیرها » استفاده می شود.
- هر رفتار در یک کلاس را « متد » یا Method می‌نامند و برای پیاده سازی آن‌ها در کلاس از « تابع‌ها » استفاده می شود.

Java packages

- جهت مدیریت کلاس‌ها، اجتناب از تداخل نام‌گذاری و کنترل دسترسی استفاده می شود از package استفاده می شود.
- برخی از package ها مهم در جاوا عبارتند از java.lang، java.io، java.util و ...
- اگر ازهیچ packageای استفاده نشود، کلاس داخل package پیش‌فرض قرار می‌گیرد.
- packageها را می توان بصورت متداخل با استفاده از . تعریف نمود. بنابراین تعریف یک کلاس در یک package به صورت زیر است:

```
package packagenameL1.packagenameL2.packagenameL3 ;
```

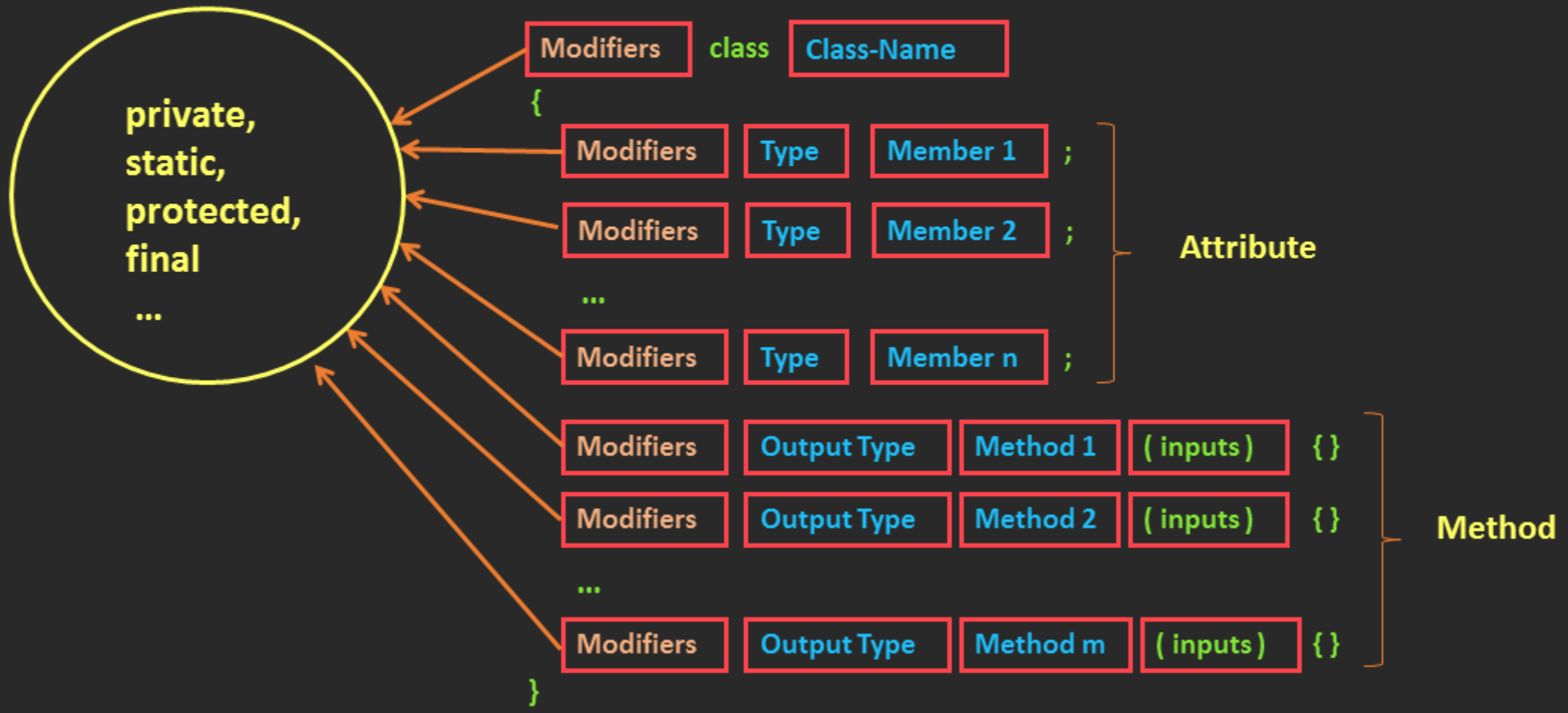
- برای اینکه بتوان از کلاس‌های خارج از package جاری استفاده کرد، می‌توان از دستور زیر استفاده کنید:

```
import packagename.ClassName;
```

- برای استفاده از کلیه کلاس‌های یک package از * استفاده می شود:

```
import packagename.*;
```

ساختار کلاس


private,
static,
protected,
final
...
Modifiers
class
Class-Name
{
Modifiers
Type
Member 1
;
Modifiers
Type
Member 2
;
...
Modifiers
Type
Member n
;
Attribute
Modifiers
Output Type
Method 1
(inputs)
{ }
Modifiers
Output Type
Method 2
(inputs)
{ }
...
Modifiers
Output Type
Method m
(inputs)
{ }
Method
}

مثالی از یک کلاس

```
public class Student {
    private String name;
    private int id;
    public int getId() {
        return id;
    }
    public String getName() {
        return name;
    }
    public void setInfo(String n, int i) {
        name = n;
        id = i;
    }
    public String toString() {
        return ("student's name is" + name + "Id is:" + id);
    }
}
```

تعریف شیء نمونه از کلاس

- تعریف یک نمونه از کلاس به صورت زیر است:

نام کلاس نام شیء = new نام کلاس ( ) ;

- البته می‌توان ابتدا اعلان و سپس ایجاد را انجام داد:

نام کلاس نام شیء ;

نام شیء = new نام کلاس ( ) ;

- ساخت یک نمونه از کلاس برای دستیابی به عناصر کلاس (متغیرها/متدها) از . استفاده می‌شود:

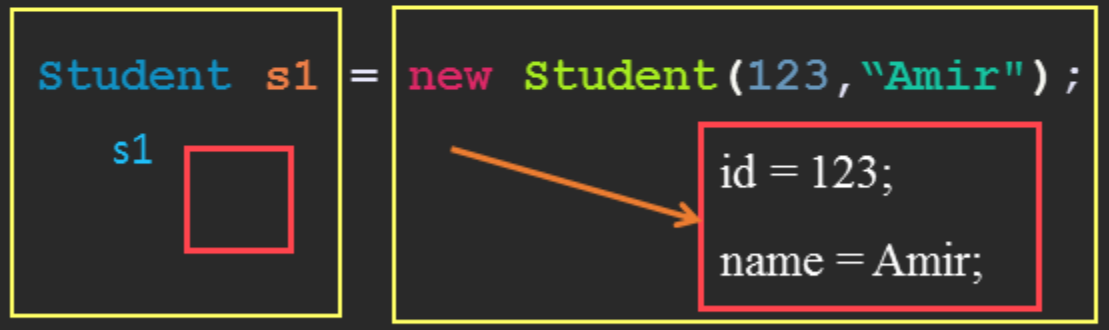
نام شیء . متغیر ;

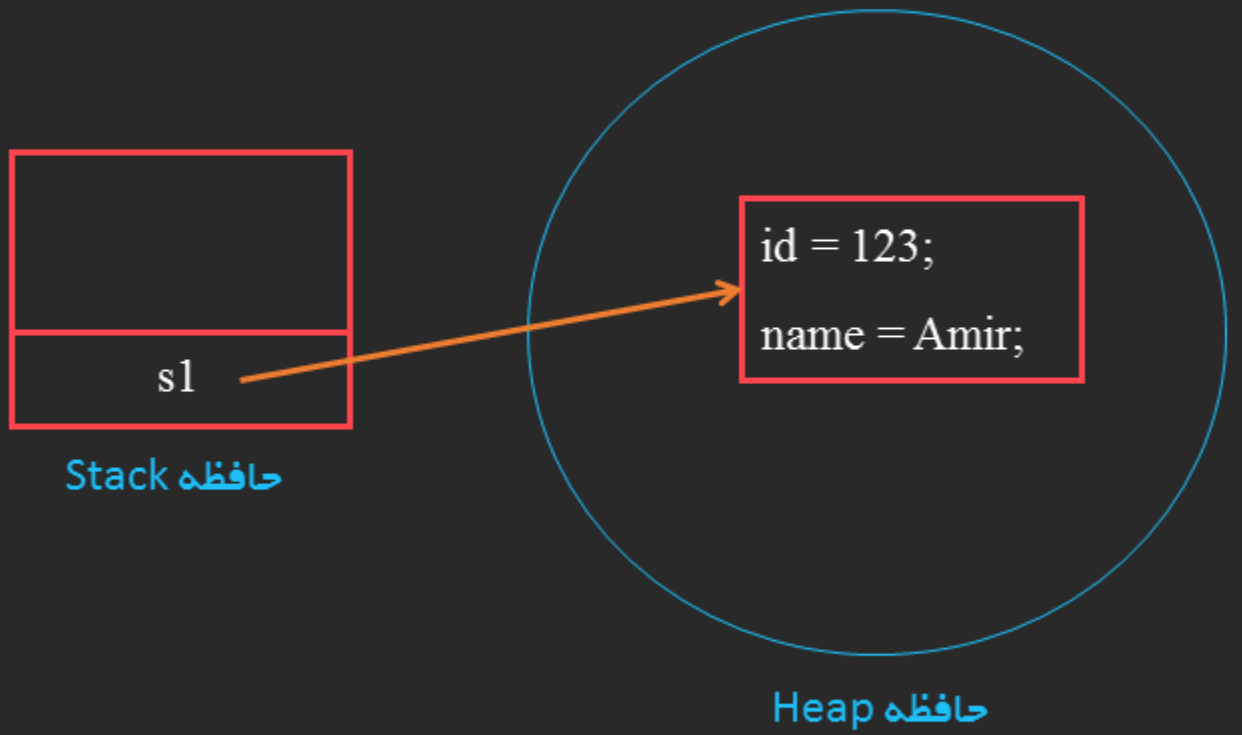
نام شیء . متد (inputs) ;

new نام کلاس ()

حافظه مورد نیاز برای اشیاء

```
public class Student {
    private String name;
    private int id;
}
```





modifiers

- modifier ها کلمات کلیدی هستند که برای تغییر معنا به ابتدای تعریف کلاس، متد یا متغیر اضافه می شوند.
- modifier ها در جاوا به دو دسته تقسیم می شوند:

Java Access Modifier's

Java Non Access Modifier's

```
public class Myclass {
    private boolean myFlag;
    static final double weeks = 9.52;
    protected static final int BOXWIDTH = 42;
    public static void main(String[] arguments) {
        // body of method
    }
}
```

Access Control Modifiers

- بیانگر سطح دسترسی به متغیر / کلاس / متد است. و به انواع زیر تقسیم می‌شود:

```
private,  default,  protected,  public
```

- default یا پیشفرض به معنای عدم استفاده از آن‌هاست و تنها در داخل package تعریف شده قابل دید است.
- private به معنای آن است که تنها در داخل کلاس تعریف شده قابل دید است.
- protected به معنای آن است که در داخل هر کلاس ارث برده از کلاس تعریف شده قابل دید است.
- public به معنای آن است که همه جا قابل دید است.

Non Access Control Modifiers

- بیانگر اهدای عملکرد خاص به متغیر یا کلاس یا متد است که مهمترین آنها عبارتست از:

`static, final, abstract, transient, synchronized, volatile`

ویژگی	متغیر	متد	کلاس	مفهوم
static	✓	✓	✓	هر موجودیت static برای کلیه نمونه های کلاس تنها یکی است.
Final	✓	✓	✓	هر موجودیت final فقط یکبار مقداردهی می شود و تغییر پذیر نمی باشد.
abstract	✗	✓	✓	کلاس abstract قابل نمونه سازی نیست و متد abstract تنها توسط subclass ها قابل پیاده سازی است.
transient	✓	✗	✗	جهت استفاده متغیرها بطور موقت در ارسال در شبکه کاربرد دارد.
synchronized و volatile	✗	✓	✗	برای threadها استفاده می شوند.

CamelCase in java naming conventions

- در کلیه اسامی چند کلمه ای حرف اول کلمات دوم به بعد بزرگ می باشد.

class name	با حروف بزرگ آغاز گردد مانند: String, Color, Button, System, ...
interface name	با حروف بزرگ آغاز گردد و صفت نام‌گذاری شود مانند: Runnable, Remote, ActionListener, ...
method name	با حروف کوچک آغاز گردد و فعل نام‌گذاری شود مانند: actionPerformed(), main(), print(), println() , ...
variable name	با حروف کوچک آغاز گردد مانند: firstName, orderNumber , ...
package name	کلیه حروف آن کوچک باشد مانند: java, lang, sql, util, ...
constants name	کلیه حروف آن بزرگ باشد مانند: RED, YELLOW, MAX_PRIORITY, ...

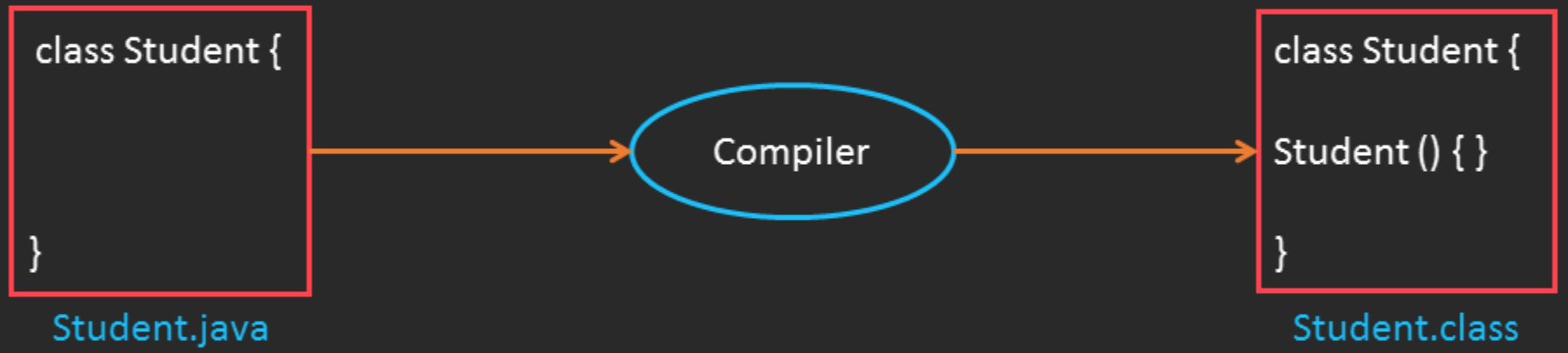
سازنده یا constructor

- متد ویژه‌ای است از یک کلاس است که معمولا برای مقداردهی به مقادیر object تعریف شده از آن کلاس بکار می رود.
- متد Constructor هم نام کلاس بوده و مقدار بازگشتی ندارد.
- Constructor در زمان ایجاد شیء فراخوانده شده و داده ها را برای شیء فراهم می سازد.
- متد Constructor به صورت پیش فرض بدون ورودی تعریف می شود اما می تواند با پارامتر ورودی نیز تعریف شود.

```
public class Student {
   String name;
   int id;
   public Student() {
      System.out.println("Student is created");
   }
   public static void main(String[] args) {
      Student s = new Student();
   }
}
```

سازنده یا constructor

- اگر در کلاس Constructor تعریف نشود، یک Constructor پیش‌فرض تعریف می گردد.



- Constructor پیش‌فرض مقادیر 0 و NULL را برای ویژگی‌های اشیاء ایجاد می کنند.

constructor overloading

- Constructor می تواند overload شود یعنی تعدادی constructor همنام با امضاهای متفاوت تولید شود.

```
class Student {
        int id;
        String name;
        int age;
        Student(int i, String n) {id = i; name = n;}
        Student(int i, String n, int a) {id = i; name = n; age = a;}
        void display() {
                System.out.println(id + " " + name + " " + age);
        }
        public static void main(String[] args){
                Student s1 = new Student(2017,"kashefism");
                Student s2 = new Student(1985,"nikita",25);
                s1.display();
                s2.display();
        }
}
```

```
2017 kashefism 0
1985 nikita 25
```

copy constructor

- جهت کپی نمودن اطلاعات یک object به objectدیگر، می توان به صورت زیر عمل نمود.

```
class Student {
        int id;
        String name;
        Student(int i, String n) {id = i;name = n;}
        Student(Student s) {id = s.id;name = s.name;}
        void display() {
                System.out.println(id + " " + name);
        }
        public static void main(String[] args){
                Student s1 = new Student(2017,"kashefism");
                Student s2 = new Student(s1);
                s1.display();
                s2.display();
        }
}
```

```
2017 kashefism
2017 kashefism
```

اشاره‌گر this

- در جاوا اشاره‌گر this یک متغیر ارجاعی به object جاری است.
- this را می توان در موارد زیر به کاربرد:
  - فراخوانی اشیاء، متدها یا سازنده کلاس جاری
  - ورودی یا خروجی یک تابع
  - ورودی یک سازنده

اشاره‌گر this

- این کلمه کلیدی می تواند برای رجوع به شیء کلاس جاری باشد.

```
class Student {
	int id;
	String name;
	Student(int id, String name) {this.id = id; this.name = name;}
	void display() {
		System.out.println(id + " " + name);
	}
	public static void main(String[] args){
		Student s1 = new Student(2017,"kashefism");
		Student s2 = new Student(1985,"nikita",25);
		s1.display();
		s2.display();
	}
}
```

اشاره‌گر this

- This() می تواند برای فراخوانی constructor کلاس جاری استفاده شود:

```
class Student {
        int id;
        String name;
        int age;
        Student(int id, String name) {this.id = id; this. name = name;}
        Student(int id, String name, int age) {this(id , name); this.age = age;}
        void display() {
                System.out.println(id + " " + name + " " + age);
        }
        public static void main(String[] args){
                Student s1 = new Student(2017,"kashefism");
                Student s2 = new Student(1985,"nikita",25);
                s1.display();
                s2.display();
        }
}
```

اشاره‌گر this

- This() می تواند برای فراخوانی یک متد از کلاس شیء جاری استفاده شود:

```
class Student {
	int id;
	String name;
	Student(int id, String n) {this.id =  id; name = n;}
	public void f1() { this.display();}
	void display() {
		System.out.println(id + " " + name);
	}
	public static void main(String[] args){
		Student s1 = new Student(2017,"kashefism");
		s1.f1();
	}
}
```

اشاره‌گر this

- This می تواند به عنوان آرگمان ورودی به متد دیگر ارسال شود یا به عنوان خروجی متد دیگر در نظر گرفته شود:

```
class MyClass {
    void smile(MyClass obj) {
        System.out.println(":D");
}

    void p() {
        smile(this);
    }

    public static void main(String args[])
    {
        MyClass s1 = new Myclass();
        s1.p();
    }
}
```

```
class MyClass {
    public MyClass m() {
        return this;
    }
    void smile() {
        System.out.println(":D");;
    }
}
```

```
class Test {
    public static void main(String args[])
    {
        new MyClass().m().smile();
    }
}
```

اشاره‌گر this

- کلمه کلیدی this می تواند به constructor نیز ارسال شود. این امر زمانی مفید است که از یک object در چندین کلاس استفاده شود.

```
class MyClass {
   OtherClass obj;

   MyClass(OtherClass obj) {
      this.obj = obj;
   }
   void display() {
      System.out.println(obj.data);
      // using data member of A class
   }
}
```

```
public class OtherClass {
   int data = 10;

   OtherClass() {
      MyClass m = new MyClass(this);
      m.display();
   }
   public static void main(String args[]){
      OtherClass o=new OtherClass();
   }
}
```

برنامه نویسی جاوا

امیر حسین کاشفی

مرور مطالب

- جلسه پنجم – ادامه مباحث شیء گرایی
  - بلوک‌های خالی و استاتیک
  - method overloading
  - اصلاحگر Static
  - اصلاحگر Final
  - مفهوم Enum
  - اصول شیء گرایی

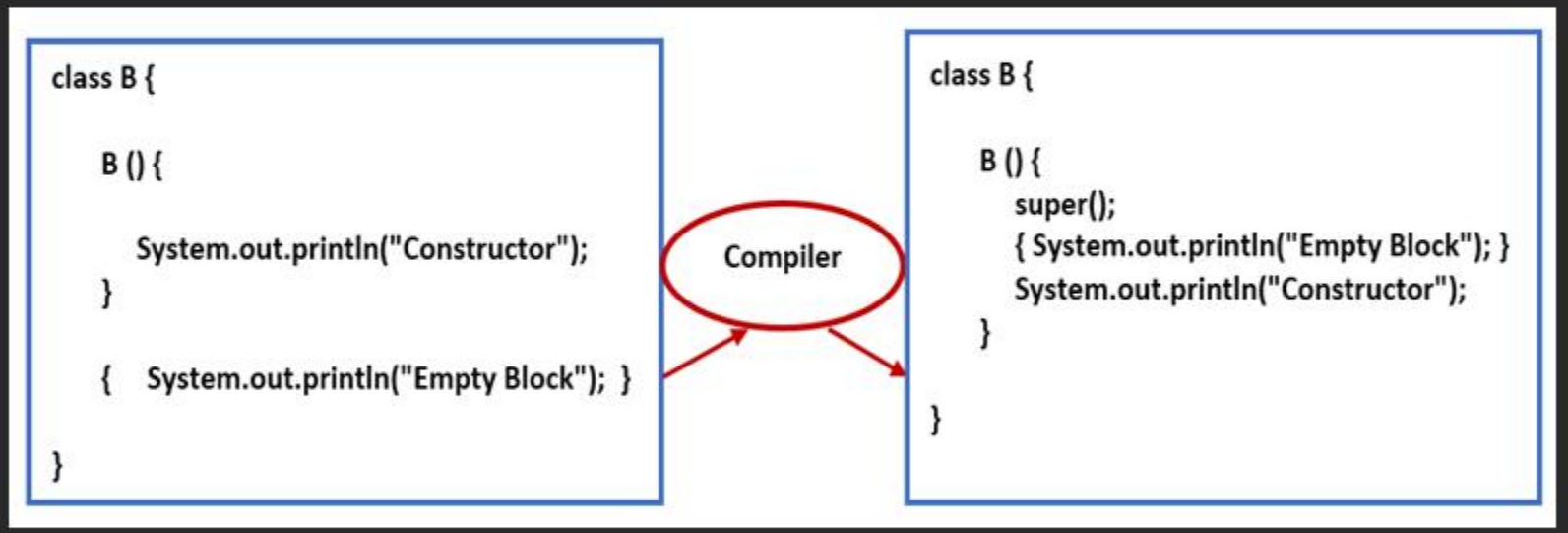
بلوک‌های خالی و استاتیک

- اگر چندین سازنده در کلاس وجود داشته باشد و بخواهیم قبل از همه آنها مقداردهی به اعضای کلاس انجام شود، از این بلوک‌ها استفاده می شود.
- تفاوت بلوک خالی و بلوک استاتیک در این است که بلوک استاتیک در زمان اعلان و بلوک خالی در زمان ایجاد نمونه از کلاس اجرا می شوند.

```
public class Test {
   int speed;
   Test(){System.out.println(speed);}
   { speed=100;
     System.out.println("Empty block");  }
   static {  System.out.println("Static block");  }

   public static void main(String[] args) {
      Test t;
      System.out.println("============");
      t = new Test();
   }
}
```

```
Static block
============
Empty block
100
```

بلوک‌های خالی و استاتیک



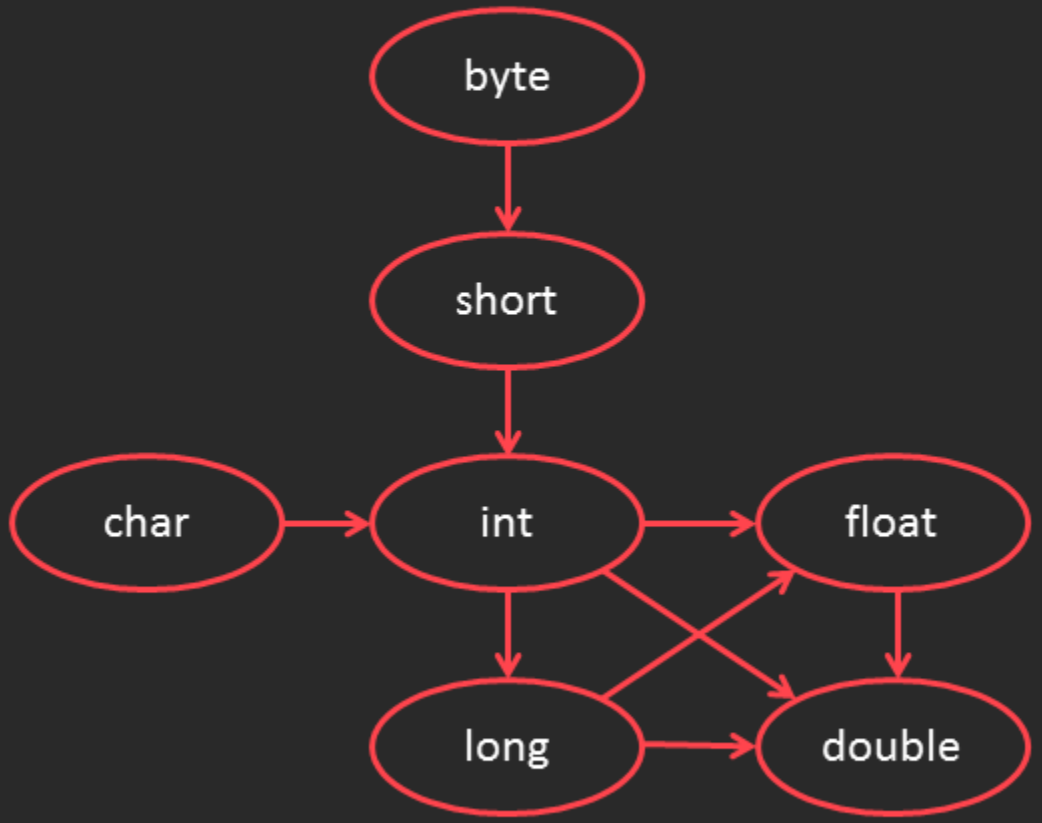
Method Overloading

- متدهای همنام با امضاهای مختلف (تعداد و نوع پارامترها) در یک کلاس را Method Overloading گویند.
- Method Overloading قابلیت خوانایی برنامه را بالا خواهد برد.
- در جاوا تغییر نوع مقدار خروجی یا return type مقدور نیست زیرا منجر به ابهام خواهد شد.

```
class Calculation {
  int sum(int a, int b) {
    return(a + b);
  }
  double sum(int a, int b) {
    return(a + b);
  }
  public static void main(String args[]) {
    Calculation obj = new Calculation();
    int result = obj.sum(20, 20); // Compile Time Error
  }
}
```

ترقی نوع

```
class Calculation {
    void sum(int a, long b) {
        System.out.println(a + b);
    }
    void sum(int a, int b, int c) {
        System.out.println(a + b + c);
    }
    public static void main(String args[]) {
        Calculation obj = new Calculation();
        obj.sum(20, 20);
        // second int promoted to long
    }
}
```



اصلاحگر Static

```
class Student{
  int rollno;   String name;
static String college ="ITS";

  Student(int r,String n)
{  rollno = r;     name = n;  }

  void display ()
 {System.out.println(rollno+" "+name+" "+college);}
  public static void main(String args[]){
      Student s1 = new Student (111,"Karan");
      Student s2 = new Student (222,"Aryan");
      s1.display();
      s2.display();
  }
}
```

- Static در جاوا برای مدیریت حافظه بکار می رود.
- متغیر static در هر کلاس به یک ویژگی یکسان بین تمامی اشیا اشاره دارد و بین تمامی اشیا به اشتراک گذاشته می شود.
- برای متغیر static تنها یکبار در محیط کلاس در زمان بارگزاری کلاس برمی گرداند.

اصلاحگر Static

```
class Student{
    int rollno;    String name;
     static String college = "ITS";
    static void change(){ college = "BBDIT"; }
    Student(int r, String n){    rollno = r;    name = n;  }
    void display ()
    {System.out.println(rollno+" "+name+" "+college);}
     public static void main(String args[]){
          Student.change();
           Student s1 = new Student (111,"Karan");
          Student s2 = new Student (222,"Aryan");
          s1.display();
          s2.display();
    }
}
```

- متد static به کلاس تعلق دارد نه object.
- یک متد static می تواند بدون نیاز به ایجاد یک نمونه از کلاس فرا خوانده شود. بنابراین تنها کافیست از الگوی زیر استفاده شود:

  نام کلاس. نام متد static
- متدهای static می توانند به مقادیر متغیر های static دسترسی داشته و آن‌ها را تغییر دهند

محدودیت‌های static

- متد static نمی تواند از متغیرهای غیر static استفاده کند و یا متدهای غیر استاتیک کلاس را فراخوانی کنند.
- کلمات کلیدی this و super نمی توانند در متد static استفاده شوند.
- چرا متد main() باید static تعریف شود؟
- علت این امر این است که object لازم نیست تا یک متد static را فراخوانی نماید. اگر static main() نبود، jvm ابتدا object را تولید و سپس متد main() را فراخوانی می‌کرد که حافظه زیادی مصرف می گردید.

اصلاحگر Final

- جهت محدودکردن کاربر از این کلمه کلیدی استفاده می شود.
- این کلمه کلیدی قبل از متغیر، متد یا کلاس می تواند به کار رود.
- متغیر final دارای مقدار ثابت است و نمی توان مقدار آن را تغییر داد. این مقدار باید یا در زمان اعلان آن یا در سازنده مشخص شود.
- متغیر final را می‌توان در بلوک خالی نیز مقداردهی نمود و برای مقداردهی آن در بلوک استاتیک، باید متغیر final static باشد.
- اگر پارامتری در یک تابع را نیز final تعریف نماییم مقدار آن قابل تغییر نیست.

اصلاحگر Final

- متدی که final تعریف شود، قابل override شدن نمی باشد.
- آیا میتوان سازنده را final تعریف نمود ؟
- کلاسی که final تعریف شود، قابل extends شدن نمی باشد. برای مثال کلاس String در جاوا غیر قابل تغییر است، بنابراین finalتعریف شده است.
- اگر یک کلاس final تعریف شود تمامی متدهایش هم finalمی شود.
- آیا می توان یک متد final را به ارث برد ؟

Java Final Keyword

-> Stop Value Change
-> Stop Method Overriding
-> Stop Inheritance

Enum در جاوا

- enum یک نوع داده dataType یا کلاس در جاواست که شامل تعداد ثابتی از Constant هاست.

```
enum Season {
WINTER, SPRING, SUMMER, FALL
}
```

- اعضای یک enum همگی بطور ضمنی، final و static هستند.

- enum می تواند مستقل یا در داخل یک کلاس دیگر تعریف شود.

```
class Myclass {
    enum Season {
    WINTER, SPRING, SUMMER, FALL
    }

    public static void main(String[] args) {
        Season s = Season.WINTER;
        System.out.println(s);
    }
}
```

Enum در جاوا

- با تعریف یک enum کامپایلر متد values() را به طور ضمنی به enum اضافه میکند که آرایه ای از مقادیر enum را بر می‌گرداند.
- enum می تواند دارای field، constructor و یا حتی متد باشند.
- اعضای داخل enum به ترتیب value ... 2 , 1 , 0 به خود می‌گیرند. اما می‌توان برای enum یک یا چند مقدار تعریف نمود با field و constructor به آن‌ها مقدار داد.

```
enum Car {
    Samand(30), pride(20), Tiba(24);
    private int price;

    Car(int p) {
        price = p;}
    int getPrice() {
        return price;}
}
```

```
class class2 {
    public static void main(String[] args) {
        System.out.println("All car prices:");
        for (Car c : Car.values())
            System.out.println(c + "->" + c.getPrice() + " m");
        }
}
```

Enum در جاوا

- آنچه کامپایلر با تعریف enum ایجاد می کند به شرح زیر است:

```
enum Season {
    WINTER(10), SUMMER(20);
    private int value;

    Season(int value) {
        this.value = value;
    }
}
```

```
final class Season extends Enum {
    public static Season[] values() {
        return (Season[])$VALUES.clone();}

    public static Season valueOf(String s) {
        return (Season) Enum.valueOf(Season, s);}

    private Season(String s, int i, int j) {
        super(s, i);
        value = j;}

    public static final Season WINTER;
    public static final Season SUMMER;
    private int value;
    private static final Season $VALUES[];
    static {
        WINTER = new Season("WINTER", 0, 10);
        SUMMER = new Season("SUMMER", 1, 20);
        $VALUES = (new Season[] { WINTER, SUMMER });
    }
}
```

اصول شیء گرایی

تجرید (Abstraction)

- پنهان‌سازی جزئیات درونی و نمایش قابلیت‌ها. مانند تلفن‌همراه.
- در جاوا از abstract class و interface برای رسیدن به تجرید استفاده می شود.
- برای پیاده‌سازی اینترفیس در جاوا از کلمه implement استفاده می‌شود.

بسته‌بندی (Encapsulation)

- پوشاندن کدها و داده‌های مرتبط، در قالب یک بسته یا کپسول را بسته‌بندی گویند.
- در جاوا class نمونه‌ای از یک بسته است.

اصول شیء گرایی

چندریختی(Polymorphism)

- انجام یک وظیفه با شیوه‌های مختلف را چندریختی گویند.
- در جاوا از method overloading و method overloading برای حصول چندریختی استفاده می‌شود.

وراثت (Inheritance)

- زمانیکه یک شیء به تمامی خصوصیات و رفتارهای یک شیء دیگر نیاز داشته باشد، از ارث‌بری استفاده می شود.
- ارث‌بری منجر به قابلیت استفاده مجدد در code می گردد و برای چندریختی زمان اجرا استفاده می شود.
- برای ارث برای در جاوا از کلمه extends استفاده می شود.

برنامه نویسی جاوا

امیر حسین کاشفی

مرور مطالب

- جلسه ششم – ارث‌بری
  - انواع ارث‌بری
  - اشاره‌گر super
  - رابطه aggregation
  - Method Overriding
  - چندریختی
  - Upcasting

ارث‌بری Inheritance

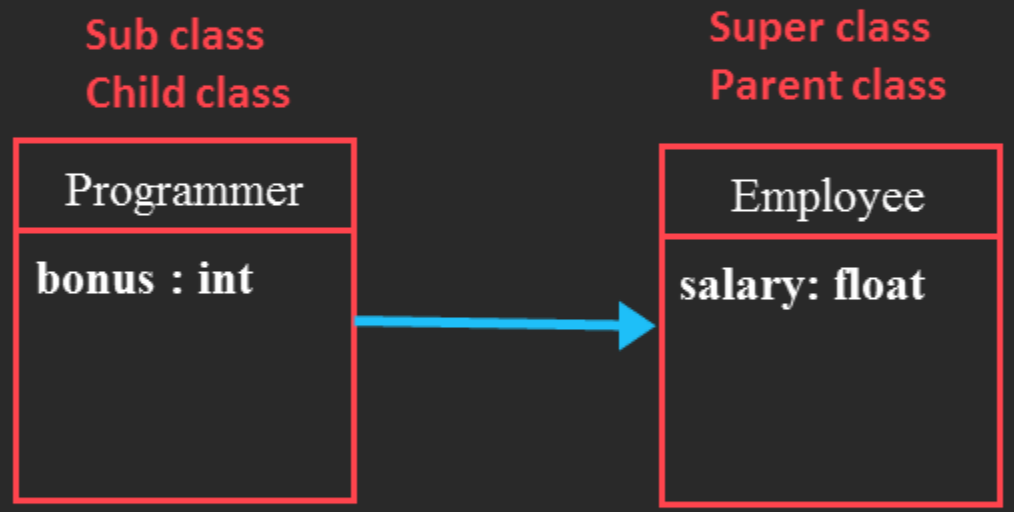
- Inheritance در جاوا مکانیزمی است که یک شیء به تمامی خواص و رفتارهای شیء والد نیاز دارد.
- با ارث بری از یک کلاس به عنوان می توان از متدها و فیلدهای کلاس والد استفاده مجدد نمود بعلاوه اینکه می توان متدها و فیلدهای جدیدی را نیز اضافه نمود.
- Inheritance یک رابطه is-A یا والد-فرزند را نشان می دهد.
- در جاوا کلاس والد که از آن ارث برده می شود را super-class و به کلاس جدیدی که ارث می برد را subclass می‌گویند.
- برای تعریف subclass از یک super-class، از کلمه extends استفاده می شود.

```
class SubClassName extends SuperClassName
{

}
```

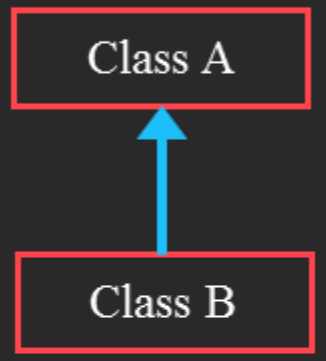
Java

Inheritance ارث‌بری

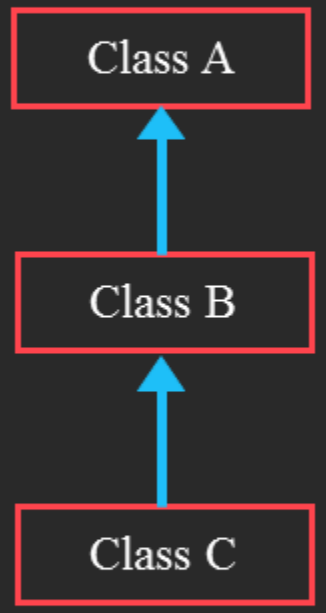
```
class Emploee {
    float salary=40000;
}

class Programmer extends Emploee {
    int bonus = 1000;
    public static void main(String args[]) {
        Programmer p = new Programmer();
        System.out.println("Programmer salary is:" + p.salary);
        System.out.println("Bonus of Programmer is:" + p.bonus);
    }
}
```



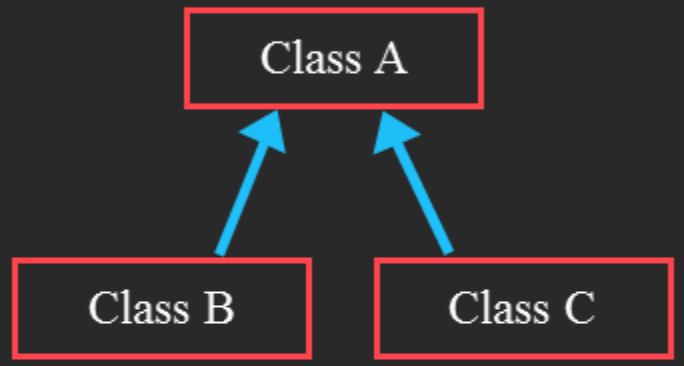
انواع ارث‌بری



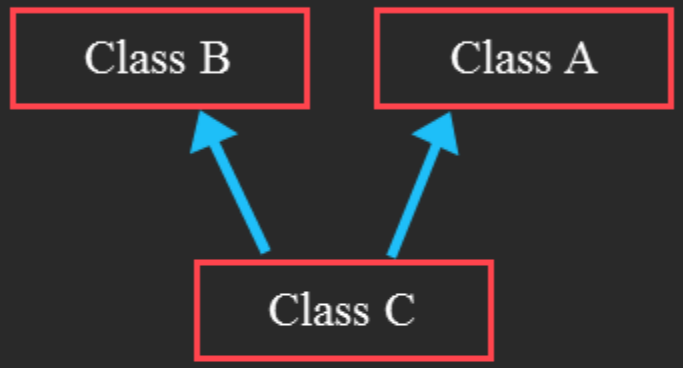
Single



Multilevel



Hierarchical



Multiple

ارث‌بری چندگانه غیر مجاز ؟؟؟

```
class A {
   int bonus = 1000;
   void msg() {System.out.println("I am A"); }
}
```

```
class B {
   int bonus = 1000;
   void msg() {System.out.println("I am B"); }
}
```

```
class C extends A,B { //Suppose if it were
   public static void main(String args[]) {
      C obj = new C();
      obj.msg(); //Now which msg() method would be invoked?
   }
}
```

super keyword in java

- super متغیری برای ارجاع به نزدیکترین super class یک subclass است که با ایجاد یک نمونه از کلاس بطور ضمنی ساخته می شود.

- کاربرد های کلمه کلیدی super عبارتست از:
  ۱. ارجاع به کلاس parent
  ۲. super() برای فراخوانی سازنده کلاس parent استفاده می شود.
  ۳. فراخوانی متدهای موجود در کلاس parent

- اگر متغیر و یا متدی را هر دوی super class و sub class داشته باشند، اولویت با اجرای متد در sub class است. مگر آنکه از کلمه کلیدی super استفاده کنیم.

ارجاع به کلاس parent

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

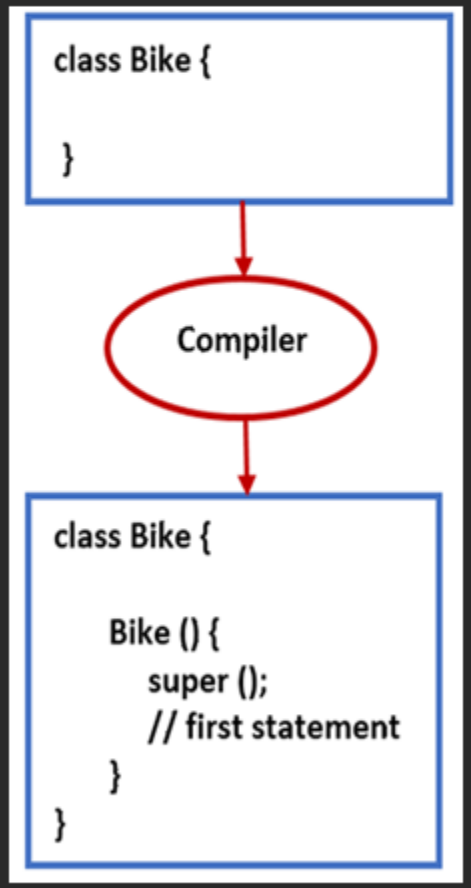
```
class Vehicle{
   int speed=50;
}
class Bike extends Vehicle{
   int speed=100;
   void display(){
      System.out.println(super.speed);
     //will print speed of Vehicle
   }
   public static void main(String args[]){
      Bike b=new Bike();
      b.display();
   }
}
```

ارجاع به متدهای کلاس parent

```
class Person{
   void message(){System.out.println("welcome");}
}
class Student extends Person{
   void message(){System.out.println("welcome to java");}

   void display(){
      message();//will invoke current class message() method
      super.message();//will invoke parent class message() method
   }
   public static void main(String args[]){
      Student s=new Student();
      s.display();
   }
}
```

فراخوانی سازنده کلاس parent

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

class Bike extends Vehicle{
  Bike(){
    super();//will invoke parent class constructor
    System.out.println("Bike is created");
  }
  public static void main(String args[]){
    Bike b=new Bike ();

  }
}
```



Bike.java

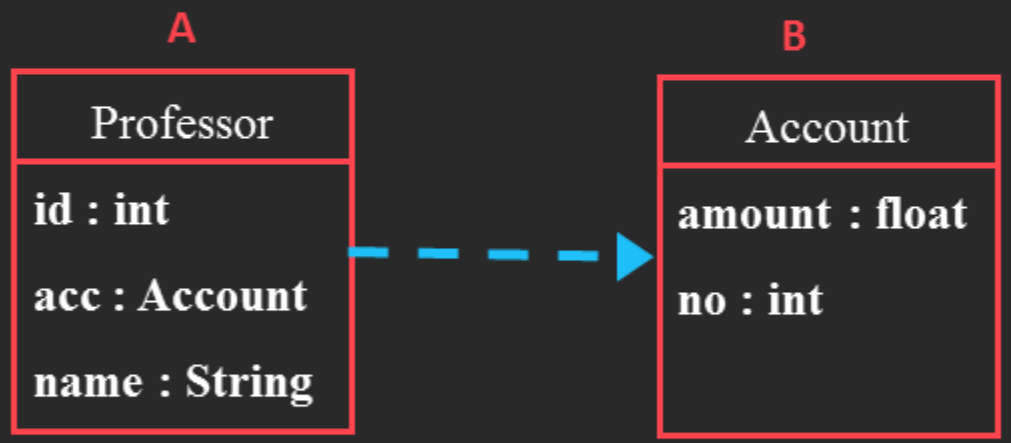
Bike.class

Aggregation in Java

- اگر کلاس A یک نمونه از کلاس B ایجاد شود، یک رابطه aggregation بین A و B برقرار خواهد شد.

```
class Account {
    float amount;
    int no; }
}
```

```
class Professor {
    int id;
    Account acc;
    String name;
}
```



- aggregation یک رابطه Has-A را نشان می دهد.

کاربرد Aggregation

- در صورت استفاده از یک کلاس دیگر و is-A نبودن آن، رابطه باید aggregation باشد.
- مهمترین کاربرد آن استفاده مجدد از کد و حذف وابستگی تا حد ممکن است.
- مثلا در با وجود کلاسی مانند **address** در هر کلاس دلخواهی می توان یک شیء از این کلاس تعریف نمود.

```
class Address {
	String city, state, country;
	Address(String city, String state , String country) {
		this.city =  city;
		this.state =  state;
		this.country =  country;
	}
}
```

Method Overriding in Java

- اگر sub class یا child class متد یکسانی در parent class داشته باشد و بخواهد متد موجود در parent class خود را به شیوه خود پیاده‌سازی کند، این امر Method Overriding یا بازنویسی متد نامیده می شود.
- Method Overriding برای runtime polymorphism استفاده می‌شود
- در حالیکه Method Overloading برای compile time polymorphism استفاده می‌شود.
- در child class یک متد برای Overriding باید:
  - رابطه is-A حتما وجود داشته باشد.
  - هم نام با متد موجود در parent class خود باشد.
  - ورودی یکسانی با متد موجود در parent class خود داشته باشد.

Method Overriding

```
class Vehicle{
   void run(){System.out.println("Vehicle is running");}
}
class Bike extends Vehicle{
   public static void main(String args[]){
     Bike obj = new Bike();
     obj.run();
   }
}
```

```
class Vehicle{
   void run(){System.out.println("Vehicle is running");}
}
class Bike extends Vehicle{
   void run(){System.out.println("Bike is running safely");}
   public static void main(String args[]){
     Bike obj = new Bike();
     obj.run();
   }
}
```

Covariant Return Type

- اگر return type اولیه نباشد، می توان با تغییر هر نوع غیر اولیه برای آن، متد را override نمود.
- مقدار return یک متد در superclass، می‌تواند در subclass متفاوت باشد که به این امر covariant return type می گویند.

```
class A{
   A get(){return this;}
}

class B1 extends A{
   B1 get(){return this;}
   void message(){System.out.println("welcome to covariant return type");}

   public static void main(String args[]){
      new B1().get().message();
   }
}
```

چند ریختی

POLYMORPHISM

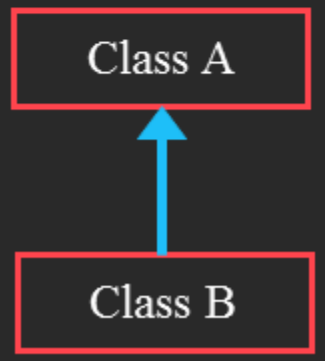
poly — many

morphos — forms

- runtime polymorphism
- compile time polymorphism

Upcasting

- فرآیندی که در آن فرخوانی به یک متد override شده در زمان اجرا مشخص می شود را Runtime polymorphism گویند.
- زمانیکه متغیر ارجاعی کلاس parent با نوع کلاس فرزند ایجاد می‌شود، Upcasting نامیده می شود:




```
class A{}
class B extends A{}

A a=new B();     //upcasting
```

Runtime polymorphism

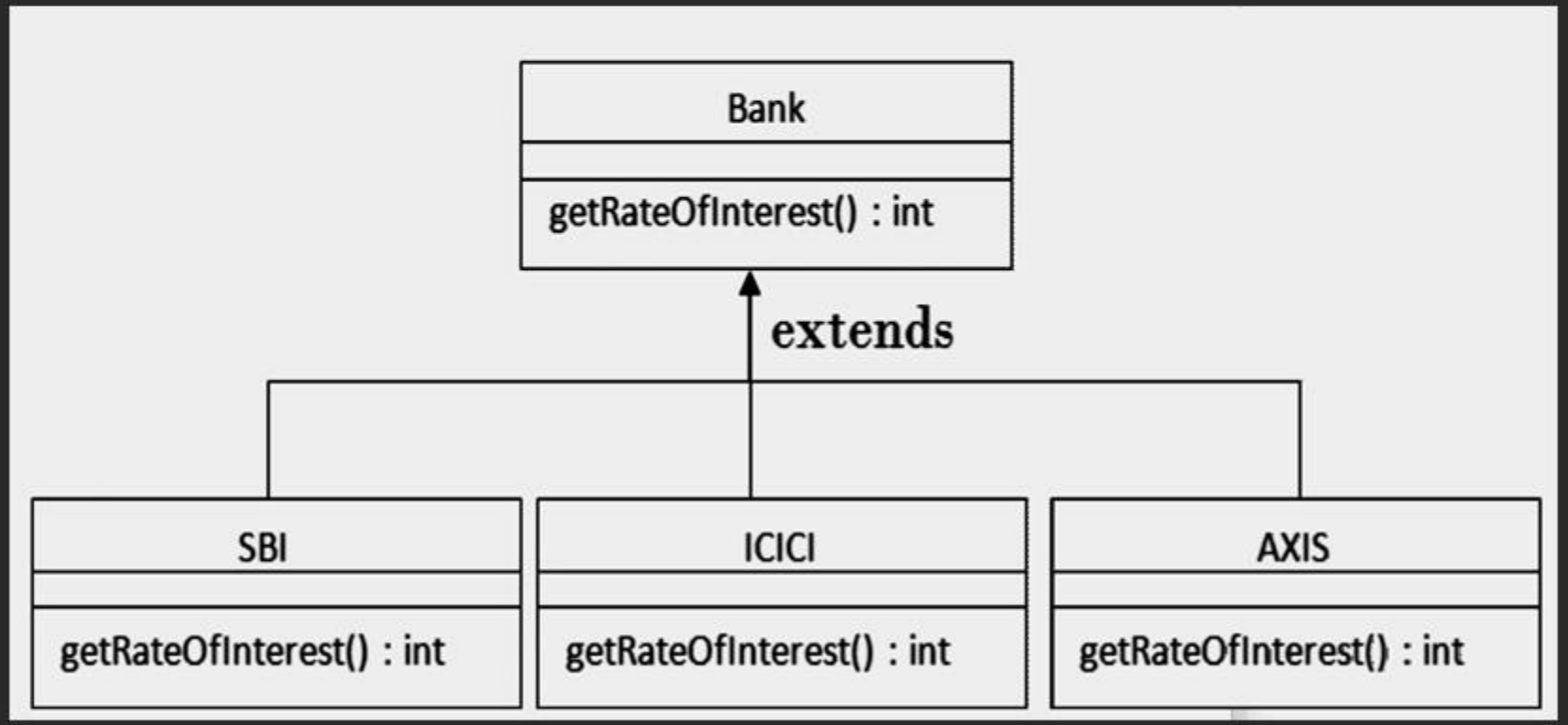
- پلی‌مورفیسم در زمان اجرا تنها در مورد متدها اجرایی می شود نه متغیرها.

```
class Bike{
  void run(){System.out.println(”30KM");}
}
class Splender extends Bike{
    void run(){
       System.out.println("60KM");
    }
    public static void main(String args[]){
        Bike b = new Splender();//upcasting
        b.run();
    }
}
```

```
class Bike{
   int speed=90;
}
class Honda extends Bike{
    int speed=150;

    public static void main(){
        Bike obj=new Honda ();
        System.out.println(obj.speed);
}
```

Runtime polymorphism

- فرآیندی که در آن فرخوانی به یک متد override شده در زمان اجرا مشخص می شود.



Runtime polymorphism

```
class Bank{
   int getRate (){return 0;}
}
class SBI extends Bank{
   int getRate (){return 8;}
}
class ICICI extends Bank{
   int getRate (){return 7;}
}
class AXIS extends Bank{
   int getRate (){return 9;}
}
```

```
class Test{
   public static void main(String args[]){
   Bank b1=new SBI();
   Bank b2=new ICICI();
   Bank b3=new AXIS();
   System.out.println("SBI  Rate : "+b1.getRate ());
   System.out.println("ICICI  Rate : "+b2.getRate ());
   System.out.println("AXIS  Rate : "+b3.getRate ());
   }
}
```

multilevel polymorphism

```
class Animal{
    void eat(){
        System.out.println("eating");}
}

class Dog extends Animal{
    void eat(){

System.out.println("eating  fruits");}
}
```

```
class BabyDog extends Dog{
    void eat(){
        System.out.println("drinking  milk");}

    public static void main(String  args[]){
        Animal  a1,a2,a3;
        a1=new Animal();
        a2=new Dog();
        a3=new BabyDog();
        a1.eat();
        a2.eat();
        a3.eat();
}
```

multilevel polymorphism

```
class Animal{
     void eat(){System.out.println("animal is eating...");}
}

class Dog extends Animal{
   void eat(){System.out.println("dog is eating...");}
}

class BabyDog1 extends Dog{
   public static void main(String args[]){
      Animal a=new BabyDog1();
      a.eat();
   }
}
```

برنامه نویسی جاوا

امیر حسین کاشفی

مرور مطالب

- جلسه هفتم – تجرید
  - انقیاد و انواع آن
  - کلاس مجرد
  - واسط
  - Method Overriding
  - Downcasting

Binding

- اتصال یک فراخوانی متد به بدنه متد را انقیاد یا binding می گویند.
- دو نوع binding وجود دارد:

- static binding (early binding)
- dynamic binding (late binding)

Binding

متغیرها دارای یک نوع هستند:

```
int data=30;
```

اشیاء نیز دارای یک نوع هستند:

```
class Animal{}
class Dog extends Animal{

public static void main(String args[]){
        Dog d1=new Dog();
    }
}
```

ارجاعات دارای یک نوع هستند:

```
class Dog{
   public static void main(String args[]){
   Dog d1;//Here d1 is a type of Dog
   }
}
```

static binding

- زمانیکه نوع یک‌شیء توسط کامپایلر در compile-time مشخص شود، به آن static binding گویند.
- متدهای private، final و static در یک کلاس بطور استاتیک bind می شوند.

```
class Dog{
    private void eat(){
        System.out.println("dog is eating...");}

    public static void main(String args[]){
        Dog d1=new Dog();
        d1.eat();
    }
}
```

dynamic binding

- زمانیکه نوع یک شیء توسط JVM در run-time مشخص شود، به آن dynamic binding گویند.

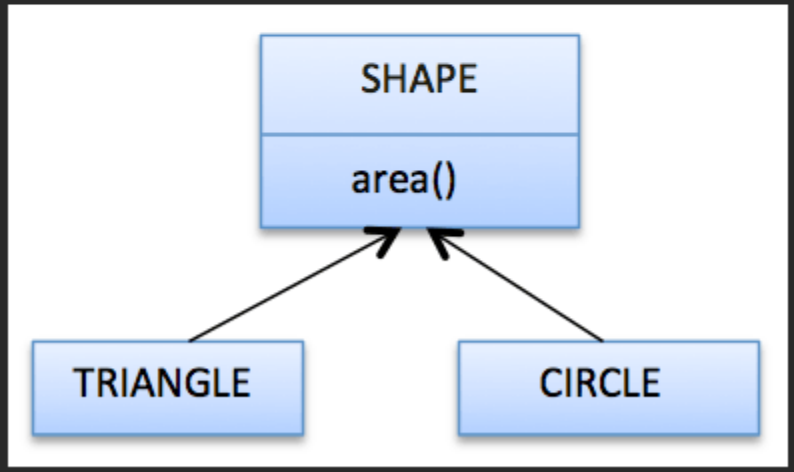
```
class Animal{
    void eat(){ System.out.println("animal is eating...");}
}

class Dog extends Animal{
    void eat(){ System.out.println("dog is eating...");
    }
    public static void main(String args[]){
        Animal a=new Dog();
        a.eat();
    }
}
```

Compiler نمی تواند تشخیص دهد که کدام متد را اجرا کند زیرا a هم نمونه ای از کلاس Dog است و هم Animal پس این binding در زمان اجرا انجام می شود.

تجرید یا Abstraction

- تجرید فرآیند پنهان‌سازی جزئیات پیاده سازی و ارائه قابلیت‌ها به کاربر.
- Abstraction روی آنچه object انجام می‌دهد تمرکز دارد نه چگونگی انجام آن.
- تجرید به دو صورت فراهم می شود:

1. **Abstract class (0 to 100%)**
2. **Interface (100%)**



Abstract class

- Abstraction class با کلمه کلیدی abstract قبل از نام کلاس تعریف می شود و می تواند شامل متدهای abstract و non-abstract و هر عضو دیگر کلاس باشد.

```
abstract class Class-name
{
  // abstract or non- abstract methods
}
```

- Abstraction method با کلمه کلیدی abstract قبل از نام متد تعریف می شود و هیچ پیاده‌سازی در آن وجود ندارد.

```
abstract void printStatus();//no body and abstract
```

- اگر کلاسی Abstraction method داشته باشد حتما باید آن کلاس abstract باشد.

Abstract class

```
abstract class Bike {
        abstract void run();
}
```

```
class Honda extends Bike{
    void run()  {
          System.out.println("running safely..");
    }
    public static void main(String args[]) {
        Bike obj = new Honda();   obj.run();
    }
}
```

مثالی دیگر از Abstract class

```
abstract class Shape{
      abstract void draw();
}
```

```
//In real scenario, implementation is provided by others i.e. unknown by end use
r
class Rectangle extends Shape{
     void draw(){ System.out.println("drawing rectangle"); }
}
class Circle extends Shape{
    void draw(){ System.out.println("drawing circle"); }
}
```

```
//In real scenario, method is called by programmer or user
class TestAbstraction {
     public static void main(String args[]){
            Shape s=new Circle();
            s.draw();  }
}
```

مثالی دیگر از Abstract class

```
abstract class Bank{
    abstract int getRateOfInterest();
}
```

```
class SBI extends Bank{
    int getRateOfInterest() { return 7; }
}
```

```
class PNB extends Bank{
    int getRateOfInterest() { return 8; }
}
```

```
class TestBank{
   public static void main(String args[]){
       Bank b=new SBI();
   //if object is PNB, method of PNB will be invoked
  int interest=b.getRateOfInterest();
  System.out.println("Rate of Interest is: "+interest+" %");    }}
```

واسط یا interface

- Interface یک blueprint یک کلاس است و با کلمه کلیدی interface تعریف می شود و می تواند تنهاشامل متدهای abstract و ثوابت static باشد.

```
interface interface-name
{
  // abstract methods  or static constants
}
```

- در جاوا interface می تواند یک رابطه is-a و ارث‌بری چندگانه را پیاده سازی کند.

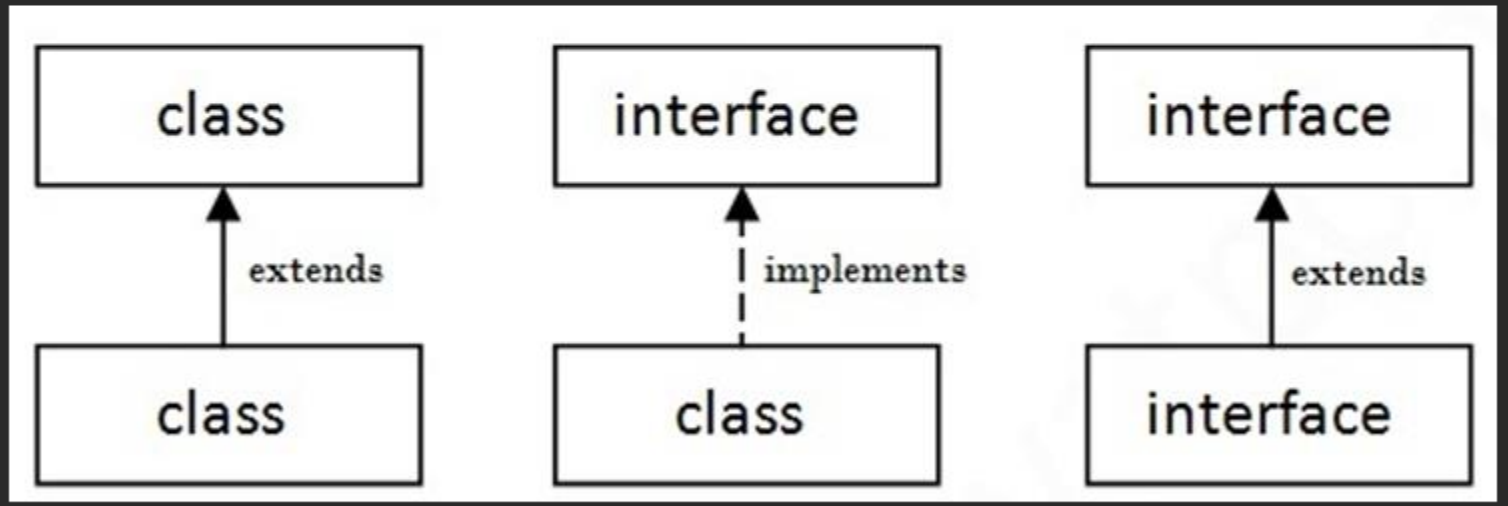
چرا interface ؟

- پیاده‌سازی تجرید کامل
- پشتیبانی از کاربرد ارث‌بری چندگانه
- دستیابی به خاصیت loosly coupled

واسط یا interface

- در Interface بطور پیش‌فرض کلیه متغیرها public static final و کلیه متدها public abstract تعریف می‌شوند.

رابطه بین کلاس و اینترفیس:



مثالی از interface

```
interface printable{
    void print();
}
```

```
class IClass implements printable{
     public void print() { System.out.println("Hello"); }
     public static void main(String args[]){
            IClass obj = new  IClass();
            obj.print();
     }
}
```

Multiple interface

```
interface printable{
    void print();
    void show();
}
```

```
interface Showable{
    void show();
}
```

```
class IClass implements  Printable, Showable{
     public void print() { System.out.println("Hello"); }
     public void show() { System.out.println("Welcome"); }
     public static void main(String args[]){
            IClass obj = new  IClass();
            obj.print();
            obj.show();
     }
}
```

مثالی از ارث‌بری interface

```
interface printable{
    void print();
}
```

```
interface Showable extends Printable{
    void show();
}
```

```
class IClass implements  Showable{
     public void print() { System.out.println("Hello"); }
     public void show() { System.out.println("Welcome"); }
     public static void main(String args[]){
            IClass obj = new  IClass();
            obj.print();
            obj.show();
     }
}
```

interface vs abstract class

Abstract class	interface
شامل متدهای abstract و غیره	تنها شامل متدهای abstract
عدم پشتیبانی از ارث‌بری چندگانه	پشتیبانی از ارث‌بری چندگانه
کلیه متغیرها	تنها متغیرهای static و final
شامل متد main، سازنده و متد static	عدم وجود متد main، سازنده و متد static

Java instanceof

- عملگری است که مشخص می کند آیا یک شیء نمونه ای از یک کلاس، زیرکلاس یا اینترفیس هست یا خیر.
- instanceof در جاوا یک عملگر مقایسه‌گر نوع است زیرا یک instance را با یک type مقایسه می کند.

```
class Simple1{
    public static void main(String args[]){
        Simple1 s=new Simple1();
        System.out.println(s instanceof Simple1); //true
    }
}
```

- یک instance از یک کلاس نوعی از آن کلاس و کلیه super class های وابسته به آن کلاس می باشد.

Downcasting

- زمانیکه متغیر ارجاعی کلاس child با نوع کلاس والد ایجاد می‌شود، Downcasting نامیده می شود:
- در صورت انجام این امر به error خواهیم خورد:

```
Dog d=new Animal();//Compilation error
```

- با typecasting به error در زمان اجرا خواهیم خورد:

```
Dog d=(Dog)new Animal();
//Compiles successfully but ClassCastException is thrown at runtime
```

- با عملگر instanceof این امر ممکن می شود.

Downcasting

```
class Animal { }
class Dog extends Animal {
      static void method(Animal a) {
            if(a instanceof Dog){
            Dog d=(Dog)a;//downcasting
            System.out.println("ok downcasting performed");
       }
 }
      public static void main (String [] args) {
           Animal a=new Dog();
           Dog.method(a);
     }
}
```

برنامه نویسی جاوا

امیر حسین کاشفی

مرور مطالب

- جلسه هشتم – کلاس‌های Generic و تودرتو
  - Java Generic
  - لیست آرایه‌ای
  - کلاس تودرتو و انواع آن

Java Generic

- اگر منطق پیاده سازی یک کلاس یا متد برای انواع مختلف یکسان باشد، لازم است آنها را generic یا عام تعریف نمود.
- استفاده از مفهوم generic منجر به کاهش کدنویسی خواهد شد.
- برای تعریف یک کلاس و یا متد Generic لازم است < T > را در کلاس یا متد استفاده کرد که T یک متغیر دلخواه است.
- زمان استفاده از کلاس یا متد، بجای T از Wrapper class ها نام یک کلاس که بیانگر نوع دلخواهی هستند، قرار داده می شود.
- علاوه بر متد و کلاس، یک abstract class و interface نیز می توانند generic باشند.

Generic Class

```
public class Gen<Y> {
	Y var;
	public void print(Y[] input) {
		for (Y o : input)
			System.out.print(o);
	}
	public Y first(Y[] input) {
		return input[0];
	}
}
```

- ساختار تعریف یک کلاس Generic و نحوه فراخوانی آن به صورت زیر است:

```
Gen<String> gs = new Gen<String>();
Gen<Integer> gi = new Gen<Integer>();
```

- کلاس ذکر شده در نمونه سازی می تواند ذکر نشود و بصورت پیش فرض استنتاج شود (عمل دیاموند <>)

- یک کلاس Generic می تواند چندتایی هم باشد:

```
public class Gen<A,B,C> {

}
```

```
Gen<String, Integer, Double> gs;
gs = new Gen<String, Integer, Double> ();
```

Generic Class

- یک کلاس Generic را می توان محدود نمود برای این منظور می توان از یکی از کلاس‌های زیرمجموعه object ارث بری نمود.

```
public class Gen<Y extends Number> {

}
```

- در مثال روبرو نوع های تعریف شده برای جایگزینی با Y باید عددی باشند.

- نمونه سازی، تعریف آرایه، تعریف متغیر استاتیک از نوع پارامتر Generic منجر به خطای کامپایل خواهد شد.

```
Y a = new Y();
```

```
public static Y a;
```

```
Y[] a=new Y[10];
```

Generic Method

- یک متد Generic را می توان بصورت زیر در یک کلاس تعریف نمود:

```
public class Gen {
   int var;
   public <Y> void print(Y[] input) {
      for (Y o : input)
        System.out.print(o);
   }
   public static void main(String[] args) {
      String[] s = { "Amir", "Hosein", "Kashefi" };
      Integer[] i = { 1, 2, 3 };
      Gen gs = new Gen();
      gs.print(s);
      gs.print(i);
   }
}
```

لیست آرایه‌ای

- آرایه دارای ساختار ایستاست، زیرا اندازه‌ی اولیه آن مشخص است.
- لیست آرایه‌ای یا ArrayList یک آرایه پویاست که طول آن در متغیر بوده و آیتم ها قابل حذف و اضافه شدن از لیست می‌باشند.
- در مرحله اول باید کلاس ArrayList را در برنامه **import** نمود:

```
import java.util.ArrayList;
```

- عناصر ArrayList می‌توانند از نوع خاصی باشند و برای تعریف آن داریم:

```
ArrayList<Type> lstTest = new ArrayList<Type>();
```

- عناصر ArrayList می‌توانند Generic باشند و اعضا از انواع مختلفی باشند:

```
ArrayList lstTest = new ArrayList();
```

متدهای مهم لیست آرایه‌ای

- بعد از ایجاد یک ArrayList می توان با متد add()، عناصری به آن اضافه کرد:

```
ArrayList lstTest = new ArrayList();

lstTest.add( "item0" );
lstTest.add( 1001 );
lstTest.add( true );
```

lstTest

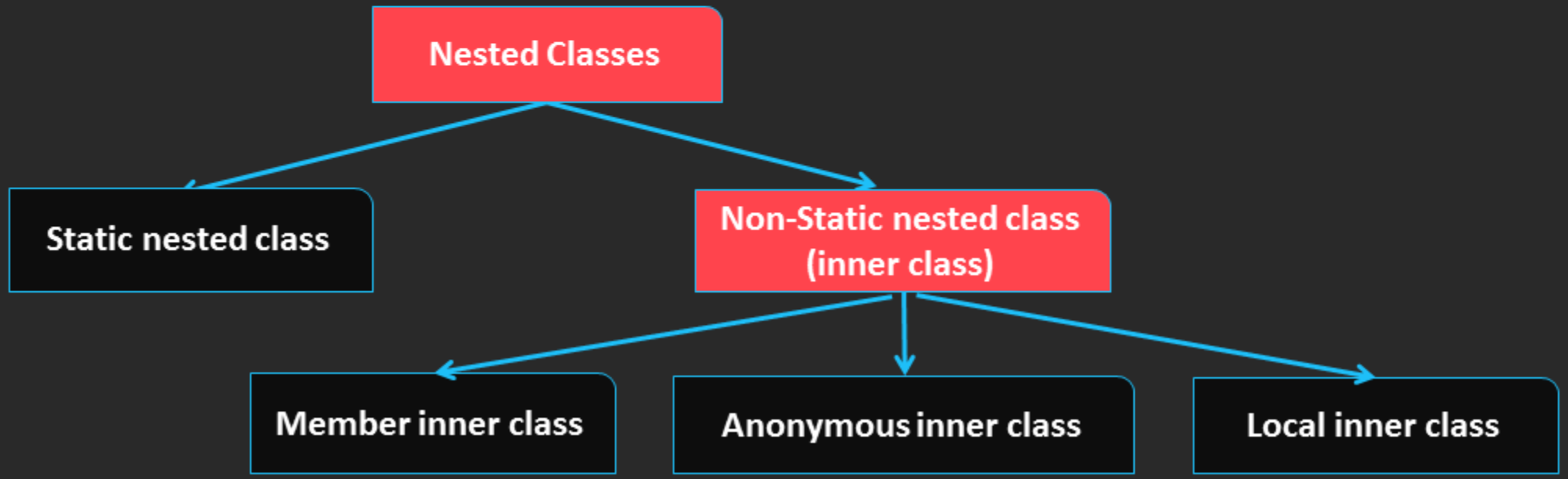
0	1	2
item0	1001	true

- دیگر متدهای مهم در لیست‌های آرایه ای عبارتند از:

متد	مفهوم
lstTest.size()	تعداد عناصر لیست
lstTest.add(index, item)	افزودن عنصری با مقدار item در جایگاه index
lstTest.remove(index)	حذف عنصر جایگاه index
lstTest.remove(item)	حذف عنصری با مقدار item
lstTest.contain(item)	یافتن اینکه عنصر item در لیست هست یا خیر
lstTest.get(index)	بازیابی عنصر جایگاه index

Nested classes

- کلاس‌ها می توانند به صورت تودرتو درون یکدیگر تعریف شوند.
- نوشتن کلاس‌ها بصورت تودرتو منجر به نگهداری بهتر و کاهش کدنویسی می گردد.
- در جاوا انواع کلاس های تودرتو را می توان به‌صورت زیر طبقه بندی نمود:



Member inner class

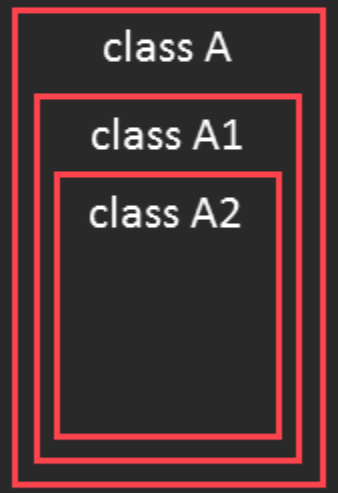
- کلاسی که درون یک class یا interface تعریف شود را inner class می گویند و کلاس بیرونی را outer class گویند.
- از آنجا که inner class یک ارجاع به outer class است، به کلیه اعضای outer class حتی موارد private دسترسی دارد.
- کلاسی که درون یک class یا interface به عنوان یک عضو تعریف شود را member inner class می گویند.

```
class JavaOuterClass {
    //code

    class JavaInnerClass {
        //code
    }
}
```

Member inner Classes

- یک interface نیز می تواند در یک کلاس top-level، یک interface دیگر یا یک حوزه static تعریف شود.

نمونه سازی از یک member inner class




```java
public class A {
  int a;

  class A1 {
    int a1;

    class A2 {
       int a2;
    }
  }
}
```

```java
class Programmer {

   public static void main(String args[]) {
      A x = new A();
      A.A1 x1 = x.new A1();
      A.A1.A2 x2 = x1.new A2();
      x.a = 0;
      x1.a1 = 1;
      x2.a2 = 2;
   }
}
```

Local inner class

```
public class localInner1 {
    private int data = 30;// instance variable

    void display() {
        class Local {
            void msg() {
                System.out.println(data);
            }
        }
        Local l = new Local();
        l.msg();
    }

    public static void main(String args[]) {
        localInner1 obj = new localInner1();
        obj.display();
    }
}
```

- کلاسی که درون یک method کلاس تعریف شود local inner class را می گویند.
- local inner class ها نمی توانند بیرون از متد فراخوانی شوند.

Anonymous inner class

- یک کلاس بدون نام است که جهت بازنویسی متد(های) یک abstract class یا interface، new شده و درون یک class یا interface تعریف شود را Anonymous inner class می گویند.
- نام این کلاس‌های بی نام، توسط کامپایلر تعیین می شود.

کامپایلر

```
static class TestAnonymousInner$1 extends Person
```

```
abstract class Person {
    abstract void eat();
}
```

```
class TestAnonymousInner {
    public static void main(String args[]) {
        Person p = new Person() {
            void eat() {
                System.out.println("nice fruits");
            }
        };
        p.eat();
    }
}
```

Static nested class

```
class A {
  static int data = 30;

  static class Inner {
    static void msg() {
      System.out.println("data is " + data++);
    }
  }

  public static void main(String args[]) {
     A obj = new A.Inner();
     obj.msg();
     A.Inner.msg();
  }
}
```

- یک static class که درون یک class تعریف شود را static nested class می گویند.
- یک static nested class تنها می تواند به متدها و اعضای static کلاس outer دستیابی داشته باشد و معمولا کلاس outer از آن نمونه سازی می کند.