کلاس ها

کلاس ها

زبان های برنامه نویسی ترکیبی،زبان هایی هستند که قسمتی یا تمامی مفاهیم مختص زبانهای functional را استفاده می کنند و علاوه بر آن قابلیت برنامه نویسی به سایر روش ها را نیز در اختیار برنامه نویسی قرار می دهند

هم زبان C++ و هم زبان جاوا جزء زبان های ترکیبی هستند. اما در جاوا فرآیند ترکیب به اندازه زبان C++ مهم نیست.یک زبان ترکیبی روش های برنامه نویسی چند منظوره را نیز پشتیبانی می کند.

زبان C++ به دلیل سازگاری با زبان C یک زبان ترکیبی است.

زبان جاوا فرض می کند که شما تنها می خواهید که یک برنامه ی شی گرا داشته باشید.

شما به اشیا(object) از طریق ارجاعات (reference) دسترسی دارید.

- هر زبان برنامه نویسی به شیوه ی خود به المان های موجود در حافظه دسترسی پیدا می کند و آن ها را دستکاری می کند.
- گرچه به همه ی اشیا به عنوان یک آبجکت نگاه می کنیم اما چیزی که دستکاری می شود در حقیقت referencee به Object است.
- یک تلویزیون(object) و یک کنترل از راه دور(reference) را در نظر بگیرید.وقتی فردی می گوید "کانال تلویزیون را عوض کن" یا "صدای تلویزیون را کم کن" در واقع چیزی که در دسترس داریم و مورد دستکاری است reference است که به نوبه خود آبجکت را نیز تغییر می دهد.

شما به object ها از طریق reference دسترسی دارید.

1 Declare a reference variable

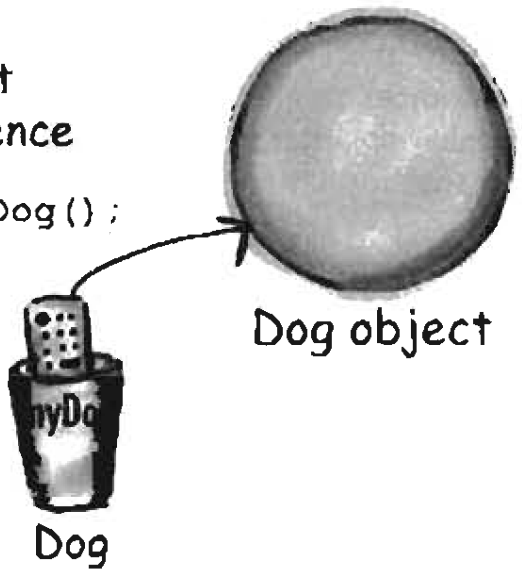
```
Dog myDog
```

2 Create an object

```
new Dog();
```

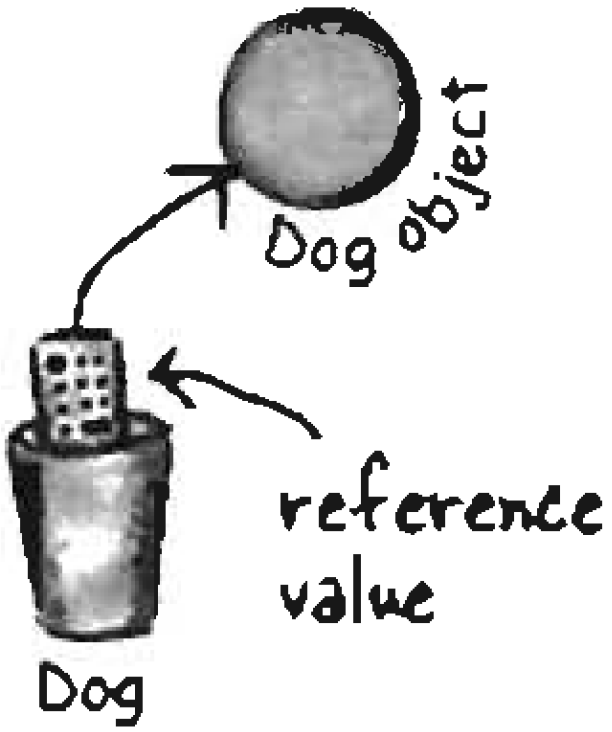
3 Link the object and the reference

```
Dog myDog = new Dog();
```



شما به object ها از طریق reference

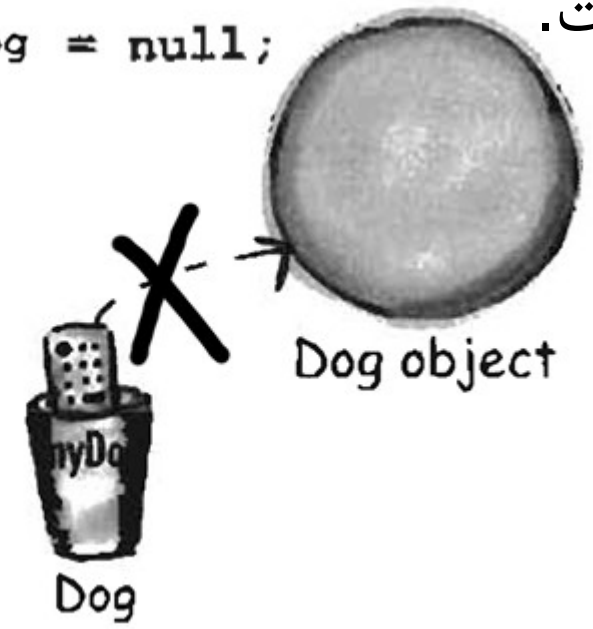
دسترسی دارید.



دسترسی به object ها از طریق reference

در مثال زیر شما تنها یک reference ایجاد نموده اید نه یک object.اگر در این مرحله بخواهید پیغامی را به myDog ارسال کنید با پیغام خطا مواجه می شود.چون myDog در واقع به چیزی متصل نیست.

```
Dog myDog = null;
```



ذخیره سازی در کجا صورت می گیرد

- **Register** (In CPU – کامپایلر از آن استفاده میکند)
- **The stack** (In RAM – متدها،متغیرها و متغیرهای محلی)
- **The heap** (In RAM – اشیاء)
- **Constant storage** حافظه بدون تغییر (در دیسک-در کد برنامه)
- **Non-RAM storage** (اشیای جاری-پایگاه داده)

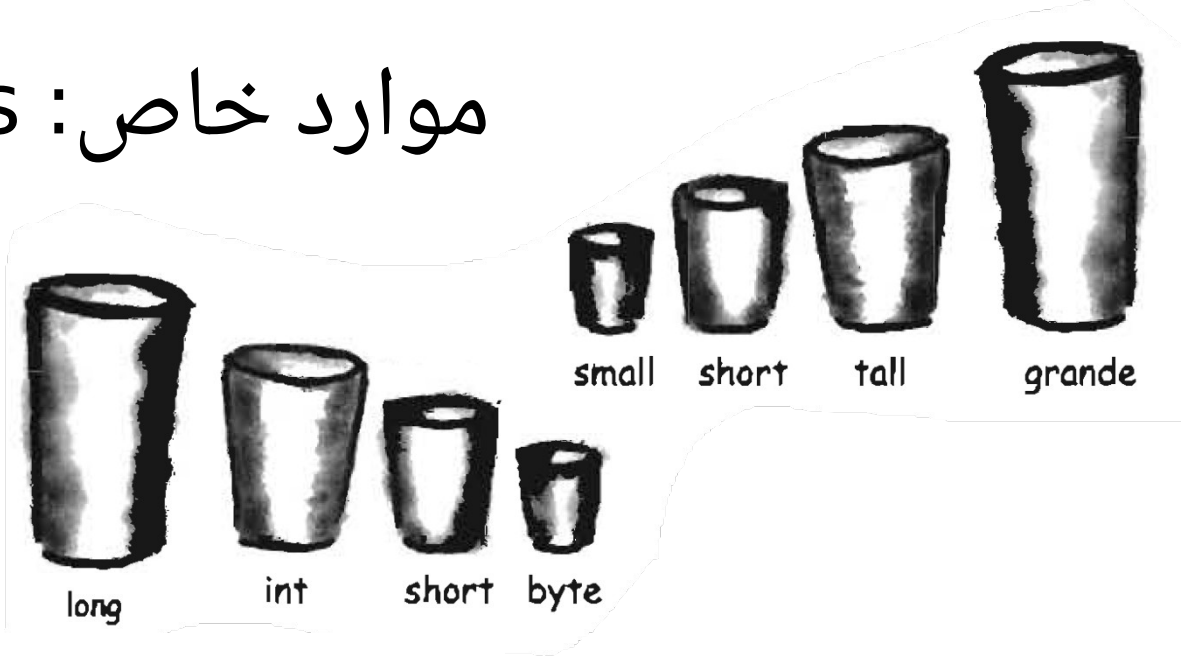
موارد خاص:متغیرهای اولیه primitive data types

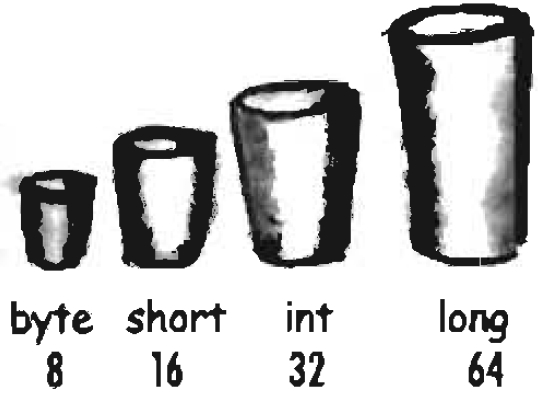
- گروهی از متغیرهای که اغلب در برنامه نویسی نیز زیاد استفاده می شوند،دارای رفتاری خاص هستند و به آن ها متغیرهای اولیه می گویند.
- دلیل این پردازش خاص نیاز نداشتن به ایجاد یک آبجکت با استفاده از new می باشد.
- به جای ایجاد متغیر با استفاده از new، متغیری خودکار ایجاد می شود که reference نیست.متغیر مستقیما مقدار را در خود نگه می دارد و در stack قرار داده می شود.

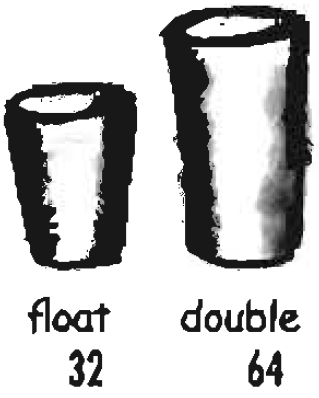
primitive data types :موارد خاص

Primitive type	Looks like	Size	Minimum	Maximum
boolean	**true** or **false**	–	–	–
char	‘A’	16 bits	Unicode 0	Unicode 2^{16}- 1
byte	(byte)42	8 bits	-128	+127
short	(short)42	16 bits	–32768	+32767
int	42	32 bits	–2147483648	+2147483647
long	42L	64 bits	–9223372036854775808	+9223372036854775807
float	42.0F	32 bits	–3.4 × 1038	+3.4 × 1038
double	42.0	64 bits	–1.8 × 10308	+1.8 × 10308
void		–	–	–

primitive data types :موارد خاص







موارد خاص:

primitive data types

```
int x = 24;
byte b = x;
//won't work!!
```

Wrapper type

Primitive type	Wrapper type
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
void	Void

```
char c = 'x';
Character ch = new Character(c);
```

Autoboxing

Primitive type	Wrapper type
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
void	Void

Autoboxing تبدیل اتوماتیکی است که کامپایلر جاوا بینprimitive data typeو wrapper typeآبجکت مربوطه انجام می دهد .

```
Character ch = 'x';
char c = ch;
```

اعداد با دقت بالا

جاوا شامل دو کلاس برای انجام محاسبات با دقت بالا می باشد:

BigInteger و **BigDecimal**

- **BigInteger**

پشتیبانی از اعداد صحیح با دقت دلخواه

- **BigDecimal**

برای اعداد اعشاری با دقت دلخواه.معمولا برای محاسبات دقیق پولی استفاده می شود.

می توان هر عملیاتی را که با int یا float انجام می دهیم با **BigDecimal** یا **BigInteger** نیز انجام دهیم تنها تفاوت فراخوانی متدها به جای استفاده از اپراتورها می باشد.

حوزه تعریف Scope

کد زیر اشتباه است.گرچه در زبان های C و C++ نوشتن کد به شیوه زیر صحیح است

```
{
  int x = 12;
  {
    int x = 96; // Illegal
  }
}
```

Object Scope حوزه ی تعریف اشیا

```
{
  String s = new String("a string");
} // End of scope
```

reference s گرچه آبجکت در پایان ح

string ای که sبه آن اشاره می کند همچنان حافظه را اشغال نموده ولی در اینجا راهی برای دسترسی به آبجکت در پایان حوزه تعریف وجود ندارد چرا که تنهاreference.آن خارج از حوزه تعریف است

ایجاد نوع داده جدید:class

کلید واژه class به این معنی است که "من می خوام به شما بگویم که یک نوع جدید از شی به چه صورت است":

```
class ATypeName{
    /* Class body goes here */
}
```

می توان با استفاده از new شی ای از نوع این کلاس ایجاد کرد:

```
ATypeName a = new ATypeName();
```

فیلد و متدها

می توان دو نوع المان را در کلاس قرار داد:field ها (اعضای داده) و methodها(اعضای ساختار)

یک فیلد،آبجکتی است که می تواند از هر نوع داده ای باشد و می توان از طریق reference با آن صحبت نمود و یا یک primitive data type است.

```
class DataOnly {
  int i;
  double d;
  boolean b;
}
```

فیلدها

می توان به فیلدها مقادیری را اختصاص داد اما ابتدا باید بدانید که چگونه به عضوی از یک آبجکت اشاره کرد.این کار از طریق نوشتن نام object Referenceسپس گذاشتن نقطه و بعد نوشتن نام عضو موجود در آبجکت انجام می گیرد.

```
//objectReference.member
data.i = 47;
data.d = 1.1;
data.b = false;
```

مقادیر پیش فرض برای اعضای اولیه

مقادیر پیش فرض مقادیری هستند که جاوا هنگامی که متغیر به عنوان عضوی از کلاس استفاده می شود آن را تعیین می کند

Primitive type	Default
boolean	false
char	‘\u0000’ (null)
byte	(byte)0
short	(short)0
int	0
long	0L
float	0.0f
double	0.0d

متدها،آرگومان ها و مقادیر بازگشتی

اصطلاح متداولی که در جاوا استفاده می شود method می باشد و به معنی روش انجام کار می باشد.

```
ReturnType methodName( /* Argument list */) {
 /* Method body */
}
```

متدها در جاوا تنها به عنوان عضوی از کلاس ایجاد می شوند.یک متد می تواند برای یک آبجکت فراخوانی شود.

```
objectName.methodName(arg1, arg2, arg3);
```

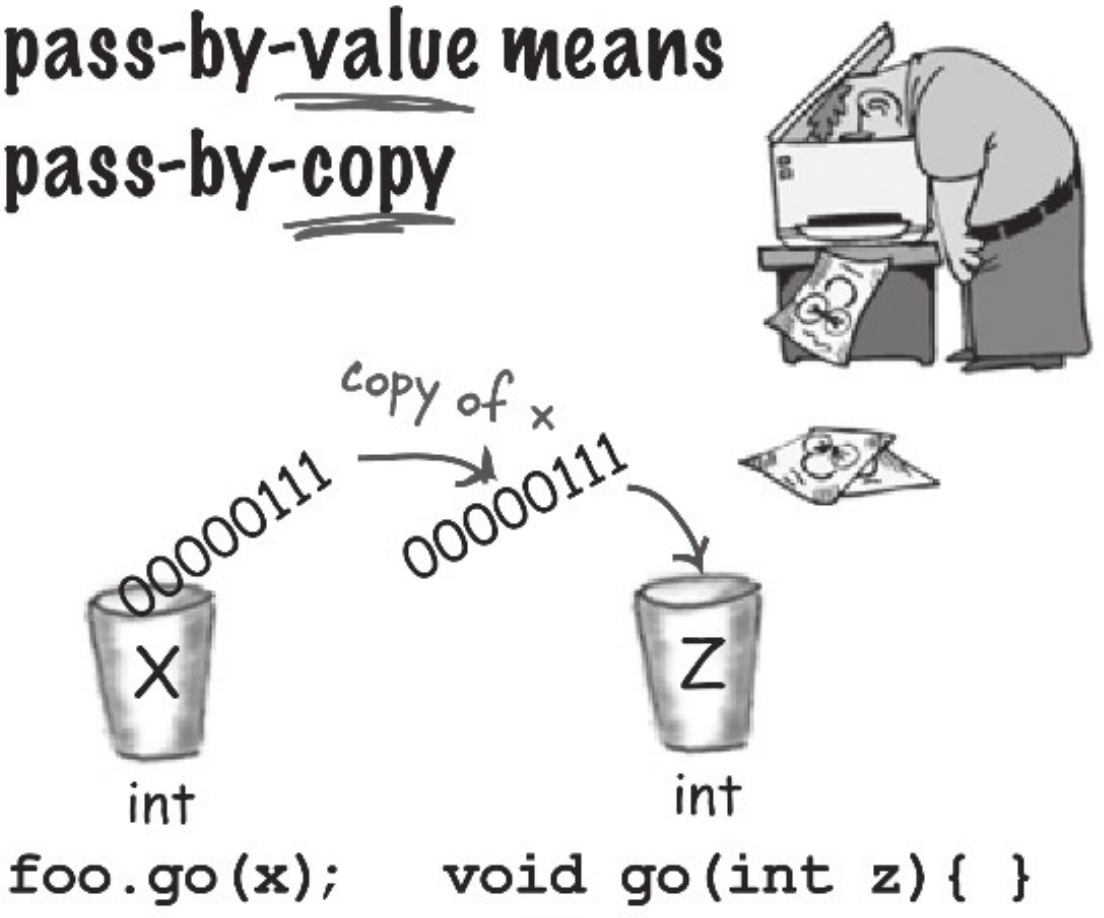
لیست آرگومان ها

لیست آرگومان های متد مشخص می کند که چه اطلاعاتی به متد ارسال شود و در واقع به صورت reference ارسال می گردد. در نوع داده های اولیه جاوا مقداری را به متد ارسال می کند.

```
int storage(String s) {
   return s.length() * 2;
}
```

برای)ارسال مقدار

primitive data types)



بازگشت مقادیر

می توان هر نوع داده ای را از متد برگرداند. در صورتی که بخواهیم نشان دهیم که متد مقداری را برنمی گرداند از **void** استفاده می کنیم:

```
boolean flag() { return true; }

double naturalLogBase() { return 2.718; }

void nothing() { return; }

void nothing2() {}
```

Name visibility

برای تولید یک نام بدون ابهام برای کتابخانه،سازندگان جاوا از شما می خواهند که در نام از دامنه اینترنتی تان استفاده کنید تا یکتا بودن آن تضمین شود.به عنوان مثال اگر www.Demisco.ir نام دامنه اینترنتی باشد،کتابخانه ابزار من می تواند به صورت com.demisco.utility باشد.بعد از مشخص نمودن دامنه اینترنتی واژه ای که بعد از نقطه می آید زیرشاخه موجود را مشخص می کند.

در فایل کدهای جاوا،بسته ای که در آن کلاس هایی وجود دارد و یا فایل های آن به صورت کلاس هستند با عبارت package در ابتدای نام آن مشخص می شوند

```
package com.demisco.utility;
```

استفاده از کامپوننت های دیگر

هرگاه بخواهید از یک کلاس از پیش تعریف شده استفاده کنید،کامپایلر باید بداند که چگونه آن را پیدا کند.

برای حل این مشکل باید تمام ابهامات موجود را از بین برد.این کار از طریق مشخص کردن دقیق کلاس برای کامپایلر جاوا با استفاده از کلید واژه Import انجام می گیرد

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

در صورتی که بخواهید چند کلاس از آن ها را با هم و بدون نیاز به تعریف جدا مشخص نمایید با استفاده از * به عنوان wild card این کار را انجام دهید.

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

کلید واژه Static

دو حالت وجود دارد که در آن ها ایجاد یک شی جدید برای استفاده از اعضای آن کافی نیست:

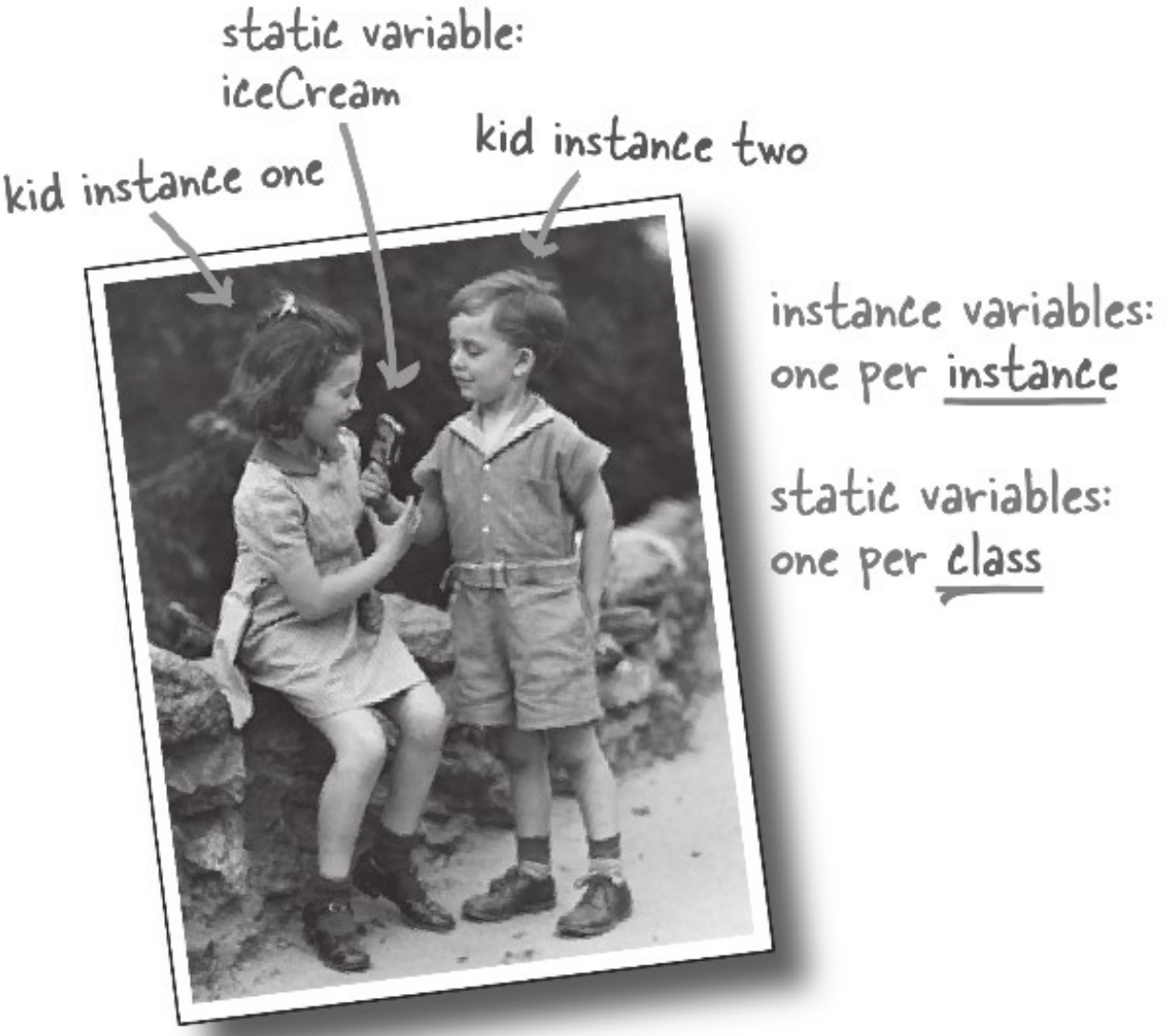
- در صورتی که بخواهید مقداری حافظه جدا برای فیلدی خاص داشته باشید.
- در صورتی که بخواهید متدی داشته باشید که با شی خاصی از اعضای کلاس در ارتباط نباشد.

با استفاده از کلید واژه Static می توان به دو موقعیت بالا دست یافت.

Staticبه این معنی است که فیلد یا متدی خاص با شی خاصی از کلاس در ارتباط نیست .

کلید واژه

Static



کلید واژه Static

```
class StaticTest {   static int i = 47; }
StaticTest st1 = new StaticTest();
StaticTest.i++;

class Incrementable {
  static void increment() {
      StaticTest.i++;
    }
}
```

main متد

```
java.util.*;
public class HelloDate {
public static void main(String[] args) {
    System.out.println("Hello, it’s: ");
    System.out.println(new Date());
  }
```

متد main

کلاس java.lang.System به صورت خودکار import می شود چون **java.lang** به صورت ضمنی در هر فایل کد جاوا گنجانده شده است.

در آینده خواهید دید که کلاس System دارای چندین فیلد است و اگر بخواهید آن را انتخاب کنید به یک شی از نوع **static PrintStream** می رسید.

println()به آنچه را به عنوان ورودی می دهم در یک صفحه کنسول نمایش بده و در خط جدید عملیات را پایان بده "

کلید واژه Public به این معنی است که متد می تواند در محیط بیرونی نیز در دسترس باشد.

args در این زبان استفاده نمی شود اما کامپایلر جاوا اصرار دارد که وجود دارد چرا که در نگهداری آرگومان ها در خطوط کامنت استفاده می شود

کامنت ها و مستندات

دو نوع کامنت در جاوا وجود دارد.اولین نوع همان شیوه کامنت گذاری سنتی زبان C می باشد که در C++ نیز به ارث برده شده است.کامنت ها با علامت /* آغاز می شوند و در هر چند خط که لازم است ادامه می یابند و در نهایت با علامت */ پایان می یابند.

```
/* This is a comment that continues
 *  across lines
 */

// This is a one-line comment
```

کامنت ها و مستندات

- تمام دستورات Javadoc با /** در کامنت ، شروع و با */ پایان می یابند.

دو راه اولیه برای استفاده از Javadoc :

- قرار دادن HTML و یا استفاده از تگ های توضیحات
- تگ های توضیحات دستوراتی هستند که با @ آغاز می شوند و در ابتدای خط کامنت قرار می گیرند.

سه نوع مستندسازی کامنت وجود دارد که به المانی که کامنت مد نظر در ابتدای آن قرار دارد بستگی دارد:کلاس،فیلد و یا متد

چند مثال از استفاده از تگ ها

@see
@version
@author
@since
@param
@return
@throws
@deprecated

شیوه ی کدنویسی

شیوه کدنویسی ای که در زبان برنامه نویسی جاوا مرسوم است،بزرگ نوشتن اولین حرف نام کلاس می باشد.همچنین اولین حرف هر لغت در نام موجود نیز به صورت بزرگ نوشته می شود:

```
class AllTheColorsOfTheRainbow { // ...
```

به این شیوه camel-casing می گویند.

به جز کلاس برای موارد دیگر اولین حرف به صورت کوچک نوشته می شود:

```
int anIntegerRepresentingColors;
void changeTheHueOfTheColor(int newHue){}
```