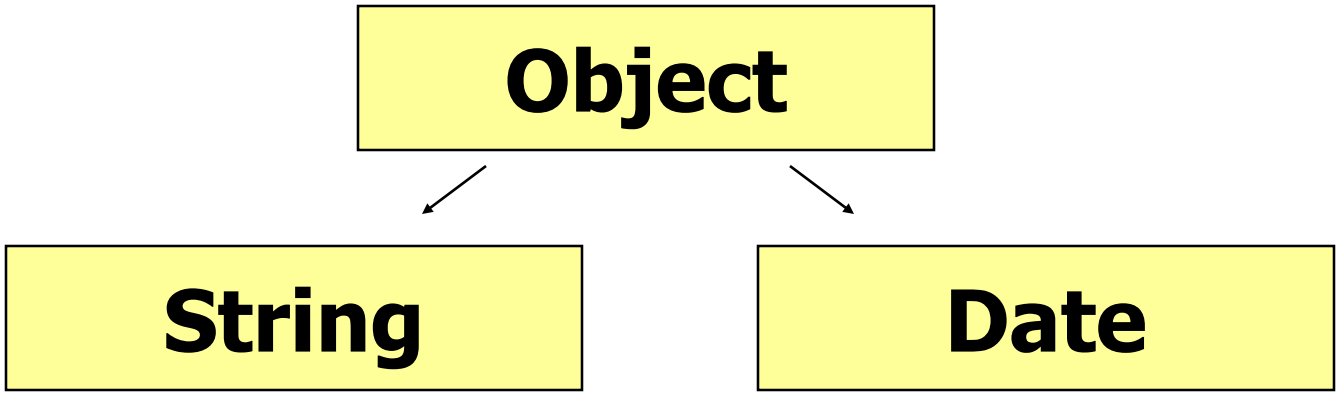
A+ Computer Science

Writing Classes

Object Class

Object Class

In Java, all classes are sub classes of class Object. This adds greater flexibility when writing programs in Java.



Object Class

```
public class Monster extends Object
{
   public void print( )
   {
      out.println("Monster");
   }
}
```

Object Class

Because all classes are sub classes of Object, all classes start with the same methods.

.equals( )
.toString( )
. . . . . and more

Overriding

Often, new classes override at least .equals() and .toString().

.equals( )
.toString( )
. . . . . and more

equals() method

The equals() method is used to determine if two objects have the same contents / values.

```
String one = "comp";
String two = "sci";
out.println(one.equals(two));
```

equals() method

```
class Monster
{
  private int height;

  //methods

  public boolean equals(Object obj){
    Monster other = (Monster)obj;
    if(getHeight()==other.getHeight())
      return true;
    return false;
  }

  //methods
}

//test code in the main
Monster one = new Monster(33);
Monster two = new Monster(12);
out.println(one.equals(two));
```

OUTPUT

false

equals.java

toString.java

The Monster Class

Monster Class

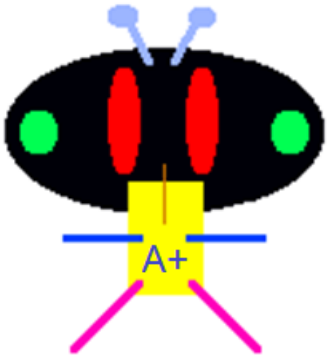
Monster() - constructors

setX( ) - mutators

getX( ) - accessors

toString( ) - accessor

Monster



Monster

```
class Monster
{
  //instance vars / data fields

  public Monster(){
    //code
  }

  public void setX( params ){
    //code
  }

  public int getX(){
    //code
  }

  public String toString() {
    //code
  }
}
```

constructor

mutator

accessor

accessor

Overloading

Overloading occurs when you have more than one method or constructor with the same name. Each method or constructor must have a different parameter list.

# of parameters && data types matter

```
class Monster{
  private int height;              //default assinged to 0
  private double weight;           //default assinged to 0
  private int age;                 //default assinged to 0

  public Monster(){
    height=0;
    weight=0.0;
    age = 0;
  }

  public Monster(int ht){
    height=ht;
    weight=0.0;
    age = 0;
  }

  public Monster(int ht, double wt){
    height=ht;
    weight=wt;
    age = 0;
  }
}
```

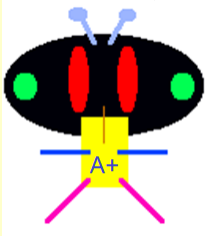
Overloading

Overloading

Monster m = new Monster();

m

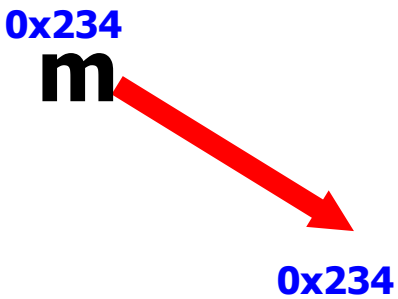
MONSTER
properties
– height – 0 weight – 0.0 age - 0
methods



m is a reference variable that refers to a Monster object.

Overloading

Monster m = new Monster(23);

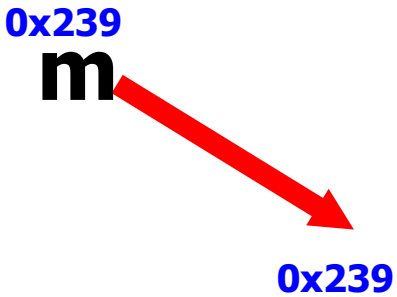


MONSTER
properties
– height – 23 weight – 0.0 age - 0
methods

m is a reference variable that refers to a Monster object.

Overloading

Monster m = new Monster(23, 45);

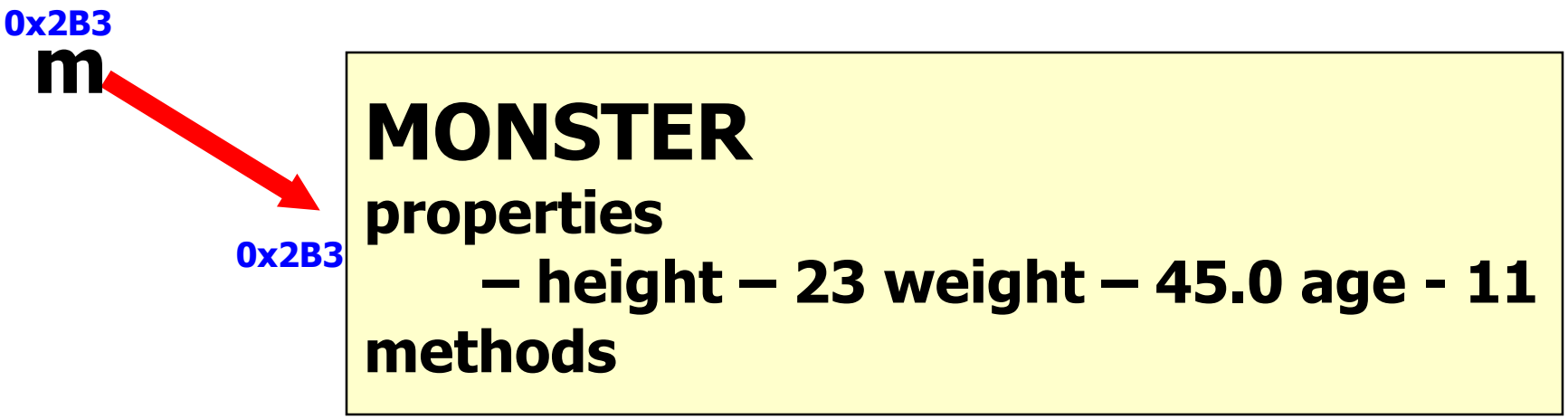


MONSTER
properties
– height – 23 weight – 45.0 age - 0
methods

m is a reference variable that refers to a Monster object.

Overloading

Monster m = new Monster(23, 45, 11);



m is a reference variable that refers to a Monster object.

overload.java

can you see this

this

this – refers to the object/class you are working in

this.toString( ); calls the toString of this class

this.x = 1524;

this( ); calls a constructor of this class

this

```
class Monster
{
  private String name;

  public Monster() {
    this("Monster");
  }
  public Monster( String name ) {
    this.name = name;
  }
  public String toString() {
    return this.name + "\n";
  }
}
```

calls Monster(name)

this.java

static

Static

Static is a reserved word use to designate something that belongs to a class.

Static variables and methods exist even if no object of that class has been instantiated.

Static

Static means one!

All objects will share the same static variables and methods.

Static

```
out.println(Math.floor(3.254));
out.println(Math.ceil(2.45));
out.println(Math.pow(2,7));
out.println(Math.abs(-9));
out.println(Math.sqrt(256));
```

OUTPUT

```
3.0
3.0
128.0
9
16.0
```

All of the Math methods are static. All of the Math methods can be used with instantiating an object. There is one copy of each of the Math methods.

Static

```
class Monster
{
   private String name;
   private static int count = 0;

   public Monster() {
     name ="";
     count++;
   }
   public Monster( String name ) {
     this.name = name;
     count++;
   }
}
```

all Monster share count

Static

```
class Monster
{
   private String name;
   private static int count = 0;

   //other stuff not shown

   public static int getCount( )
   {
     return count;
   }
}
```

all Monster share getCount()

static.java

Variable Scope

Scope

```
{
   int fun = 99;
}
```

Any variable defined inside of braces, only exists within those braces.

That variable has a scope limited to those braces.

Declaring vs. Assigning

```
int num;
```

declartion only

```
int num = 99;
```

declaration and assignment

```
num = 56;
```

assignment only

Local Variables

When you need only one method to have access to a variable, you should make that variable a local variable.

The scope of a local variable is limited to the method where it is defined.

Local Variables

```
public class LocalVars
{
   private int fun;          //instance variable

   public void change()  {
    int fun = 99;            //local variable
   }

   public void print()  {
    System.out.println(fun);
   }

   public static void main(String args[])
   {
     LocalVars test = new LocalVars();
     test.change();
     test.print();
   }
}
```

OUTPUT

0

LocalVars

fun
0

change

fun = 99

print

localvars.java

Parameters Expanded

Parameters

Formal Parameters

```
void fillRect(int x, int y, int width, int height)
```

Actual Parameters

```
window.fillRect( 10, 50, 30, 70 );
```

Parameters

void fillRect(int x, int y, int width, int height)

window.fillRect(10, 50, 30, 70);

The call to fillRect would draw a rectangle at position 10,50 with a width of 30 and a height of 70.

Passing by Value

Java passes all parameters by VALUE.

Primitives are passed as values by VALUE.

References are passed as addresses by VALUE.

Passing by Value

Passing by value simply means that a copy of the original is being sent to the method.

Passing by Value

If you are sending in a primitive, then a copy of that primitive is sent.

If you are sending in a reference or memory address, then a copy of that memory address is sent.

Passing by Value

```
public static void go( int x )
{
  x = 7;
  System.out.println( x );
}

//test code  -  runner code
int one=5;
System.out.println( one );
go( one );
System.out.println( one );
```

OUTPUT

```
5
7
5
```

Passing by Value

```
public static void go( Integer x )
{
  x = 7;
  System.out.println( x );
}

//test code  -  runner code
Integer one=5;
System.out.println( one );
go( one );
System.out.println( one );
```

OUTPUT

```
5
7
5
```

Passing by Value

```
public static void go( String s )
{
   s = s.substring( 0, 2 );
   System.out.println( s );
}

//test code  -  runner code
String one = "aplus";
System.out.println( one );
go( one );
System.out.println( one );
```

OUTPUT

```
aplus
ap
aplus
```

Passing by Value

```
public static void swap( int x, int y)
{
   int t = x;
   x = y;
   y = t;
}
```

This attempted swap has local effect, but does not affect the original variables. Copies of the original variable values were passed to method swap.

passbyvalueone.java

Passing by Value

```
class A{
   private String x;
   public A( String val ){
     x = val;
   }
   public void change( ){
     x = "aplus";
   }
   public String toString(){
      return x;
   }
}

class B{
   public void mystery(A x) {
      x.change();
   }
}

//test code in the main in another class
B test = new B();
A one = new A("comp");
System.out.println( one );
test.mystery( one );
System.out.println( one );
```

OUTPUT

comp
aplus

Passing by Value

```
class A{
  private String s;
  public A( String val ){
    s = val;
  }
  public void change( ){
    s = "aplus";
  }
  public String toString(){
    return s;
  }
}

class B{
  public void mystery(A x, A y) {
    x.change();
    y = x;
  }
}

//test code in the main in another class
B test = new B();
A one = new A("comp");
A two = new A("sci");
System.out.println(one + " " + two);
test.mystery(one,two);
System.out.println(one + " " + two);
```

OUTPUT

```
comp sci
aplus sci
```

passbyvaluetwo.java

Work on Programs!

Crank Some Code!

A+ Computer Science

WRITING CLASSES