بسم الله الرحمن الرحیم

مقدمه ای بر زبان برنامه نویسی جاوا
به همراه مثال های کاربردی

مجتمع فنی تهران

گردآورنده : محمد صادق صابری

sadeghsabery@gmail.com

بهار 1396

فهرست

مقدمه :

جاوا یک زبان برنامه نویسی قدرتمند است، و در سال 1995، توسط جیمز گاسلینگ به طور رسمی به برنامه نویسان و گسترش دهندگان برنامه های کاربردی معرفی گردید. ابتدا نام این زبان، اوآک (OAK) بوده، که به دلیل مشکلات ثبتی به جاوا، تغییر نام داده است. اوآک به معنای درخت بلوط، و جاوا یک اسم محلی برای قهوه می باشد.

در ابتدا پروژه ای با نام گرین (Green) در شرکت سان تعریف شد. مقصود از این پروژه، سکویی برای پیاده سازی برنامه های کاربردی بر روی تمامی ابزارهای الکترونیکی همانند: مایکرویو، تلویزیون و ... بود. اما در آن زمان، این پروژه با شکست روبرو شد. این اتفاق باعث شد، تا جیمز (مدیر پروژه شکست خورده Green) به کمک تیم خود یک زبان برنامه نویسی جدید تولید نماید. یکی از دلایلی که بعضی از مواقع جاوا را به عنوان یک سکوی برنامه نویسی یاد می کنند، همین موضوع است.

برتری های زبان برنامه نویسی جاوا نسبت به سایر زبان های برنامه نویسی

- مستقل از سکوی اجرا (Free Platform): این خاصیت به برنامه نویسان و گسترش دهندگان این امکان را می دهد، تا بدون وابستگی به هیچ سیستم عاملی، برنامه های کاربردی خود را تولید و اجرا نمایند.
- پشتیبانی از برنامه نویسی شی گرا (Object Oriented Programming): این خاصیت به برنامه نویسان و گسترش دهندگان این امکان را می دهد، تا بتوانند برنامه های کاربردی خود را پیاده، و آن را به سهولت گسترش دهند.
- دارای تکنولوژی های متعدد کد باز و غیر کد باز: همان گونه که در پیشتر یادآور شدیم، جاوا متعلق به شرکت سان می باشد، اما شرکت های متعددی برای این زبان، تکنولوژی ها و فریم ورک های مختلفی را ارایه کرده اند.

سطوح دسترسی به کلاس :

در زبان جاوا ، سطوح دسترسی برای یک شیء وجود دارد که با توجه به آن می توان از آن شیء و یا ویژگی و رفتار آن استفاده کرد . سطوح دسترسی سه دسته می باشند :

Public : بیشترین سطح دسترسی با حداقل محدودیت می باشد .

Protected : خارج از پکیج تعریف شده دسترسی مقدور نمی باشد .

Private : محدود ترین دسترسی می باشد و تنها در کلاسی که در آن تعریف شده است به آن دسترسی داریم .

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	X
private	Y	X	X	X

نوع داده اولیه :

زبان جاوا با توجه به نوع داده ای که برای متغیر ها استفاده می کنیم ، انواع داده تعریف شده ای را ارائه می دهد که در جدول زیر آمده است :

Data Type	Default Value (for fields)	Range
byte	0	-127 to +128
short	0	-32768 to +32767
int	0	-2147483648 to 2147483647
long	0L	0 to 2^{64}-1

float	0.0f	0 to 65535
double	0.0d	
char	‘\u0000′	
String (object)	null	
boolean	false	

کلاس (Class) :

به یک موجودیت گفته می شود که می تواند دارای ویژگی ها و رفتار باشد . هر کلاس با فرمت زیر ایجاد می شود :

[سطح دسترسی خارج از کلاس] class [نام کلاس]

{ ... }

به ویژگی یک کلاس Field و به رفتار کلاس method گفته می شود . فیلد در کلاس بیانگر ویژگی ذاتی آن کلاس می باشد و متد بیانگر اعمالی است که آن موجودیت قادر به انجام آنها می باشد .

مثلا کلاس انسان : دارای ویژگی نام ، نام خانوادگی ، سن و ... می باشد و از جمله رفتار های آن می توان به صحبت

کردن ، راه رفتن ، خوردن و ... اشاره کرد .

در مثال فوق کلاس انسان را اینگونه می نویسیم :

```
public class Human{
private String firstName;
private String lastName;
        public String getFirstName()
        {
           Return firstName;
        }
}
```

در مثال فوق کلاس Human ایجاد شده است .

شیء (Object) :

به نمونه ای از کلاس یک شیء از آن کلاس گفته می شود . یک شیء از یک کلاس دارای ویژگی ها و رفتار های آن کلاس می باشد و نکته مهم ای است که در جزئیات این ویژگی ها و رفتار ، اشیاأ ساخته شده از یک کلاس می توانند با یکدیگر متفاوت باشند .

مثلا از کلاس انسان یک نمونه ایجاد می کنیم : نام آنرا "ali" و نام خانوادگی را "jafari" می گذاریم . یکی شی دیگری از کلاس انسان می سازیم . اینبار نام آنرا "john" و نام خانوادگی آنرا "smith" می گذاریم . رفتار های این دو شیء نیز می توانند با هم متفاوت باشند ، مثلا در متد صحبت کردن john عبارت "Hello" چاپ می شود و در متد مربوط به ali عبارت "سلام" چاپ می شود .

JDK چیست :

Java development kit (JDK) شامل JVM و منابعی دیگر برای کمک به توسعه دهندگان برنامه به زبان جاوا در سه پلتفرم jse(java standard edition) ، jee(java enterprise edition) و jme(java micro edition) می باشد .

JRE چیست ؟

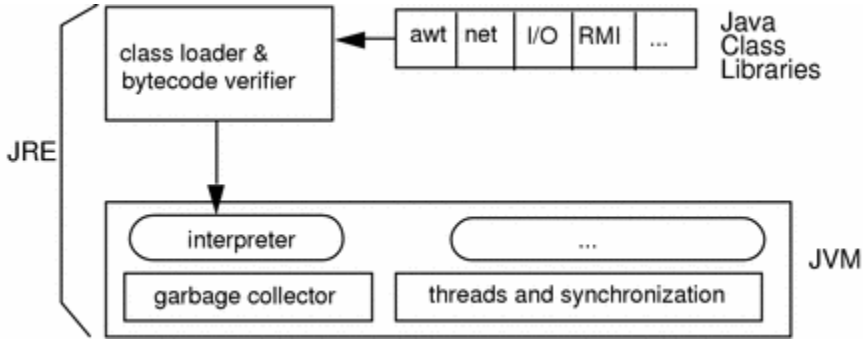
Java runtime environment (jre) ، محیطی است که برنامه های کامپایل شده را آماده اجرا در jvm می سازد .

شامل موارد زیر می باشد :

کد های ضروری برای اجرای برنامه های جاوا ، مدیریت حافظه و هندل کردن exception .

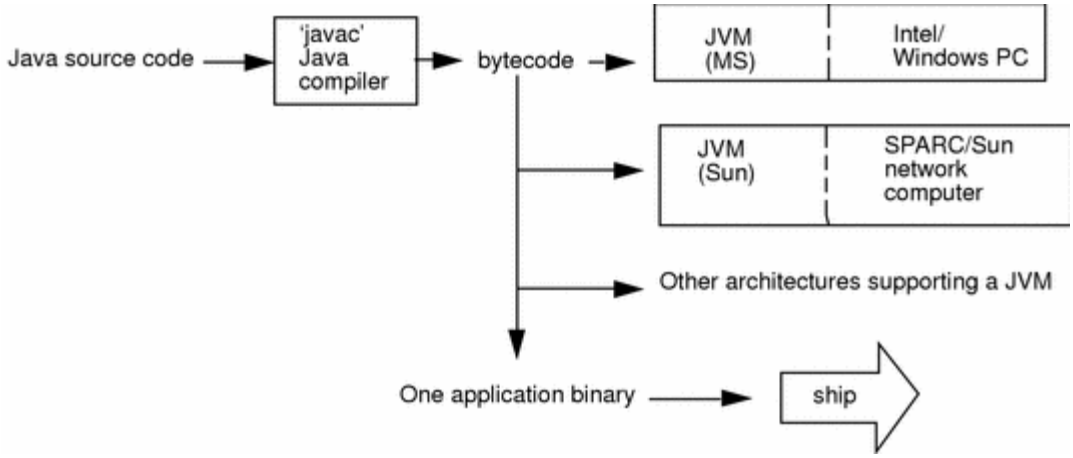
پیاده سازی JVM .

نمودار زیر بیانگر عملکرد jre می باشد .



Jvm عبارتست از یک ماشین مجازی برای اجرای برنامه های نوشته شده به زبان جاوا و مستقل از پلتفرم عمل می کند .

شکل زیر بیانگر عملکرد jvm م باشد :



Lexical Token :

به اجزاء هر برنامه جاوا lexical token و یا token گفته می شود . ازجمله token ها می توان به identifier ، keyword ، literals ، comments ، datatypes و operators اشاره کرد .

Identifier : همه نامگذاری هایی که توسط برنامه نویس انجام می گیرد مانند نام متغیر ، نام متد و نام کلاس .

Keyword : همه نامگذاری های از پیش تعریف شده توسط جاوا را گویند . لیست زیر شامل keyword ها می باشد :

Keywords in Java				
abstract	default	if	private	this
assert	do	implements	protected	throw
boolean	double	import	public	throws
break	else	instanceof	return	transient
byte	enum	int	short	try
case	extends	interface	static	void
catch	final	long	strictfp	volatile
char	finally	native	super	while
class	float	new	switch	
continue	for	package	synchronized	

Literal : مقداری که در یک متغیر ذخیره می شود و شامل اعداد ، حروف و علائم می باشد .

استاندارد javabeans : کلاسی که دارای ویژگی های زیر باشد از این استاندارد تبعیت می کند :

همه ویژگی های آن private باشند . constructor کلاس هیچ آرگومان ورودی نداشته باشد ، اینترفیس serializable را پیاده سازی کند و دسترسی به ویژگی های کلاس تنها با استفاده از متدهای setter و getter آن ویژگی مقدور می باشد .

: Constructor

کلاسی که دارای constructor می باشد اجازه می دهد که از خودش در خارج از کلاس آبجکت ساخته شود . تعریف کانستراکتور مشابه تعریف متد می باشد با دو تفاوت : 1- هم نام کلاس می باشد . 2- نوع داده ای خروجی ندارد .

یک کلاس ساده در جاوا به صورت پیش فرض دارای کانستراکتور بدون آرگومان ورودی می باشد . یک کلاس می تواند دارای چندین کانستراکتور باشد و تفاوت آنها در تعداد آرگومان های ورودی است .

: Enum

از نوع enum زمانی استفاده می شود که به مجموعه ای ثابت از مقادیر احتیاج داشته باشیم مانند فصل های یک سال و چهار عمل اصلی ریاضی .

به عنوان مثال کلاس چهار عمل اصلی ریاضی در زیر آورده شده است :

```
public enum Operation {
    PLUS,
    MINUS,
    TIMES,
    DIVIDE;
    double calculate(double x, double y) {
        switch (this) {
            case PLUS:
                return x + y;
            case MINUS:
                return x - y;
            case TIMES:
                return x * y;
            case DIVIDE:
                return x / y;
            default:
                throw new AssertionError("Unknown operations " + this);
        }
    }
}
```

جهت اجرا در کلاس main به صورت زیر دو عدد را باهم جمع می کنیم :

```
public class Main {
```

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

        double result = Operation.PLUS.calculate(1, 2);

        System.out.println(result); //3.0

    }

}
```

آرایه :

مجموعه ای از تعداد مقادیر ثابت از نوع یکسان و مشخص می باشد . مقدار دهی یک آرایه به دو صورت امکانپذیر می باشد :

روش اول : نام آرایه به همراه ایندکس آن مساوی با مقدار جدید.

به عنوان مثال :

```
int[] anArray = new int[10];

 anArray[0] = 100;
 anArray[1] = 200;
 anArray[2] = 300;
 anArray[3] = 400;
 anArray[4] = 500;
 anArray[5] = 600;
 anArray[6] = 700;
 anArray[7] = 800;
 anArray[8] = 900;
 anArray[9] = 1000;
```

روش دوم : در هنگام تعریف آرایه به جای درج تعداد خانه های آرایه ، المانها آرایه در آکولاد تعریف می گردند .

```
char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',
                    'i', 'n', 'a', 't', 'e', 'd' };
```

ساختار فایل سورس در برنامه جاوا در شکل زیر آمده است :

```
// Filename: NewApp.java

// PART 1: (OPTIONAL) package declaration
package com.company.project.fragilePackage;

// PART 2: (ZERO OR MORE) import declarations
import java.io.*;
import java.util.*;

// PART 3: (ZERO OR MORE) top-level class and interface declarations
public class NewApp { }

class AClass { }

interface IOne { }

class BClass { }

interface ITwo { }
// ...
// end of file
```

با استفاده از دستور javac می توان از commandline کلاس جاوا را کامپایل نمود .پسوند هر کلاس پس از کامپایل به class. تغییر می یابد .

کلاس و متد از نوع abstract :

یک کلاس abstract با عنوان abstract قبل از class تعریف می گردد . از این کلاس نمی توان نمونه ای ایجاد کرد و برای دسترسی به ویژگی های آن تنها باید از آن ارث بری نمود .

یک متد abstract متدی است که هیچ پیکره و بدنه ای ندارد . اگر کلاسی دارای متد abstract باشد ، باید آن کلاس از نوع abstract تعریف گردد .

یک subclass از یک کلاس abstract باید تمامی متد های abstract آنرا پیاده سازی نماید در غیر اینصورت باید خود آن subclass نیز از نوع abstract تعریف گردد .

کلاس و متد از نوع final :

زمانیکه یک متد از نوع final تعریف می شود به این معنی است که هیچ subclass نمی تواند آن متد را بازنویسی(override) کند . و نیز کلاسی که از نوع final می باشد نمی تواند به عنوان subclass نقش آفرینی کند .

نوع static :

کلاس از نوع static کلاسی است که می توان از خارج از کلاس بدون ساختن شیء از آن کلاس ، به آن دسترسی داشت .

همچنین اگر در کلاسی ، property از نوع static تعریف گردد ، مستقیما می توان در خارج از کلاس با استفاده از شیء از کلاس مربوطه به آن ویژگی دست یافت . این قاعده راجع به متد های از نوع static نیز صدق می کند .

متد synchrinized :

به مثال زیر توجه کنید :

```
public class SynchronizedCounter {
    private int c = 0;

    public synchronized void increment() {
        c++;
    }

    public synchronized void decrement() {
        c--;
    }

    public synchronized int value() {
        return c;
    }
}
```

در کلاس فوق دو متد از نوع synchronized تعریف شده است که یکی مقدار counter را به علاوه 1 و دیگری منهای 1 می کند . در صورتی که چند نمونه از کلاس SynchronizedCounter ایجاد گردد ، دسترسی همزمانی به مقدار c جهت تغییر آن وجود نخواهد داشت به این معنی که هر thread یی که زودتر فعال باشد متغیر را block می کند و پس از تغییر آن و ثبت مقدار جدید ، سایر thread ها حق دسترسی به آنرا دارند .

Conversion :

هر شیء در جاوا دارای نوع خاصی می باشد . در مواردی ، نوع داده باید به نوع داده دیگری تبدیل شود . مثلا عدد 5 در یک رشته از نوع string ذخیره سازی گردد . در اینجا باید عدد 5 به نوع رشته ای تبدیل شود .در برخی موارد این تبدیل امکان پذیر نمی باشد و خطای زمان کامپایل اعلام می گردد .

عملگر ها :

در زبان جاوا ، عملگرها به دسته های عملگر های ریاضی که عملیات ریاضی را انجام می دهند و عملگر های مقایسه ای که ویژگی مقایسه ای دارند ، عملگر های منطقی و عملگر های شرطی تقسیم می شوند .

راجع به عملگر های ریاضی به چند مثال توجه فرمائید :

```
int a = 2;

int b = 5;
```

int sum = a + b; مجموع a و b در متغیر sum ذخیره سازی می گردد .

برای عملیات ضرب ، تقسیم ، تفریق و باقیمانده به ترتیب : *، / ، - و % می باشد .

عملگر های انتسابی مرکب (compound assignment) دسته بعدی عملگر ها می باشند . مثلا :

```
a *= b;
```

بیانگر اینست که a در b ضرب می شود و نتیجه در a ذخیره می گردد .

عملگر های دیگر این دسته : =+ ، =- ، =/ و =% می باشد .

دسته دیگر عملگر ها ، عملگرهای افزایشی و کاهشی هستند . ;a++ به معنی اینست که یک مقدار به a اضافه و در a ذخیره می گردد و ;a-- نیز یک مقدار از a کم می کند .

عملگر های مقایسه ای برای مقایسه دو متغیر کاربرد دارد . == برقراری تساوی ، => کوچکتر مساوی ، =< بزرگتر مساوی و =! مخالف را بررسی می کند .

عملگر های منطقی به دسته عملگر هایی گویند که در صورت درست بودن true و در غیر اینصورت false بر می گرداند .

! که NOT می باشد ،عملگر نقض است و T را F و F را T می کند .

& که AND می باشد و زمانی خروجی T می دهد که هر دو طرف آن دارای ارزش T باشند .

| که OR خوانده می شود در صورتی F است که هر دو طرف عملگر F باشد .

^ که همان XOR می باشد و مشابه آن عمل می کند .

عملگرهای شرطی : عملگر && و || می باشد . بین دو شرط قرار می گیرند . && عملکرد AND را دارد و || عملکرد OR را دارد .

دستورات کنترلی :

اجرای دستورات برنامه به صورت عادی بالا به پایین می باشد در صورتی که عبارات کنترلی این روال را برهم می زنند و گاهی ایجاد حلقه در روال اجرا و یا اجرای دستور به شرط برقراری شرایط به عنوان روال اجرای برنامه می باشد .

دستورات کنترلی شامل دستورات تصمیم گیری (`if-then`, `if-then-else`, `switch`) ، دستورات حلقه اجرا (`for`, `while`, `do-while`) و دستورات انشعابی (`break`, `continue`, `return`) می باشد .

دستورات تصمیم گیری :

دستور if - else : در صورتیکه شرطی برقرار باشد ، عملی انجام می شود و در غیر آن صورت ، دستور دیگری انجام می پذیرد.

به عنوان مثال :

```
if (شرط)
{
        عملیات 1
}
else
{
        عملیات 2
}
```

دستور switch:

این دستور با انواع داده ای byte ، short ، char و int ، enum ها و کلاس String .

به نمونه کد زیر توجه فرمائید :

```
int month = 8;
String monthString;
switch (month) {
    case 1:  monthString = "January";
             break;
    case 2:  monthString = "February";
             break;
    case 3:  monthString = "March";
             break;
    case 4:  monthString = "April";
             break;
    case 5:  monthString = "May";
             break;
    case 6:  monthString = "June";
             break;
    case 7:  monthString = "July";
             break;
    case 8:  monthString = "August";
             break;
    case 9:  monthString = "September";
             break;
    case 10: monthString = "October";
             break;
    case 11: monthString = "November";
             break;
    case 12: monthString = "December";
             break;
    default: monthString = "Invalid month";
             break;
     return monthString;
```

کد فوق شماره ماه سال را می گیرد و نام آن ماه را بر می گرداند .

دستورات حلقه اجرا :

دستور while *و* do – while *:*

دستور تکرار while اجرای برنامه داخل آکولاد را تا جایی ادامه می دهد که شرط برقرار باشد .

```
while (شرط)
{ عملیات }
```

دستور do – while در مقایسه با دستور while تفاوتی دارد و اینست که یکبار دستور اجرا می شود و سپس شرط while جهت تکرار بررسی می گردد .

```
do {عملیات}
while( شرط );
```

دستور for(;;) *:*

این دستور اجرای عملیات را در یک حلقه محدود تکرار می کند .

```
for (شمارنده ; حد نهایت  ; مقداردهی اولیه)
{ عملیات }
```

نکات : زمانیکه درون حلقه تکرار از ;break استفاده می کنیم بلافاصله کنترل برنامه از حلقه خارج شده و اجرای برنامه از پس از حلقه ادامه می یابد . continue زمانی صدا زده شود ، کنترل برنامه درون حلقه به شرط حلقه جهت بررسی و ادامه حلقه مراجعه می کند . return زمانی صدا زده شود از برنامه خارج می شویم .

Exception :

سیگنالی است که در اثر بروز شرایط غیر منتظره و پیش بینی نشده در زمان اجرا ارسال می گردد . مانند خواندن از فایلی که وجود

ندارد ، ارسال در زمانی که اتصال قطع می باشد و ...

ساختار هندل کردن یک exception به صورت throw and catch می باشد . throw ارسال سیگنال در زمان بروز خطا می باشد و catch عبارتست از انجام عمل صحیح و مناسب در زمان بروز exception .

ماهیت exception در جاوا یک شیء از کلاس java.lang.Throwable می باشد . دو متد ()getMessage و ()printStackTrace محتویات exception را نمایش می دهند .

زیر کلاس های مربوط به کلاس Exception با توجه به علت بروز خطا متفاوت می باشند . مثلا IOException مربوط به خطای ورودی – خروجی می باشد یا FileNotFoundException زمانیکه فایل مورد نظر وجود نداشته باشد اعلام می گردد .

RunTimeException دسته دیگری هستند که در زمان اجرا اعلام می گردد . مثلا تقسیم بر صفر در عملیات ریاضی ، ArrayIndexOutOfBoundException ، NullPointerException ، NumberFormatException و ...

مکانیزم try – catch :

در زبان جاوا ، با استفاده از این مکانیزم می توان exception را هندل کرد .

```
try {                                          // try block
    <statements>
} catch (<exception type1> <parameter1>) {  // catch block
    <statements>
}
```

قسمتی از کد برنامه که احتمال بروز خطا را دارد در بلاک try قرار داده می شود و در صورت exception بلاک catch اجرا می گردد . بلاک finally پس از بلاک های catch نوشته می شود و حتما اجرا خواهد شد .

Assertion : برای تست کردن یک فرض در برنامه به کار می رود . تا زمانی که فرض صحیح باشد ، برنامه اجرا می گردد و در غیر اینصورت JVM خطای assertion error باز می گرداند . به عنوان مثال :

```
Assert num == 20 : “not valid”
System.out.println(“age is ”+num);
```

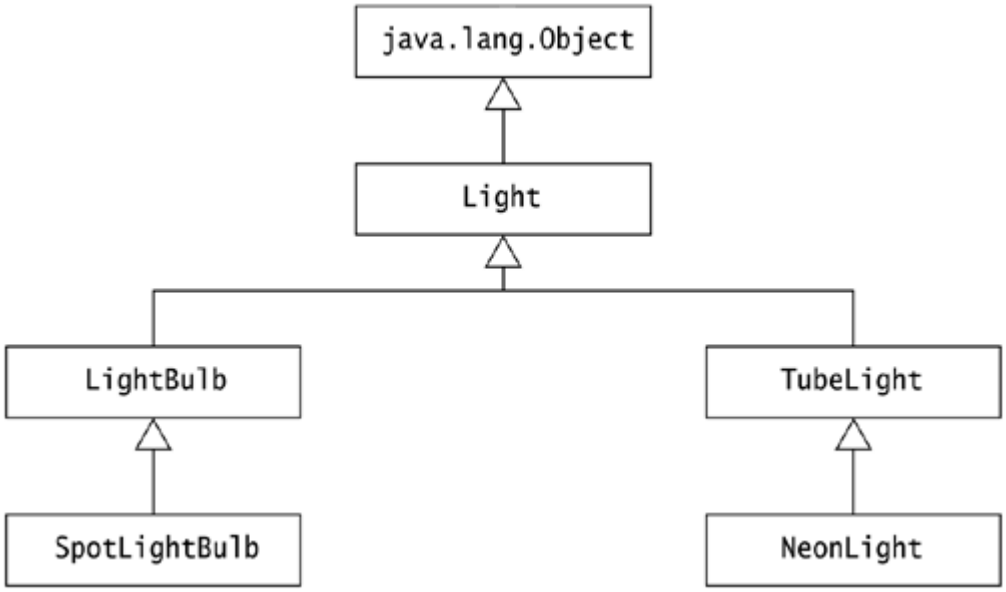
در مثال فوق در صورتیکه num بیست نباشد خطای not valid نمایش داده شده و خطا بر می گردد در غیر اینصورت پیام مورد نظر چاپ می گردد .

برای فعالسازی assertion در تنظیمات vm عبارت ea– قرار دهید .

معماری شیء گرا (Object Oriented) :

این معماری به طور کلی به تمام موجودیت ها در پروژه جاوا به صورت شیء مستقل یا وابسته نگام می کند . معماری قدیمی تر در برنامه نویسی ، structured یا ساخت یافته نام داشت . معماری شیء گرا دارای ویژگی های منحصر بفردی می باشد که در زیر به تشریح آنها می پردازیم :

ارث بری (inheritance) : کلاس جدید اجازه دارد از یک کلاس موجود ارث بری کند . متد های کلاس موجود توسط کلاس فرزند (subclass) می تواند تغییر کند که به این اتفاق overriding گویند . ویژگی های کلاس پدر (super class) عینا در فرزند موجود
می باشد . به شکل زیر توجه کنید :



این شکل بیانگر سلسله مراتب ارث بری در مورد انواع مختلف چراغ می باشد . هر زیر کلاس با فلش به superclass متصل شده است . در زبان جاوا برای تبدیل یک کلاس به subclass از واژه extends پس از نام کلاس و سپس نام superclass استفاده می کنیم .

```
public class TubeLight extends Light
{ … }
```

Overriding Method: مفهوم override به معنی بازنویسی می باشد . این ویژگی به آن معنی است که در فرآیند ارث بری ، زیر
کلاس متد های غیر static کلاس پدر را می تواند باز نویسی کند یعنی بدنه متد را تغییر دهد . لازم به ذکر است که متد باز نویسی شده باید signature (نام و پارامتر های ورودی) کاملا مشابه متد در کلاس پدر داشته باشد .

Overloading Method : زمانیکه متد با نام یکسان در کلاس پدر باشد اما از نظر پارامتر های ورودی متفاوت باشند ، متد Overload شده است .

مخفی سازی (hiding) فیلد : می توان در زیر کلاس ویژگی همنام ویژگی در super class را تعریف کرد .

بدین صورت دسترسی به ویژگی کلاس پدر از طریق فرزند امکانپذیر نمی باشد که به این عمل مخفی سازی ویژگی گویند . تنها در صورتی می توان به

ویژگی پدر دسترسی یافت که از super استفاده نمود .

به مثال چراغ توجه فرمائید :

```
class Light {
    protected String billType = "Small bill";
    protected double getBill(int noOfHours)
              throws InvalidHoursException {
        if (noOfHours < 0)
            throw new NegativeHoursException();
        double smallAmount = 10.0,
               smallBill = smallAmount * noOfHours;
        System.out.println(billType + ": " + smallBill);
        return smallBill;
    }
}
```

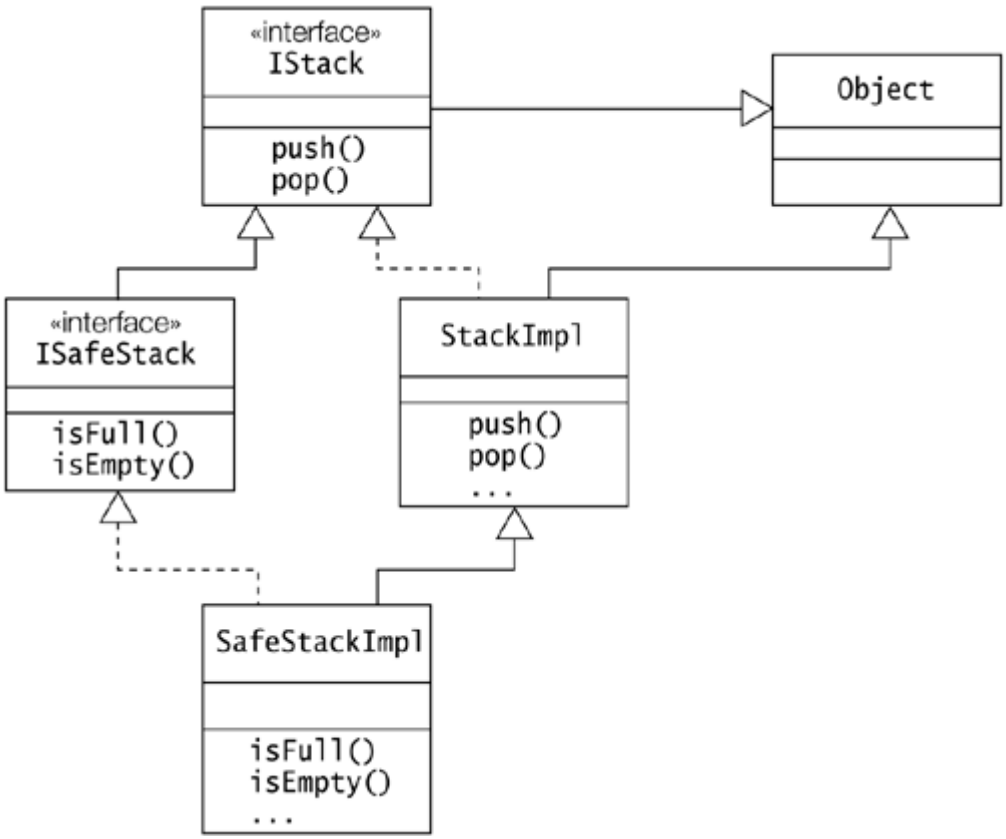
در اینجا کلاس فوق به عنوان superclass می باشد و TubeLight زیر کلاس آن می باشد :

```
class TubeLight extends Light {
    public static String billType = "Large bill"; // (4) Hiding static field.
    public double getBill(int noOfHours)
           throws ZeroHoursException {     // (5) Overriding instance method.
        if (noOfHours == 0)
```

```
            throw new ZeroHoursException();
        double largeAmount = 100.0,
               largeBill = largeAmount * noOfHours;
        System.out.println(billType + ": " + largeBill);
        return largeBill;
    }
    public double getBill() {                    // (7) Overloading method.
        System.out.println("No bill");
        return 0.0;
    }
}
```

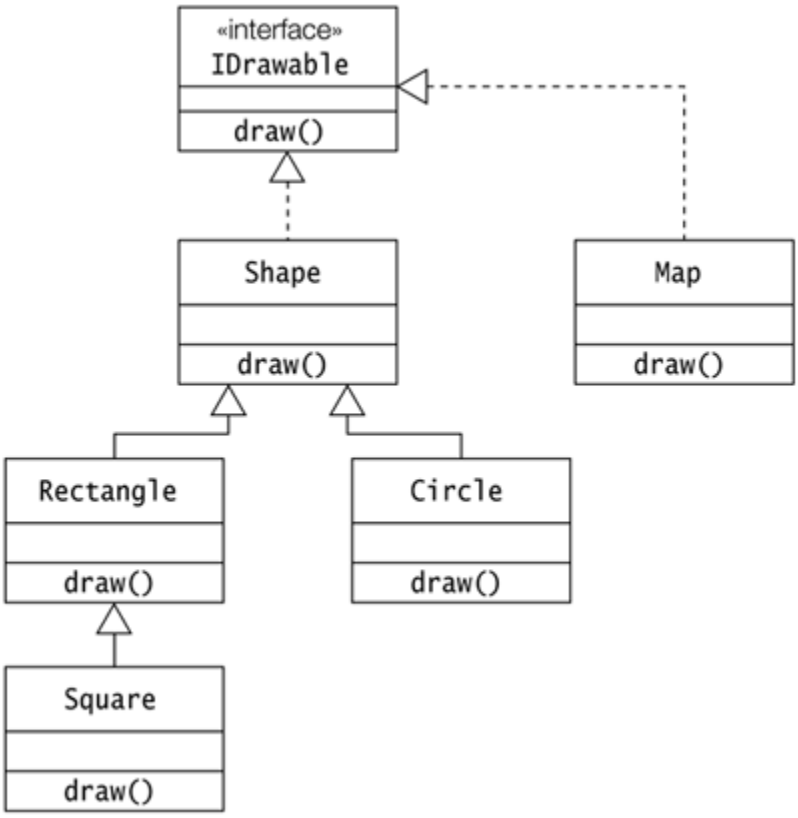
نکته : constructor در کلاس هیچگاه override نمی شود اما می توان آنرا overload کرد و این عمل تنها در همان کلاس انجام می پذیرد . در constructor برای دسترسی به فیلد های موجود در سطح کلاس از this که به معنی آبجکت جاری که در آن قرار داریم که در این مثال ، کلاس مربوطه می باشد ، استفاده می شود . از super برای دسترسی به constructor مربوط به پدر درون Constructor فرزند می توان استفاده کرد .

Interface: مشابه کلاس است اما تفاوت هایی با کلاس دارد : یک اینترفیس می تواند دارای فیلد و signature متد باشد یعنی تنها نام ، پارامتر ورودی و throw exception و متد های آن نباید بدنه پیاده سازی داشته باشند .
کلاسی که از یک اینترفیس ارث بری می کند (در اینجا از implement استفاده می شود) باید تمامی متد ها را پیاده سازی نماید .
نمودار زیر جایگاه اینترفیس را نشان می دهد :



چند فرمی (Polymorphism) :

یکی از خواص OOP می باشد به این معنی که به تعداد کلاسهایی که یک interface را پیاده سازی می کنند می توان متد های آن Interface را با بدنه های مختلفی پیاده سازی کرد . شکل زیر در مورد اشکال هندسی خاصیت چند فرمی را به تصویر می کشد .



: Encapsulation

دسترسی به ویژگی ها و فیلد های یک کلاس تنها از طریق متد getXXX() و setXXX() مقدور می باشد .

در زیر کلاس student نمایش داده می شود :

```
public class Student {
   private String name;
   private String idNum;
   private int age;

   public int getAge() {
      return age;
   }

   public String getName() {
```

```
      return name;
   }

   public String getIdNum() {
      return idNum;
   }

   public void setAge( int newAge) {
      age = newAge;
   }

   public void setName(String newName) {
      name = newName;
   }

   public void setIdNum( String newId) {
      idNum = newId;
   }
}
```

حال برای دسترسی به ویژگی های name ، idNum و age از خارج از کلاس فوق تنها از متدهای setter و getter مربوط به آنها باید استفاده کرد .

```
public static void main(String args[]) {
      Student std = new Student();
      std.setName("James");
      std.setAge(20);
      std.setIdNum("12343ms");

      System.out.print("Name : " + std.getName() + " Age : " + std.getAge());
```

```
    }
```

: Nested Class

کلاسی که درون کلاس دیگری نوشته شود nested و کلاس اصلی outer class می باشد .

```
class Outer_Demo {
   class Nested_Demo {
   }
}
```

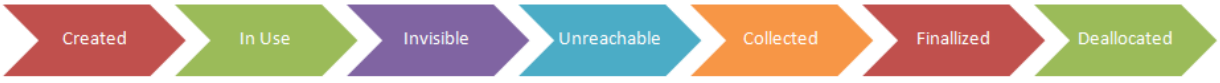
Nested class به دو دسته تقسیم می شوند : static nested class که عبارتند از اعضا static و non-static nested class که اعضای غیر static می باشند . به کلاسهای غیر static ، inner class نیز می گویند.

Inner class یک کلاس ساده درون کلاس دیگر است و بر خلاف کلاس معمولی قابلیت private بودن را دارد که در اینصورت از خارج از کلاس outer نمی توان به آن دسترسی داشت .

Static nested class : کلاس static که درون یک کلاس outer نوشته می شود و مانند یک عضو static در یک کلاس بدون نیاز به نمونه سازی از کلاس outer در خارج از آن می توان به این کلاس دسترسی داشت .

طول عمر شیء :

از زمانیکه یک شیء بوجود می آید تا حذف کامل آن از حافظه را طول عمر گویند. طول عمر یک شیء جاوا در شکل زیر آمده است :



: Garbage Collection

در جاوا ، garbage به معنی شیء است که هیچ ارجاعی (reference) به آن وجود ندارد . عمل آزاد سازی حافظه و از بین بردن اشیاء بلا استفاده در جاوا را garbage collection گویند که به صورت خودکار انجام می پذیرد .

در سه حالت یک شیء به garbage تبدیل می شود :

1- null reference به عنوان مثال :

```
1. Employee e=new Employee();
2. e=null;
```

2- اتصال ارجاع به شیء دیگر :

```
1. Employee e1=new Employee();
2. Employee e2=new Employee();
3. e1=e2;
```

3- شی ناشناخته :

New Employee();

نکته : gc تنها اشیائی که با کلمه new ایجاد شده باشند را پاک می کنند . وظیفه پاکسازی به عهده JVM می باشد .

اشیائی که new ندارند با استفاده از متد ()finalize پاک می شوند که به صورت دستی باید انجام پذیرد . هر کلاس جاوا متد finalize را پیاده سازی میکند .

اگر به صورت دستی بخواهیم garbage collector را صدا بزنیم از ()System.gc استفاده می کنیم .

مروری بر کلاس های اصلی جاوا :

: Java.lang

این پکیج ارائه دهنده کلاس هایی است که بنیاد طراحی یک پروژه جاوا را تشکیل می دهند .مهمترین کلاس آن Object

می باشد . در واقع کلاس Object پدر تمامی کلاسهای موجود در جاوا می باشد بدین معنی که همه کلاسهای primitive و کلاس های ایجاد شده توسط برنامه نویس می توانند در یک Object ذخیره شند .

کلاس Wrapper : مکانیزمی را ارائه می دهد که می توان از طریق آن می توان نوع primitive را به object و بالعکس تبدیل نمود . در اینجا با دو مفهوم autoboxing و unboxing آشنا می شویم . از j2se5 به بعد ، تبدیل خودکار نوع primitive به object را autoboxing و بر عکس این عمل را unboxing گویند .

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

مثالی از autoboxing در زیر نمایش داده می شود .

```
public class WrapperExample1{
public static void main(String args[]){
//Converting int into Integer
```

```
int a=20;
Integer i=Integer.valueOf(a);//converting int into Integer
Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally

System.out.println(a+" "+i+" "+j);
}}
```

مثالی از unboxing در زیر نمایش داده می شود :

```
public class WrapperExample2{
public static void main(String args[]){
//Converting Integer to int
Integer a=new Integer(3);
int i=a.intValue();//converting Integer to int
int j=a;//unboxing, now compiler will write a.intValue() internally

System.out.println(a+" "+i+" "+j);
}}
```

File و Stream :

پکیج java.io شامل همه کلاسهای مورد نیاز در انجام عملیات input/out می باشد .

Stream عبارتست از توالی داده . به دونوع تقسیم می گردد :

inputStream : خواندن داده از سورس را به عهده دارد .

outPutStream : برای نوشتن داده کاربرد دارد .

به مثال زیر توجه کنید :

```
public static void main(String args[]) throws IOException {
      FileInputStream in = null;
      FileOutputStream out = null;
```

```
    try {
       in = new FileInputStream("input.txt");
       out = new FileOutputStream("output.txt");

       int c;
       while ((c = in.read()) != -1) {
          out.write(c);
       }
    }finally {
       if (in != null) {
          in.close();
       }
       if (out != null) {
          out.close();
       }
    }
 }
```

ابتدا دو متغیر in و out جهت ذخیره سازی inputstream و outputstream برای خواندن از یک فایل و ایجاد و نوشتن در فایل مقصد ایجاد می کنیم . متغیر c که عدد در خود نگه می دارد و در صورت به پایان رسیدن فایل مبداء متد read از شیء inputstream عدد 1- را در آن ذخیره می کند و در این لحظه عمل خواندن و نوشتن پایان می یابد.

: Filter Stream

در جاوا نسخه 7 به قبل ، زمانیکه می خواستیم در لیستی عبارتی را بیابیم مجبور به پیمایش لیست در حلقه و بررسی تک تک المانهای آن بودیم . در جاوا ورژن 8 ، قابلیت filter کردن لیست وجود دارد. البته لیست ابتدا باید به stream تبدیل شود .

مثال :

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
public class NowJava8 {
    public static void main(String[] args) {
        List<String> lines = Arrays.asList("mohammad", "sadegh", "saberi");
        List<String> result = lines.stream()   // convert list to stream
   .filter(line -> !"saberi".equals(line))     // find every things except saberi
   .collect(Collectors.toList()); // collect the output and convert streams to a List
        result.forEach(System.out::println);            //output : mohammad, sadegh
    }

}
```

سطح دسترسی به فایل و دایرکتوری :

سطح دسترسی به فایل در جاوا به سه دسته تقسیم می شود :

1- file.canExecute(); : در صورتیکه فایل قابل اجرا باشد true و در غیر اینصورت false بر می گرداند .
2- file.canWrite(); : در صورتیکه فایل قابل نوشتن باشد true و در غیر اینصورت false بر می گرداند .
3- file.canRead(); : در صورتیکه فایل قابل خواندن باشد true و در غیر اینصورت false بر می گرداند .

به طبع آن برای اعطای سطح دسترسی از متد های file.setExecutable(boolean); ،
file.setReadable(boolean); و file.setWritable(boolean); استفاده می شود .

برای ایجاد یک دایرکتوری جدید به عنوان مثال :

```
new File("C:\\Directory1").mkdir();
```

یک دایرکتوری به نام Directory1 در درایو C ایجاد می کند .

اگر بخواهیم چند دایرکتوری درون هم ایجاد کنیم به شکل زیر عمل می کنیم :

```
new File("C:\\Directory2\\Sub2\\Sub-Sub2").mkdirs()
```

در JDK 7 پکیج های جدیدی برای ساختن فایل و دایرکتوری معرفی شده اند که عبارتند از :

`java.nio.file.Paths` و `java.nio.file.Files`

به مثال زیر توجه کنید :

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


        File file = new File("C:\\Directory1");
        if (!file.exists()) {
            if (file.mkdir()) {
                System.out.println("Directory is created!");
            } else {
```

```
                System.out.println("Failed to create directory!");
            }
        }

        File files = new File("C:\\Directory2\\Sub2\\Sub-Sub2");
        if (!files.exists()) {
            if (files.mkdirs()) {
                System.out.println("Multiple directories are created!");
            } else {
                System.out.println("Failed to create multiple directories!");
            }
        }
    }
```

برای ایجاد فایل جدید ، ابتدا یک شیء از کلاس File ایجاد می کنیم و متد createNewFile آنرا ایجاد می کنیم :

```
public static void main( String[] args )
    {
        try {
              File file = new File("c:\\newfile.txt");
              if (file.createNewFile()){
                System.out.println("File is created!");
              }else{
```

```
                System.out.println("File already exists.");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

جهت تغییر نام فایل و دایرکتوری از متد renameTo استفاده می کنیم .

برای حذف کردن یک دایرکتوری از متد delete و برای حذف کردن یک فایل از file.delete استفاده می کنیم .

خواندن محتویات فایل :

برای خواندن فایل ، ابتدا باید آبجکت file را با FileInputStream تبدیل کنیم و سپس با استفاده از BufferReader فایل مورد نظر را خط به خط بخوانیم . مثال :

```
public static void main(String[] args) {
        InputStream inputStream = null;
        BufferedReader br = null;
        try {
            inputStream = new FileInputStream("/Users/sadegh/Downloads/file.js");
            br = new BufferedReader(new InputStreamReader(inputStream));
            StringBuilder sb = new StringBuilder();
            String line;
```

```
                while ((line = br.readLine()) != null) {
                        sb.append(line);
                }
                System.out.println(sb.toString());
                System.out.println("\nDone!");

        } catch (IOException e) {
                e.printStackTrace();
        } finally {
                if (inputStream != null) {
                        try {
                                inputStream.close();
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                }
                if (br != null) {
                        try {
                                br.close();
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                }
        }
```

```
    }
```

مثال زیر نحوه خواندن از یک فایل و نوشتن در فایل دیگر توسط FileOutPutStream را نمایش می دهد :

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

        InputStream inputStream = null;

        OutputStream outputStream = null;

        try {

                // read this file into InputStream

                inputStream = new
FileInputStream("/Users/sadegh/Downloads/holder.js");

                // write the inputStream to a FileOutputStream

                outputStream =

                    new FileOutputStream(new File("/Users/sadegh/Downloads/holder-
new.js"));



                int read = 0;

                byte[] bytes = new byte[1024];

                while ((read = inputStream.read(bytes)) != -1) {

                        outputStream.write(bytes, 0, read);

                }

                System.out.println("Done!");

        } catch (IOException e) {

                e.printStackTrace();
```

```
        } finally {
                if (inputStream != null) {
                        try {
                                inputStream.close();
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                }
                if (outputStream != null) {
                        try {
                                // outputStream.flush();
                                outputStream.close();
                        } catch (IOException e) {
                                e.printStackTrace();}}}
```

کلاس Buffered Reader و خواندن ورودی کاربر

مثال :

```
BufferedReader br = null;

try {

    br = new BufferedReader(new InputStreamReader(System.in));


    while (true) {

        System.out.print("Enter something : ");
        String input = br.readLine();

        if ("q".equals(input)) {
            System.out.println("Exit!");
            System.exit(0);
        }

        System.out.println("input : " + input);
        System.out.println("-----------\n");
    }

} catch (IOException e) {
    e.printStackTrace();
} finally {
    if (br != null) {
        try {
            br.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

در مثال فوق از کلاس BufferedReader برای خواندن خط نوشته شده توسط کاربر استفاده می گردد .

آبـجکت `new InputStreamReader(System.in)` مسیر خواندن ورودی کاربر از محیط command را مشخص می کند .

Object Serialization :

مفهوم serialize کردن یک آبجکت ، ایجاد قابلیت تبدیل شدن آبجکت به رشته بایتی برای انتقال در محیط شبکه از طریق سوکت در تکنولژی هایی مانند RMI را فراهم می سازد . کلاس ObjectOutPutStream آبجکت را به فرمت

stream می سازد . کلاس ObjectInputStream برای تبدیل stream به آبجکت بکار می رود . در حقیقت کلاس ObjectOutPutStream ، کلاس را serialize می کند و کلاس ObjectInputStream آنرا deserialize می کند . واضح است که تنها شی از کلاس serializable از این امکان بهره می برد .

به مثال زیر توجه کنید :

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

      String s = "Hello World";
      byte[] b = {'e', 'x', 'a', 'm', 'p', 'l', 'e'};

      try {


         FileOutputStream out = new FileOutputStream("test.txt");
         ObjectOutputStream oout = new ObjectOutputStream(out);

         oout.writeObject(s);
         oout.writeObject(b);
         oout.flush();


        ObjectInputStream ois = new ObjectInputStream(new
      FileInputStream("test.txt"));


         System.out.println("" + (String) ois.readObject());


         byte[] read = (byte[]) ois.readObject();
         String s2 = new String(read);
         System.out.println("" + s2);


      } catch (Exception ex) {
         ex.printStackTrace();
      }

   }
```

Regular Expression :

یک الگوی جستجو برای رشته (string) می باشد . کاربرد آن در جستجو در رشته و تغییر آن می باشد . برای استفاده

از آن از regex استفاده می کنیم . در جدول زیر علائم مربوط به regular expression به همراه توضیحات آورده شده است :

Regular Expression	Description
\d	یک رقمی بین 0 تا 9
\D	یک کاراکتر غیر رقمی که در بازه 0 تا 9 نباشد
\s	یک کاراکتر فاصله
\S	یک کاراکتر غیر از فاصله
\w	یک کاراکتر کلمه شامل 0 تا 9 ، a تا z ، A تا Z
\W	یک کاراکتر بجز w
\S+	چندین کاراکتر غیر فاصله
\b	جایگاه مرز کاراکتر غیر کلمه را نشان می دهد

در بحث regular expression با علائمی به نام کمیت سنج سروکار داریم که در جدول زیر معرفی می گردند :

Regular Expression	Description	Examples
*	هیچ یا هر چند بار	X وجود دارد یا ندارد
+	یک یا چند بار	X حداقل یکبار باید وجود داشته باشد
?	هیچ یا یک بار	X? : x فقط یکبار باید وجود داشته باشد
{X}	X بار	\d{3} : پیدا کردن 3 رقم .{10} : پیدا کردن 10 تا از هر کاراکتری
{X,Y}	بین X بار تا Y بار	عدد حداقل x تا و حداکثر y تا باشد

در مثال زیر می خواهیم تمامی فاصله های بین کاراکتر ها با . یا , در صورت وجود را حذف کند . ابتدا الگو ها را گروه

بندی مشخص می کنیم . گروهبندی با پرانتز انجام می پذیرد و ارجاع به گروه ها از طریق $ و شماره گروه که طبق

ترتیب قرارگیری از 1 شروع می شود انجام می پذیرد .

```
String EXAMPLE_TEST = "Hello ,World .";
String pattern = "(\\w)(\\s+)([\\.,])";
System.out.println(EXAMPLE_TEST.replaceAll(pattern, "$1$3"));

Result : "Hello,World."
```

کاربرد regex در string :

جدول زیر حاوی متد های مورد استفاده در regex می باشد :

Method	Description
`s.matches("salam")`	ارزیابی می کند اگر همه رشته salam در s وجود داشت true بر می گرداند .
`s.split("salam")`	آرایه از زیر رشته های s می سازد و تا به رشته salam برخورد می کند آنرا حذف کرده و مابقی رشته را در زیر رشته جدید قرار می دهد .
`s.replaceFirst("salam"), "khodahafez"`	به اولین salam که برخورد آنرا با khodahafez جایگزین می کند .
`s.replaceAll("salam"), "khodahafez"`	همه salam ها را با khodahafez جایگزین می کند .

متد format :

متد java.lang.String.format وظیفه ساختار بندی رشته ورودی به متد با توجه به الگوی ساختاری تعریف شده برای آنرا دارد .

Signature مربوط به این متد عبارتست از :

public static String format(string format, Object… args)

exception های احتمالی این متد IllegalFormatException که در صورتی رخ می دهد که آرگومانهای ورودی با الگوی فرمت مطابقت نداشته باشد و NullPointerException در صورتی که الگو null باشد .

مثال : برای نمونه از stringbuilder برای ذخیره رشته استفاده می کنیم و آنرا در قالب یک شی formatter ذخیره می کنیم . سپس از متد format آن برای ساختار دهی بهره می گیریم .

```
StringBuilder sb = new StringBuilder();
Formatter formatter = new Formatter(sb);
Formatter.format(“PI = %f%n”,Math.PI);
System.out.println(sb.toString());
```

این مثال عدد پی را نمایش می دهد .

به عنوان مثال دیگر ، گاهی نیاز داریم رقم با جداکننده سه تایی جدا شود . در این صورت مانند زیر عمل می کنیم :

```
String.format(“%,d”,1000000);
```

خروجی عبارتست از :

```
1,000,000
```

و یا گاهی احتیاج داریم رشته با طول ثابت ارسال شود و در صورت داشتن طول کمتر ، آنرا padding کنیم :

```
String.format(“%020d”,88);
```

خروجی عبارتست از رشته به طول 20 که دو رقم سمت راست 88 و سایر ارقام با صفر پر می شوند .

Thread :

به طور اجمالی ، thread را می توان واحد پردازش فرآیند های برنامه تعریف کرد به صورتی که پردازش های سنگین

و پیچیده را می توان به چند thread سپرد و آنها را به صورت هم زمان اجرا و در انتها پاسخ را از مجموع تمامی آنها دریافت و به عنوان یک پاسخ به کاربر برگرداند که به این قابلیت multithreading گفته می شود .

در برنامه دو نوع thread وجود دارد : user thread که همان پردازش های کاربر می باشد . در حقیقت main به عنوان اولین user thread تعریف می شود . زمانیکه همه thread های کاربر انجام شد ، jvm برنامه را به اتمام می رساند .

ایجاد thread با استفاده از پیاده سازی اینترفیس Runnable و یا ارث بری از کلاس Thread امکانپذیر می باشد . در صورتیکه تنها نیاز به ایجاد یک thread باشد ارث بری از کلاس Thread استفاده می کنیم اما در مواردی که بیش از این مورد نیاز باشد از پیاده سازی Runnable استفاده می کنیم .

```
Thread t = new Thread(new Runnable(){

    @Override

    public void run() {

    }

});
```

در جاوا ورژن 8 ، به صورت زیر نیز می توان thread را ایجاد نمود :

```
Thread t = new Thread(() -> {System.out.println("My
Runnable");});

t.start();
```

در روش دوم ، کلاسی از فرزندان کلاس Thread ایجاد می کنیم :

```
public class MyThread extends Thread {


    public MyThread(String name) {

        super(name);

    }


    @Override

    public void run() {

        System.out.println("MyThread - START
"+Thread.currentThread().getName());

        try {

            Thread.sleep(1000);

            doDBProcessing();

        } catch (InterruptedException e) {

            e.printStackTrace();

        }

        System.out.println("MyThread - END
"+Thread.currentThread().getName());

    }


    private void doDBProcessing() throws InterruptedException {

        Thread.sleep(5000);

}}
```

در مثال فوق ، متد run در کلاس thread بازنویسی می شود و عملیات مربوط به thread درون این متد انجام می پذیرد.

در multithreading زمانی دو یا چند thread نیازمند دسترسی به یک منبع می باشند و در این حالت دسترسی باید به ترتیب به thread ها اختصاص یابد تا اعتبار منابع زیر سوال نرود . در این حالت نیاز به هم آهنگ سازی یا Synchronization می باشد .

جاوا با بلاک synchronized این قابلیت را ایجاد می کند .

```
synchronized(objectidentifier) {
   // دسترسی به منابع مشترک
}
```

برای متدها ویژگی synchronized قبل از نوع داده خروجی متد قرار می گیرد . برای مثال :

```
synchronized void printNumber (int n){
    for(int i=1;i<=5;i++){
        System.out.println(n*i);
        try{
            Thread.sleep(400);
        }catch(Exception e){System.out.println(e);}
    }

}
```

این متد در صورتیکه توسط دو thread به صورت همزمان مورد دسترسی و تغییر قرار گیرد ابتدا thread یی که زودتر اجرا شده است این متد را block می کند و پس از پایان آن thread بعدی اجرا می شود .

: Generic

متد های generic : متد هایی هستند که آرگومانهای ورودی آنها دارای نوع مشخصی نمی باشند و نوع آنها در زمان اجرا با توجه به داده های ارسالی به متد مشخص می گردد . در هنگام تعریف متد پارامتری که نوع آن مشخص نمی باشد را غالبا با یک حرف انگلیسی بزرگ مشخص می کنیم . مثال زیر چگونگی تعریف متد generic را نشان می دهد :

```
public static < E > void printArray( E[] inputArray ) {
      // نمایش المان های آرایه
      for(E element : inputArray) {
         System.out.printf(element);
      }
      System.out.println();
   }
```

کلاس های generic مشابه کلاس های معمولی تعریف می شوند با این تفاوت که نوع داده در براکت روبروی نام کلاس درج می شود و با توجه به حرف انگلیسی انتخاب شده درون کلاس از آن به عنوان نوع داده استفاده می شود . به عنوان مثال :

```
public class Box<T> {
   private T t;
   public void add(T t) {this.t = t;}
   public T get() {return t;}
   public static void main(String[] args) {
      Box<Integer> integerBox = new Box<Integer>();
      Box<String> stringBox = new Box<String>();
      integerBox.add(new Integer(10));
      stringBox.add(new String("Hello World"));

      System.out.printf("Integer Value : ", integerBox.get());
      System.out.printf("String Value : ", stringBox.get());
   }}
```

Collection و Map

اینترفیس Comparable : این اینترفیس دارای یک متد به نام compareTo می باشد :

Public int compareTo(Object obj)

برای مقایسه دو آبجکت به کار می رود .

مثال :

```
class Student implements Comparable<Student>{
int num;
String name;
int age;
Student(int num,String name,int age){
this. num = num;
this.name=name;
this.age=age;
}

public int compareTo(Student st){
if(age==st.age)
return 0;
else if(age>st.age)
return 1;
else
return -1;
}
}
```

کلاس Collections : متد های static یی ارائه میدهد تا با استفاده از آنها بتوان المان های مجموعه را مرتب سازی کرد .

:Iterator

برای کار با المان های یک کالکشن مانند اضافه و حذف کردن المان ها و نمایش المانها و در کل برای پیمایش یک

کالکشن می توان از iterator استفاده کرد .

به مثال زیر دقت کنید :

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

// ArrayList یک collection می باشد
      ArrayList al = new ArrayList();

      // افزودن المان به لیست
      al.add("C");
      al.add("A");
      al.add("E");
      al.add("B");
      al.add("D");
      al.add("F");

// استفاده از iterator برای نمایش محتویات لیست
      System.out.print("Original contents of al: ");
      Iterator itr = al.iterator();

      while(itr.hasNext()) {
         Object element = itr.next();
         System.out.print(element + " ");
      }
      System.out.println();



// با استفاده از متد listIterator می توان arrayList را به ListIterator تبدیل نمود .
      ListIterator litr = al.listIterator();
```

```
      while(litr.hasNext()) {
         Object element = litr.next();
         litr.set(element + "+");
      }
      System.out.print("Modified contents of al: ");
      itr = al.iterator();

      while(itr.hasNext()) {
         Object element = itr.next();
         System.out.print(element + " ");
      }
      System.out.println();


      System.out.print("Modified list backwards: ");

      while(litr.hasPrevious()) {
         Object element = litr.previous();
         System.out.print(element + " ");
      }
      System.out.println();
   }
```

خروجی قطعه کد بالا عبارتست از :

```
Original contents of al: C A E B D F
Modified contents of al: C+ A+ E+ B+ D+ F+
```

```
Modified list backwards: F+ D+ B+ E+ A+ C+
```

Array : یک ساختار برای ذخیره داده از نوع یکسان و به طول ثابت می باشد .

برای کار با آرایه ، باید یک متغیر تعریف شود و آنرا به یک آرایه ارجاع دهیم . نوع متغیر مذکور ، نوع داده داخل آرایه می باشد .

نحوه تعریف به شکل زیر می باشد :

```
dataType[] myArray;
یا
dataType myArray[];
```

سپس به صورت زیر initial می شود :

myArray = new datatype[array size];

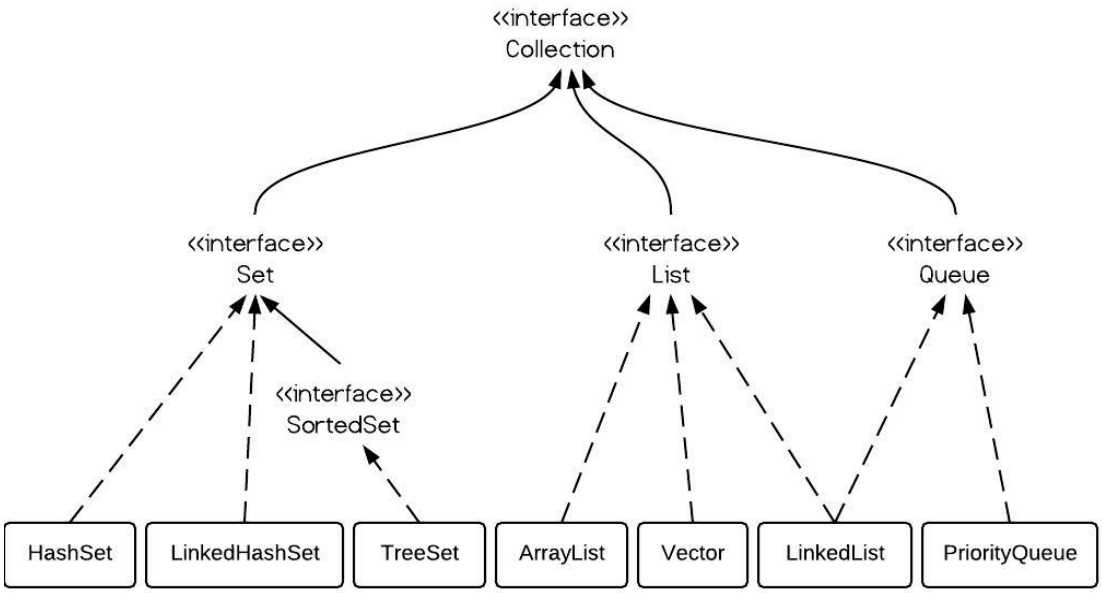
اگر المان های آرایه مشخص باشند می توان المان ها را مستقیما در آرایه قرار داد :

Int[] numbers = new int[20];

numbers = {1,55,84,200,65,388,10,54,7};

پیمایش در آرایه می تواند از طریق حلقه های تکرار صورت پذیرد .

در نمودار زیر ساختار سلسله مراتبی کلاس Collection و زیر کلاس های آن نمایش داده می شود که در ادامه به معرفی آنها خواهیم پرداخت :



Set : نوعی از کالکشن می باشد که داده تکراری در آن وجود ندارد و نیز داده ها با ترتیب خاصی در کنار هم قرار نمی گیرند .

کلاس Collection سه نوع set را پیاده سازی کرده است :

1- HashSet : ابتدایی ترین نوع set که کاربرد فراوانی دارد و هیچ ترتیبی در قرار گیری المانها در آن وجود ندارد .

2- LinkedHashSet : نوعی set که قرار گیری المان ها بر اسا ترتیب ورود آنها می باشد .

3- TreeSet : ترتیب قرار گیری آنها بر اساس مقادیر موجود در آنها می باشد .

مثال :

```
public static void main(String args[]) {
      int count[] = {34, 22,10,60,30,22};
      Set<Integer> mySet = new HashSet<Integer>();
      try {
         for(int i = 0; i < 5; i++) {
            set.add(count[i]);
         }
         System.out.println(mySet);

         TreeSet sortedSet = new TreeSet<Integer>(mySet);
         System.out.println("The sorted list is:");
         System.out.println(sortedSet);

         System.out.println("The First element of the set is: "+
(Integer)sortedSet.first());
         System.out.println("The last element of the set is: "+
(Integer)sortedSet.last());
      }
      catch(Exception e) {}
   }
```

خروجی متد فوق عبارتست از :

```
[34, 22, 10, 60, 30]
The sorted list is:
[10, 22, 30, 34, 60]
The First element of the set is: 10
The last element of the set is: 60
```

NavigableSet زیر نوع SortedSet می باشد و علاوه بر دارا بودن قابلیت ترتیب بندی ، قابلیت navigation نیز دارد .

List : کلاس list از فرزندان کلاس collection می باشد و المانهای خود را در یک توالی نگهداری می کند .
می توان در لیست ، المان جدید اضافه کرد و دسترسی به هر المان لیست با داشتن شماره خانه آن (ایندکس که از صفر شروع می شود) مقدور می باشد . یک لیست می تواند المان تکراری داشته باشد .
مثال :

```
public static void main(String[] args) {
      List a1 = new ArrayList();
      a1.add("Zara");
      a1.add("Mahnaz");
      a1.add("Ayan");
      System.out.println(" ArrayList Elements");
      System.out.print("\t" + a1);

      List l1 = new LinkedList();
      l1.add("Zara");
      l1.add("Mahnaz");
      l1.add("Ayan");
      System.out.println();
      System.out.println(" LinkedList Elements");
      System.out.print("\t" + l1);
   }
```

در باره سه زیر کلاس List مطالبی وجود دارد . arrayList دارای قابلیت تغییر اندازه می باشد و به ازای اضافه شدن هر المان جدید ، طول آن اضافه می گردد . دسترسی به المانهای آن نیز با استفاده از متدهای get و set امکانپذیر می باشد .
کلاس LinkedList نوع دیگری از List می باشد که performance آن در اضافه و حذف المان در مقایسه با arrayList بیشتر است اما در دسترسی به المان های آرایه در مقایسه با arrayList کمتر می باشد .
کلاس Vector مشابه ArrayList بوده و علاوه بر آن قابلیت synchronized را دارد .

Queue : ساختار متوالی ذخیره داده با مکانیزم FIFO (First In First Out) که عبارتست از المانی که زودتر به صف ملحق شده است زودتر خارج می گردد . PriorityQueue مشابه queue می باشد اما از قاعده fifo پیروی نمی کند .

Map : کلاسی که داده را در ساختار کلید / مقدار (Key / Value) ارائه می دهد .

متد های این کلاس به شرح زیر می باشد :

Method & Description
void clear( ) همه کلید / مقدار های map را پاک می کند .
boolean containsKey(Object k) در صورتی که کلید k در map موجود باشد true و در غیر اینصورت false بر می گرداند .
boolean containsValue(Object v) در صورتیکه مقدار v در map وجود داشته باشد true و در غیر اینصورت false بر میگرداند.
Set entrySet( ) یک آبجکت از نوع Set شامل تمامی داده های Map بر می گرداند .
boolean equals(Object obj) در صورتیکه obj یک map باشد و شامل تمامی داده های موجود در Map مورد مقایسه باشد true و در غیر اینصورت false بر می گرداند .
Object get(Object k) مقدار متناظر با کلید k را بر می گرداند .
int hashCode( ) hashcode مربوط به map را بر می گرداند .
boolean isEmpty( ) در صورتیکه map خالی باشد true و در غیر اینصورت false بر می گرداند .
Set keySet( ) یک شی set شامل تمامی کلید های map بر می گرداند .

Object put(Object k, Object v) یک جفت کلید مقدار با مقادیر k و v در map قرار می دهد . در صورتیکه map حاوی کلید k باشد ، مقدار آنرا با v جایگزین می کند و در غیر اینصورت یک جفت جدید در map ایجاد می کند .
void putAll(Map m) همه محتویات m را در map قرار می دهد .
Object remove(Object k) جفت کلید / مقداری که کلید آن k باشد را از map حذف می کند .
int size( ) تعداد جفت k/v های موجود در map را بر می گرداند .
Collection values( ) یک شی از نوع collection حاوی مقادیر موجود در map را بر می گرداند .

بعضی از ویژگی های اضافه شده در JAVA8 :

1- متد foreach در اینترفیس Iterable

نسخه 8 یک متد foreach در اینترفیس `java.lang.Iterable` معرفی کرده است که در هنگام نوشتن کد بیشتر تمرکز ما به جای جزئیات پیمایش بر روی منطق پروژه می باشد .

این متد آبجکت **`java.util.function.Consumer`** را به عنوان آرگومان می گیرد . به مثال زیر توجه کنید :

```
// ایجاد یک لیست
List<Integer> myList = new ArrayList<Integer>();
            for(int i=0; i<10; i++) myList.add(i);
```

```
// پیمایش لیست
myList.forEach(new Consumer<Integer>() {
public void accept(Integer t) {
System.out.println("forEach Value::"+t);
                    }
            });
```

حال می توان برای جدا سازی بخش پیمایش از پروژه ، ابتدا یک کلاس فرزند Consumer مانند زیر ایجاد نمود :

```
class MyConsumer implements Consumer<Integer>{
      public void accept(Integer t) {
            System.out.println("Consumer impl Value::"+t);
      }}
```

حال با قطعه کد زیر آنرا فراخوانی کرد :

```
MyConsumer action = new MyConsumer();
myList.forEach(action);
```

2- پیاده سازی متد در interface

در نسخه 8 در اینترفیس دو نوع متد default و static را می توان پیاده سازی کرد . ابتدا اینترفیس را ایجاد می کنیم :

```
@FunctionalInterface
public interface MyInterface {

    static void print(String str){
        System.out.println("Printing "+str);
    }
```

```
    default void log(String str){
        System.out.println("Printing "+str);
    }

}
```

برای فراخوانی متد static باید متد را همراه با نام کلاس اینترفیس صدا بزنیم .

```
MyInterface.print("salam");
```

برای فراخوانی متد default ابتدا کلاس باید اینترفیس را پیاده سازی نماید و در بده متد آنرا فراخوانی کرده و یا تغییراتی در خروجی متد اعمال کنیم .

```
@Override
      public void log(String str){
            System.out.println("MyClass logging::"+str);
      }
```

3- اینترفیس Functional و lambda expression

نوعی از اینترفیس که با FunctionalInterface annotation مشخص می شود و دارای تنها یک متد abstract می باشد . در مورد قبل اینترفیس MyInterface از همین نوع است و بهترین مثال برای این دسته از کلاس ها اینترفیس Runnable با متد abstract خود به نام run می باشد .

بزرگترین مزیت این دسته از اینترفیس ها قابلیت استفاده از lambda expression در آنها می باشد .

به عنوان مثال ، اینترفیس Runnable در یک کلاس اینگونه نمونه سازی شده است :

```
Runnable r = new Runnable(){
                  @Override
                  public void run() {
                        System.out.println("My Runnable");
                  }};
```

حال می توان با lambda expression قطعه کد بالا را اینگونه بازنویسی کرد :

```
Runnable r1 = () -> {
                    System.out.println("My Runnable");
             };
```

در مثالی دیگر ، اینترفیس زیر را در نظر بگیرید :

```
@FunctionalInterface
public interface MyInterface1 {
      void method1(String str);
}
```

حال از یک کلاس ، اینترفیس فوق را با استفاده از lambda expression اینگونه تعریف می کنیم :

```
MyInterface1 i1 = (s) -> System.out.println(s);
i1.method1("abc");
```

4- Java Stream Api جهت کار با داده های عظیم

در نسخه 8 برای کار با collection یک API جدید معرفی گردیده است که با استفاده از آن می توان داده های collection را در دو حالت sequence و parallel پیمایش نمود .

برای مثال لیست صد تایی زیر را داریم :

```
List<Integer> myList = new ArrayList<>();
             for(int i=0; i<100; i++) myList.add(i);
```

در حالت پیمایش به روش sequence به صورت زیر عمل می کنیم :

```
Stream<Integer> sequentialStream = myList.stream();

Stream<Integer> highNumsSeq = sequentialStream.filter(p -> p > 90);

            highNumsSeq.forEach(p -> System.out.println("High Nums sequential="+p));
```

خروجی عبارتست از نمایش اعداد لیست به ترتیب کوچک به بزرگ .

حال در روشی دیگر که غالبا برای یافتن مقدار خاص در یک collection عظیم مفید می باشد از پیمایش parallel استفاده می کنیم .

```
Stream<Integer> parallelStream = myList.parallelStream();

Stream<Integer> highNums = parallelStream.filter(p -> p > 90);

            highNums.forEach(p -> System.out.println("High Nums parallel="+p));
```

5- Java Time API

پکیج معرفی شده java.time جهت کار با تاریخ و زمان در نسخه 8 معرفی گردیده است .

دو زیر پکیج java.time.format که جهت استفاده بهینه از date و time می باشد و java.time.zone که در مواردی که با timezone سروکار داریم کاربرد دارد .

6- Collection API پیشرفته

در نسخه 8 چند متد مفید به Collection API افزوده شده است که کار با آنرا راحت تر کرده است :

- متد `forEachRemaining(Consumer action)` :
این متد عمل مورد نیاز که به عنوان پارامتر به متد پاس شده است را برای المانهای باقیمانده که پردازش نشده اند اعمال می کند . این متد در کلاس Iterator اضافه شده است .

- متد `removeIf(Predicate filter)` :
همه المانهای یک Collection را که شرط filter برای آنها صادق باشد ، از collection حذف می کند . این متد در کلاس Collection اضافه شده است .

- متد `spliterator()` :
یک آبجکتی از کلاس Spliterator بر می گرداند . این متد در کلاس Collection اضافه شده است .

- متد های replaceAll ، compute و merge که مربوط به کلاس Map می باشد .

7- Concurrency API پیشرفته

متد های forEach(), compute()، forEachEntry()، forEachKey()، forEachValue()، merge() ، reduce()

و search() به کلاس ConcurrentHashMap افزوده شده است .

CompletableFuture یکی دیگر از کلاس هایی می باشد که در رابطه با اتمام و یا عدم اتمام یک task و روال پس از آن تصمیم می گیرد .

متد newWorkStealingPool در کلاس Executor که قابلیت را فراهم می سازد که تمامی پردازشگر های سیستم که در سطح موازی پردازشگر thread جاری قرار دارند و در حال حاضر بی کار هستند جهت اجرای thread اقدام کنند .

8- Java IO پیشرفته

- Files.list (Path dir) : یک لیستی از محتویات درون دایرکتوری با آدرس مشخص شده را بر میگرداند .
- Files.lines (Path dir) : همه خط های فایل با آدرس مشخص شده را می خواند .
- BufferedReader.lines () : یک استریم شامل المان های موجود در خطوط خوانده شده توسط bufferreader را بر میگرداند .