Object Oriented Programming

Topics To Be Covered Today

Method

- Overloading

Static Class Members

Method Overloading

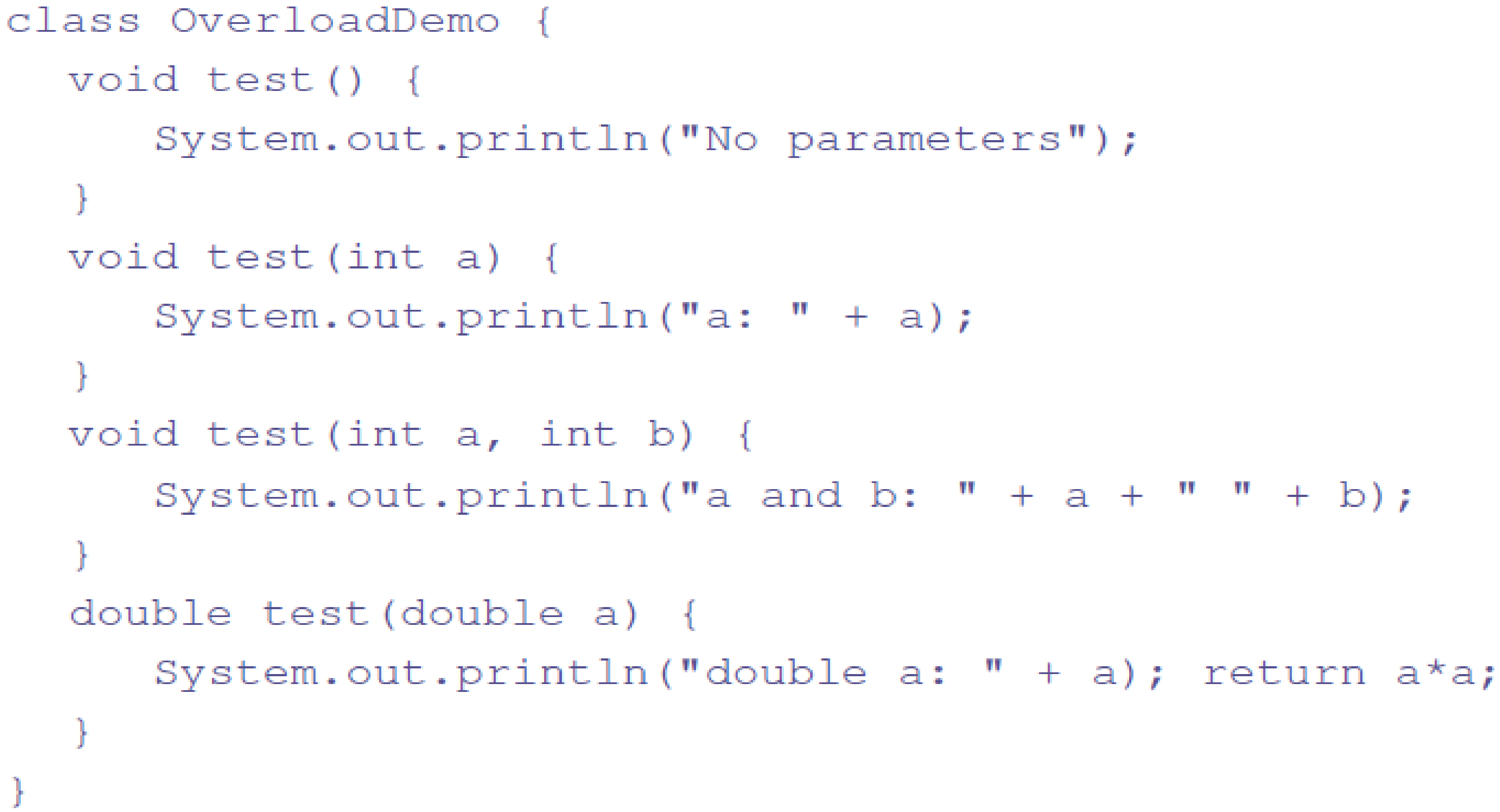
It is legal for a class to have two or more methods with the same name.

However, Java has to be able to uniquely associate the invocation of a method with its definition relying on the number and types of arguments.

Therefore the same-named methods must be distinguished:

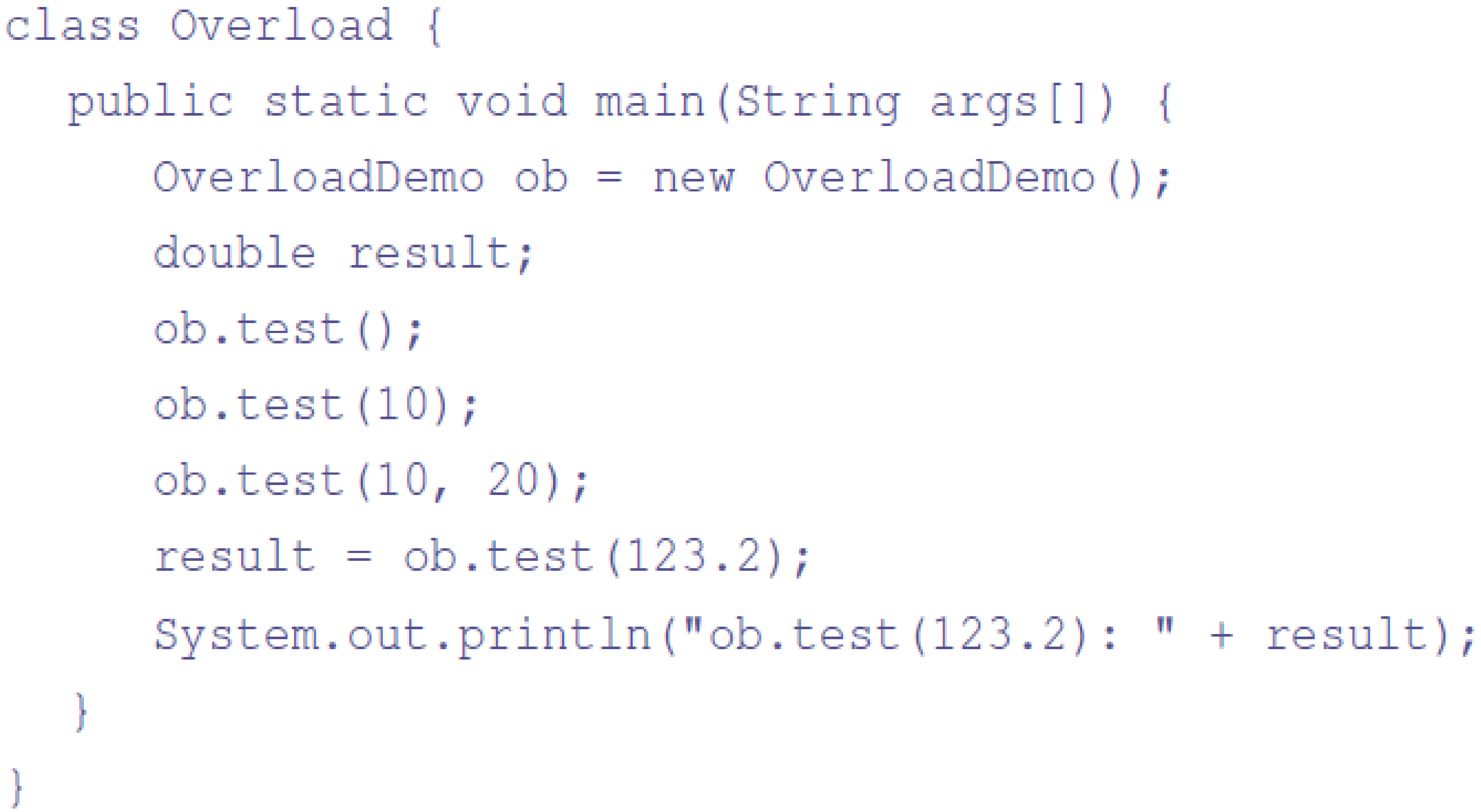
- by the number of arguments, or
- by the types of arguments

Example: Method Overloading



```
class OverloadDemo {
  void test() {
     System.out.println("No parameters");
  }
  void test(int a) {
     System.out.println("a: " + a);
  }
  void test(int a, int b) {
     System.out.println("a and b: " + a + " " + b);
  }
  double test(double a) {
     System.out.println("double a: " + a); return a*a;
  }
}
```

Example: Method Overloading

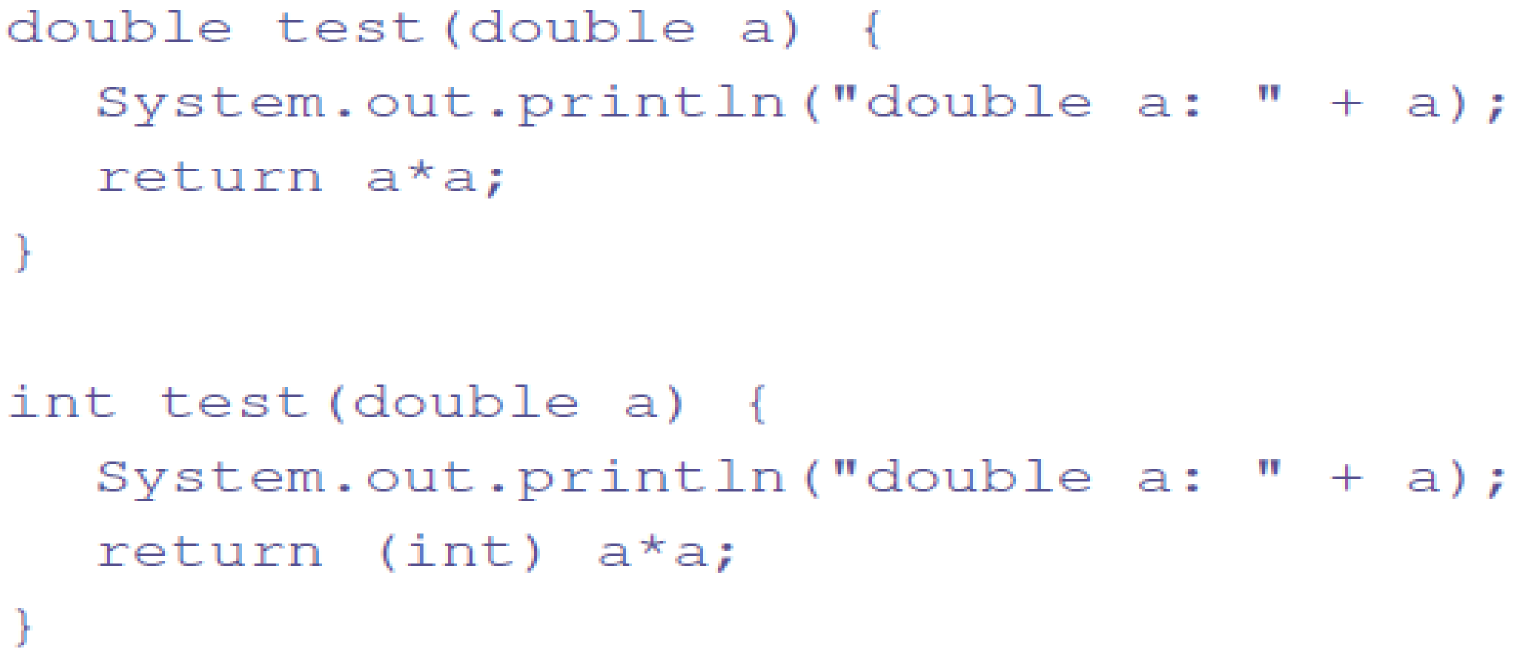


```
class Overload {
  public static void main(String args[]) {
     OverloadDemo ob = new OverloadDemo();
     double result;
     ob.test();
     ob.test(10);
     ob.test(10, 20);
     result = ob.test(123.2);
     System.out.println("ob.test(123.2): " + result);
  }
}
```

Different Return Types

Different return types are insufficient.

The following will not compile:



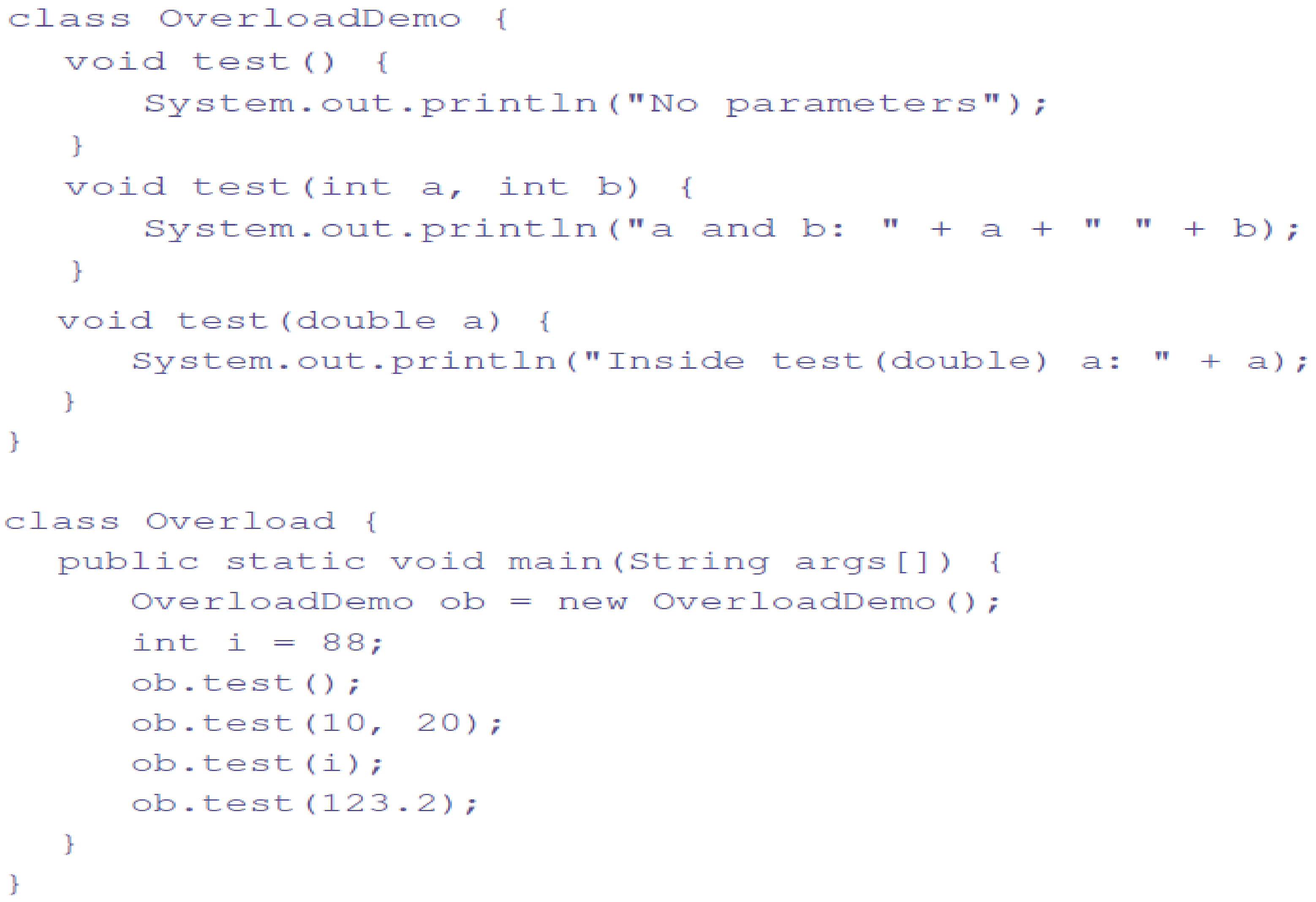
```
double test(double a) {
   System.out.println("double a: " + a);
   return a*a;
}

int test(double a) {
   System.out.println("double a: " + a);
   return (int) a*a;
}
```

Overloading and Conversion

When an overloaded method is called, Java looks for a match between the arguments used to call the method and the method’s parameters.

When no exact match can be found, Java’s automatic type conversion can aid overload resolution:

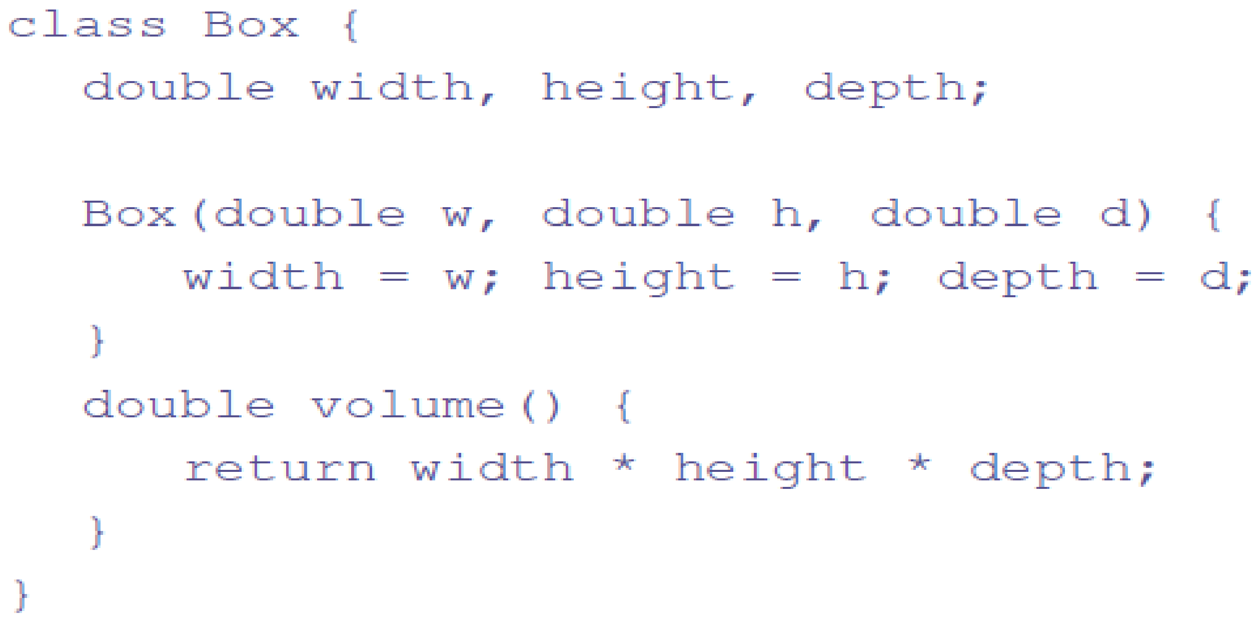
Overloading and Conversion



```
class OverloadDemo {
  void test() {
     System.out.println("No parameters");
  }
  void test(int a, int b) {
     System.out.println("a and b: " + a + " " + b);
  }
  void test(double a) {
     System.out.println("Inside test(double) a: " + a);
  }
}

class Overload {
  public static void main(String args[]) {
     OverloadDemo ob = new OverloadDemo();
     int i = 88;
     ob.test();
     ob.test(10, 20);
     ob.test(i);
     ob.test(123.2);
  }
}
```

Constructor Overloading

Why overload constructors? Consider this:

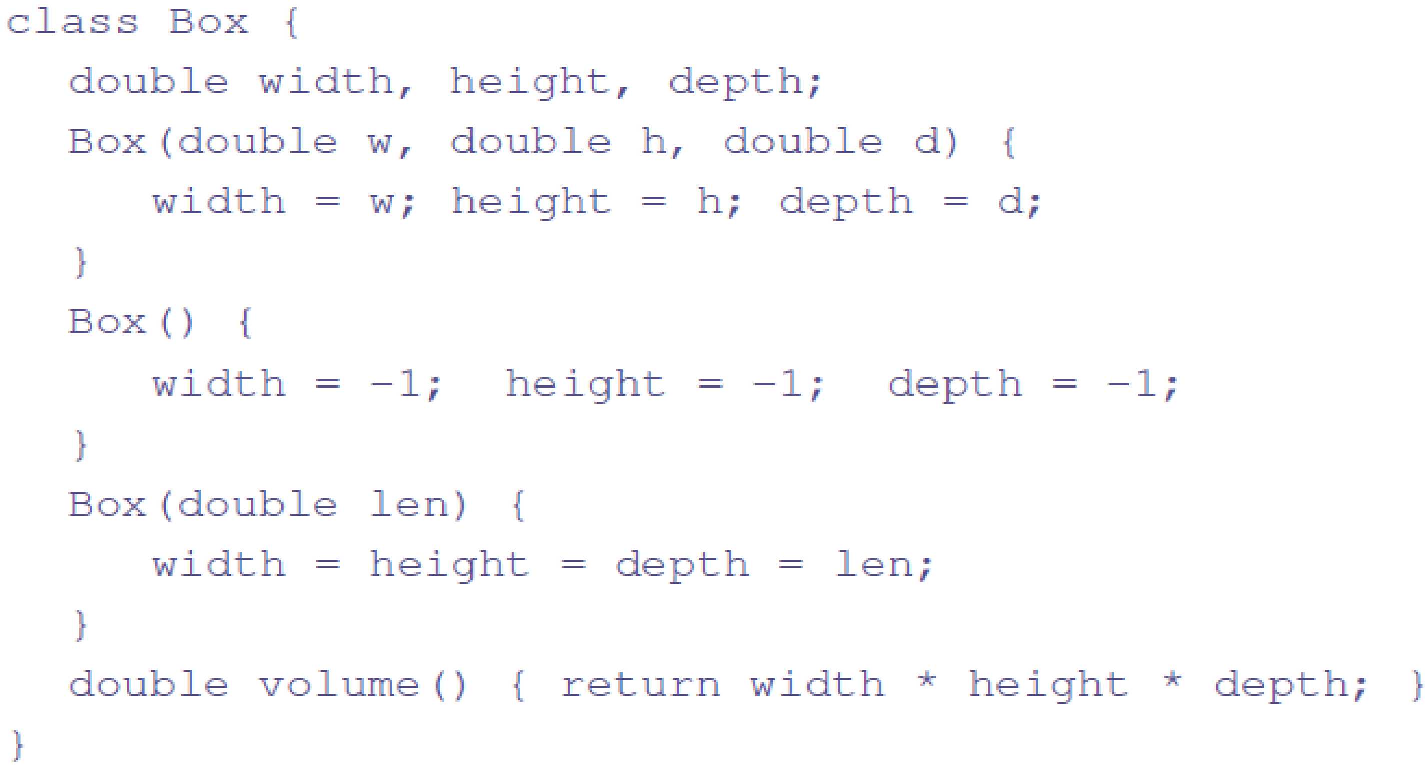


```
class Box {
   double width, height, depth;

   Box(double w, double h, double d) {
      width = w; height = h; depth = d;
   }
   double volume() {
      return width * height * depth;
   }
}
```

All Box objects can be created in one way: passing all three dimensions.

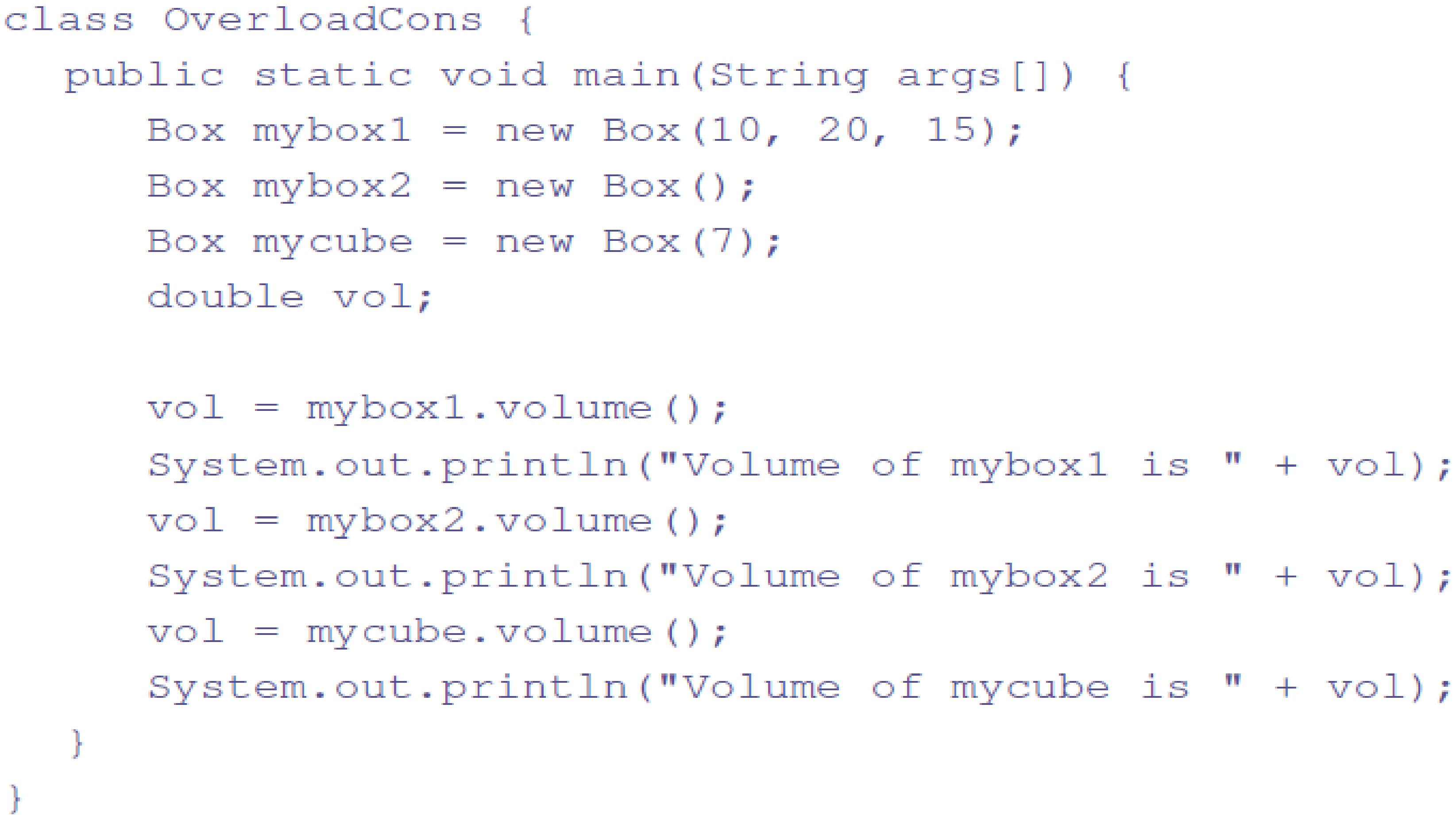
Example: Overloading

Three constructors: 3-parameter, 1-parameter, parameter-less.



```
class Box {
  double width, height, depth;
  Box(double w, double h, double d) {
     width = w; height = h; depth = d;
  }
  Box() {
     width = -1;  height = -1;  depth = -1;
  }
  Box(double len) {
     width = height = depth = len;
  }
  double volume() { return width * height * depth; }
}
```

Example: Overloading



```
class OverloadCons {
  public static void main(String args[]) {
     Box mybox1 = new Box(10, 20, 15);
     Box mybox2 = new Box();
     Box mycube = new Box(7);
     double vol;

     vol = mybox1.volume();
     System.out.println("Volume of mybox1 is " + vol);
     vol = mybox2.volume();
     System.out.println("Volume of mybox2 is " + vol);
     vol = mycube.volume();
     System.out.println("Volume of mycube is " + vol);
  }
}
```

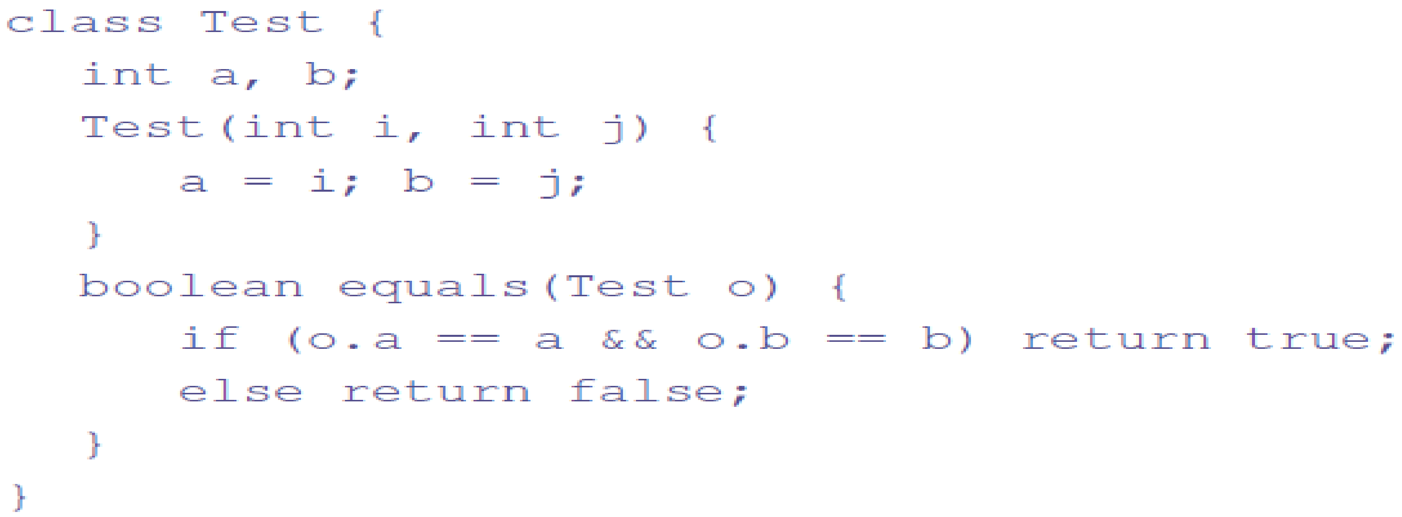
Object Argument

So far, all method received arguments of simple types.

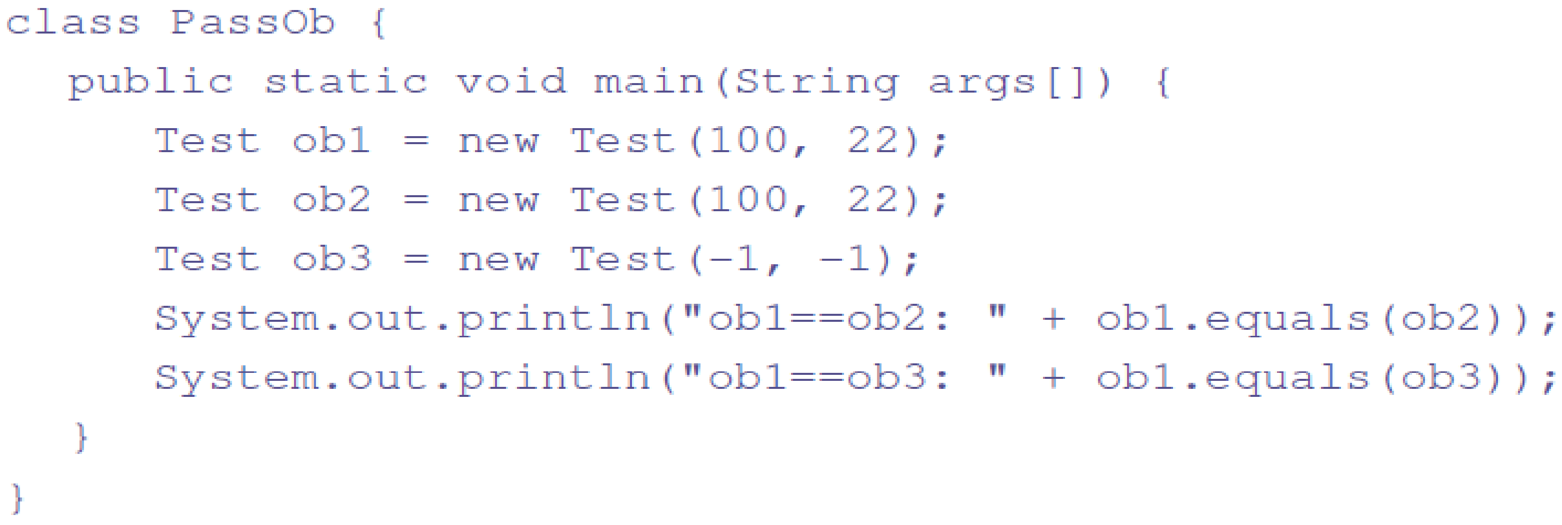
They may also receive an object as an argument.

In the next slide, lets see a method to check if a parameter object is equal to the invoking object.

Object Argument



```
class Test {
  int a, b;
  Test(int i, int j) {
    a = i; b = j;
  }
  boolean equals(Test o) {
    if (o.a == a && o.b == b) return true;
    else return false;
  }
}
```

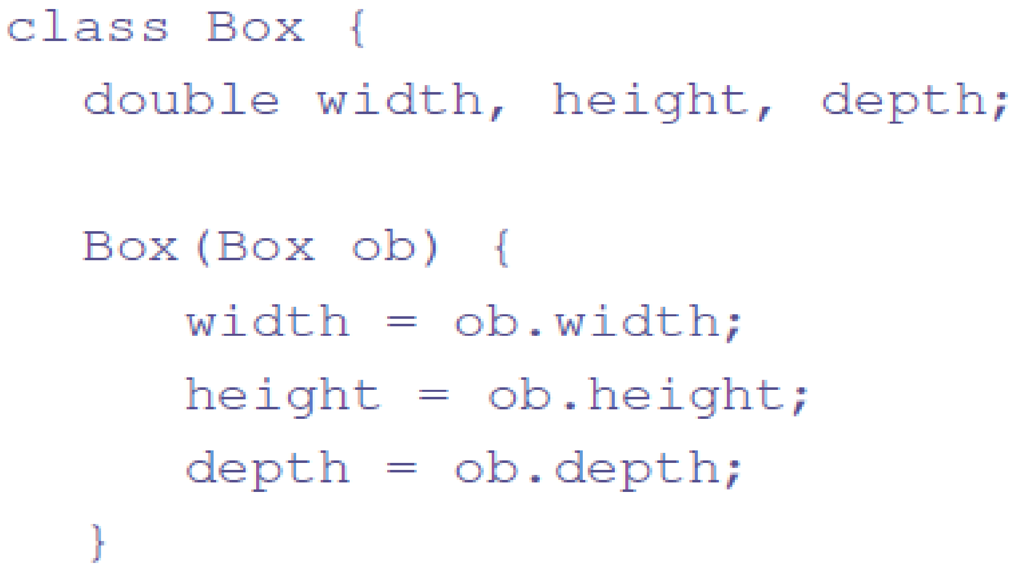


```
class PassOb {
  public static void main(String args[]) {
    Test ob1 = new Test(100, 22);
    Test ob2 = new Test(100, 22);
    Test ob3 = new Test(-1, -1);
    System.out.println("ob1==ob2: " + ob1.equals(ob2));
    System.out.println("ob1==ob3: " + ob1.equals(ob3));
  }
}
```

Passing object to Constructor

A special case of object-passing is passing an object to the constructor.

This is to initialize one object with another object:



```
class Box {
  double width, height, depth;

  Box(Box ob) {
      width = ob.width;
      height = ob.height;
      depth = ob.depth;
  }
  Box(double w, double h, double d) {
     width = w;
     height = h;
     depth = d;
  }

  double volume() {
     return width * height * depth;
  }
}
class OverloadCons2 {
  public static void main(String args[]) {
     Box mybox1 = new Box(10, 20, 15);
     Box mybox2 = new Box(mybox1);
     double vol;

     vol = mybox1.volume();
     System.out.println("Volume of mybox1 is " + vol);

     vol = mybox2.volume();
     System.out.println("Volume of mybox2 is " + vol);
  }
}
```

Argument Passing

Two types of variables:

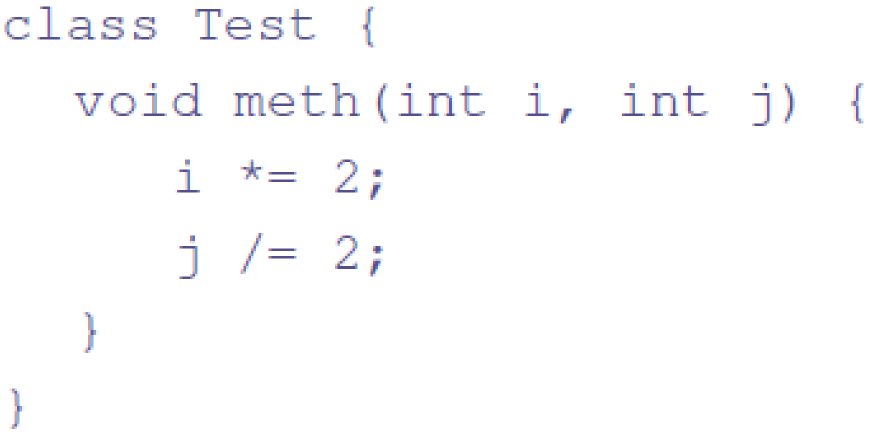
- simple types
- class types

Two corresponding ways of how the arguments are passed to methods:

- by value a method receives a copy of the original value;parameters of simple types
- by reference a method receives the memory address of the original value, not the value itself; parameters of class types

Simple Type Argument Passing

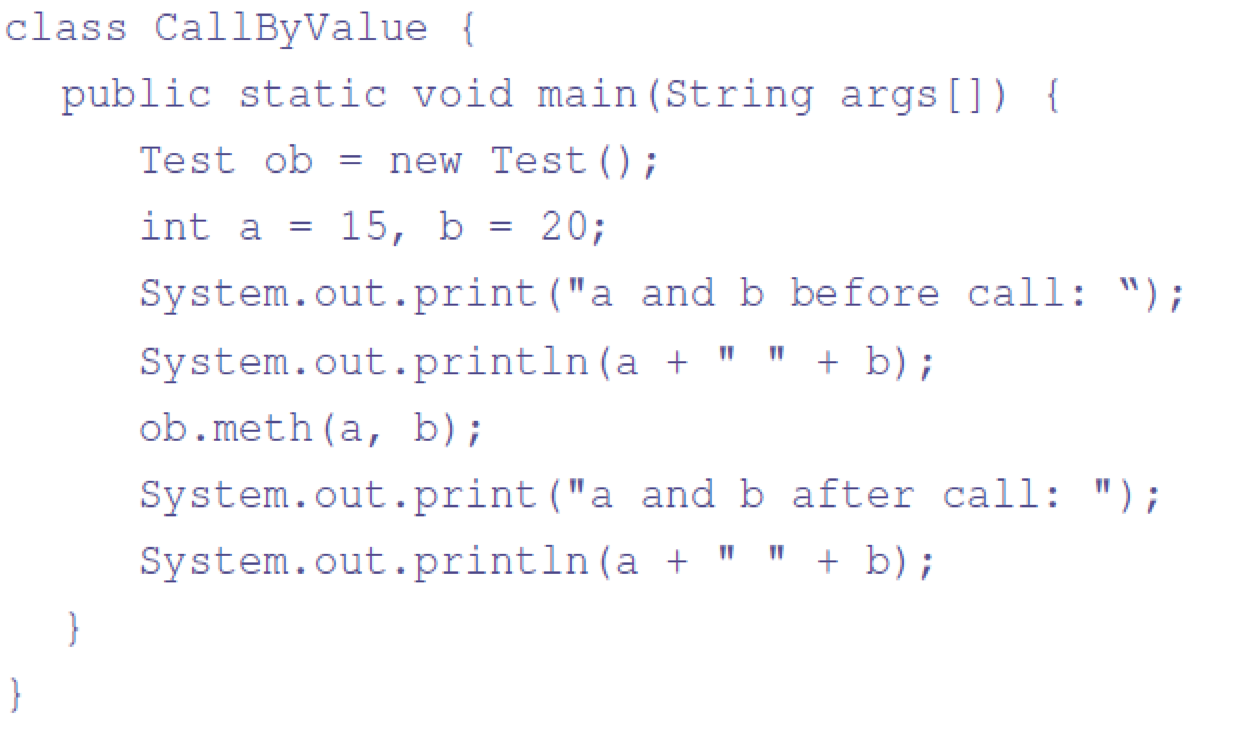
Passing arguments of simple types takes place by value:



```
class Test {
  void meth(int i, int j) {
     i *= 2;
     j /= 2;
  }
}
```

Simple Type Argument Passing

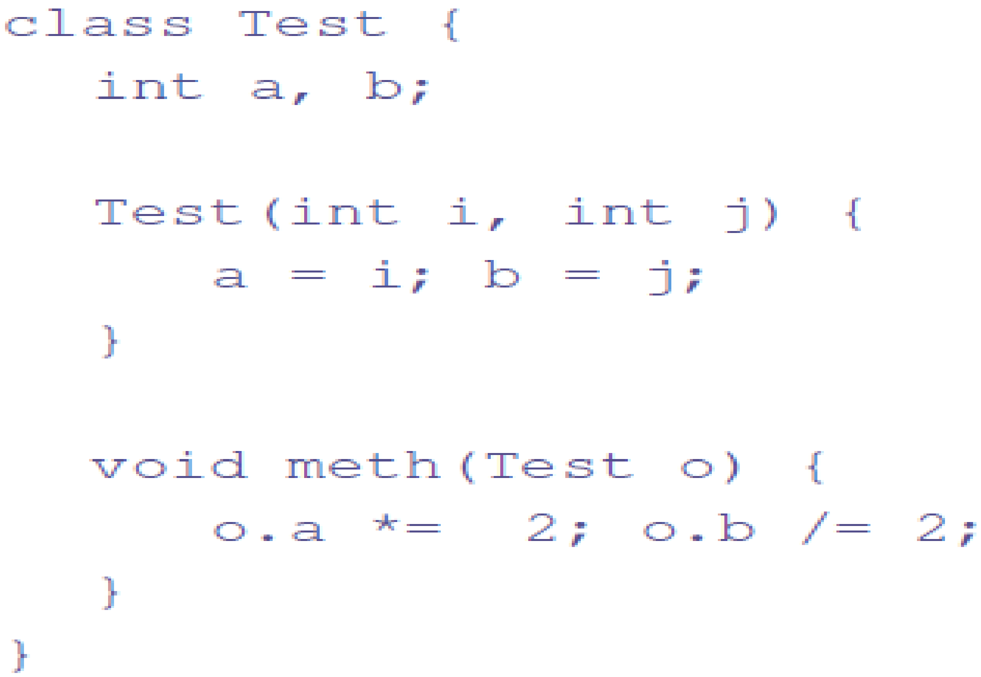
With by-value argument-passing what occurs to the parameter that receives the argument has no effect outside the method:



```
class CallByValue {
  public static void main(String args[]) {
     Test ob = new Test();
     int a = 15, b = 20;
     System.out.print("a and b before call: “);
     System.out.println(a + " " + b);
     ob.meth(a, b);
     System.out.print("a and b after call: ");
     System.out.println(a + " " + b);
  }
}
```

Class Type Argument Passing

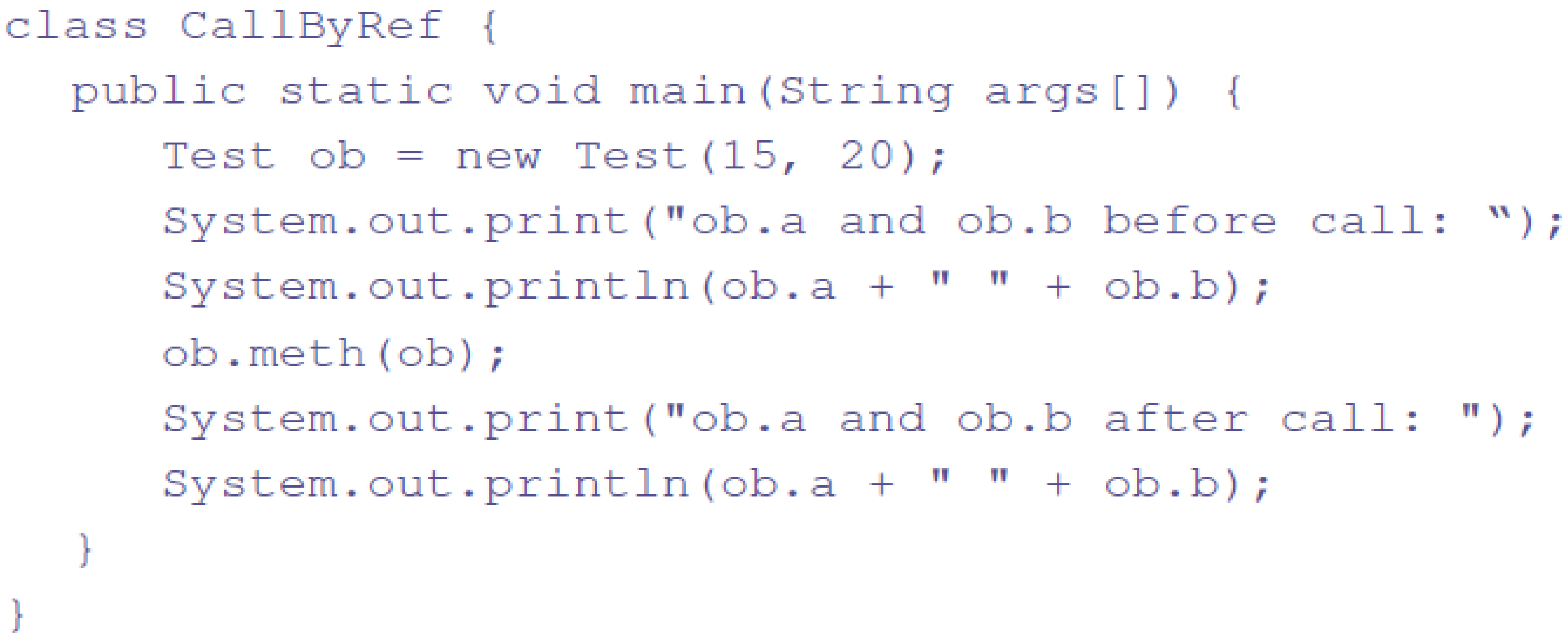
Objects are passed to the methods by reference: a parameter obtains the same address as the corresponding argument:



```
class Test {
   int a, b;

   Test(int i, int j) {
      a = i; b = j;
   }

   void meth(Test o) {
      o.a *=  2; o.b /= 2;
   }
}
```

Class Type Argument Passing

As the parameter hold the same address as the argument, changes to the object inside the method do affect the object used by the argument:

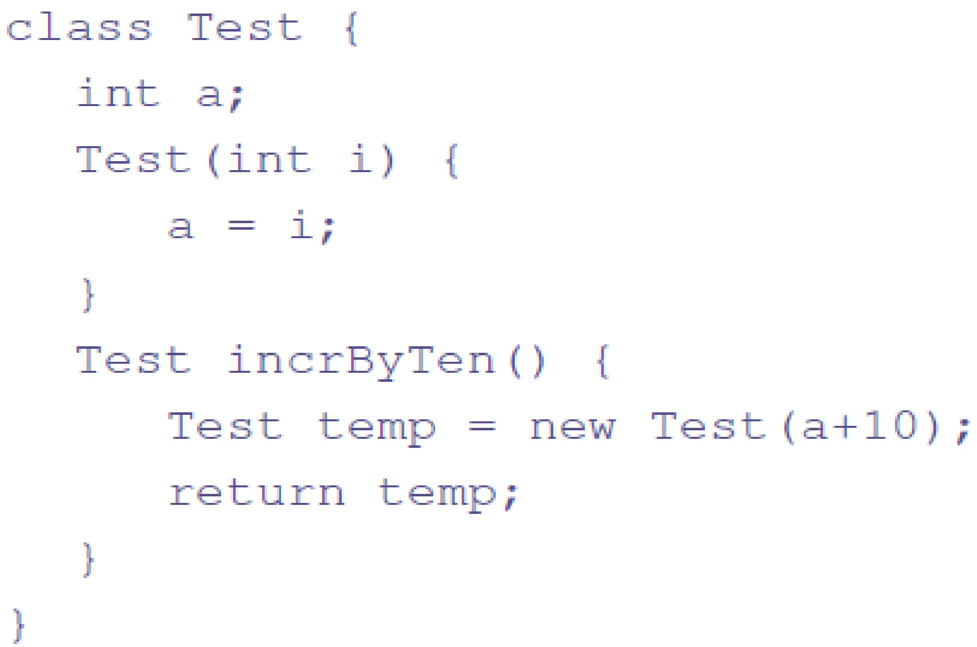


```
class CallByRef {
  public static void main(String args[]) {
     Test ob = new Test(15, 20);
     System.out.print("ob.a and ob.b before call: “);
     System.out.println(ob.a + " " + ob.b);
     ob.meth(ob);
     System.out.print("ob.a and ob.b after call: ");
     System.out.println(ob.a + " " + ob.b);
  }
}
```

Returning Objects

So far, all methods returned no values or values of simple types.

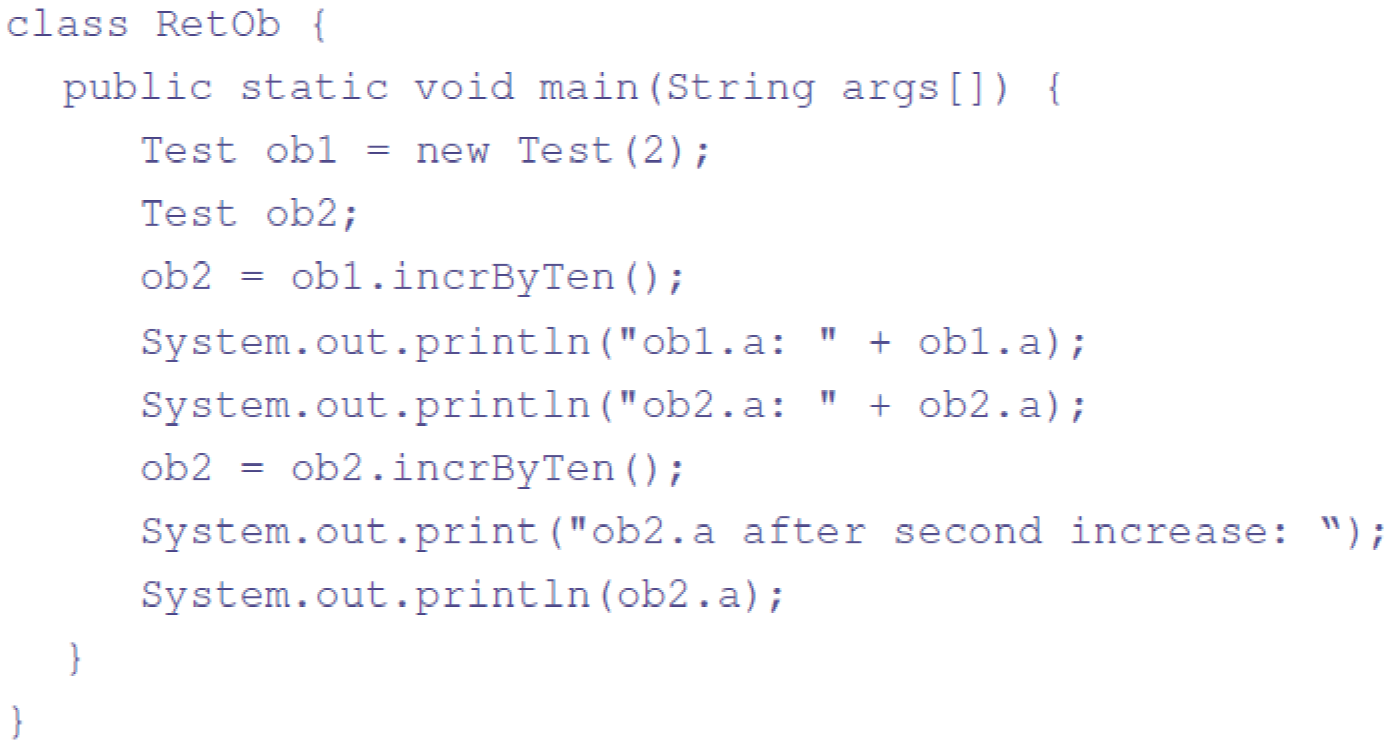
Methods may also return objects:



```
class Test {
   int a;
   Test(int i) {
      a = i;
   }
   Test incrByTen() {
      Test temp = new Test(a+10);
      return temp;
   }
}
```

Returning Objects

Each time a method **incrByTen** is invoked a new object is created and a reference to it is returned:



```
class RetOb {
  public static void main(String args[]) {
     Test ob1 = new Test(2);
     Test ob2;
     ob2 = ob1.incrByTen();
     System.out.println("ob1.a: " + ob1.a);
     System.out.println("ob2.a: " + ob2.a);
     ob2 = ob2.incrByTen();
     System.out.print("ob2.a after second increase: “);
     System.out.println(ob2.a);
  }
}
```

Static Class Members

Normally, the members of a class (its variables and methods) may be only used through the objects of this class.

Static members are independent of the objects:

- Variables
- Methods
- initialization block

All declared with the static keyword.

Static Variable

Static variable:

```
static int a;
```

Essentially, it a global variable shared by all instances of the class.

It cannot be used within a non-static method.

Static Methods

Static method:

- static void meth() { … }

Several restrictions apply:

- can only call static methods
- must only access static variables
- cannot refer to this

Static Block

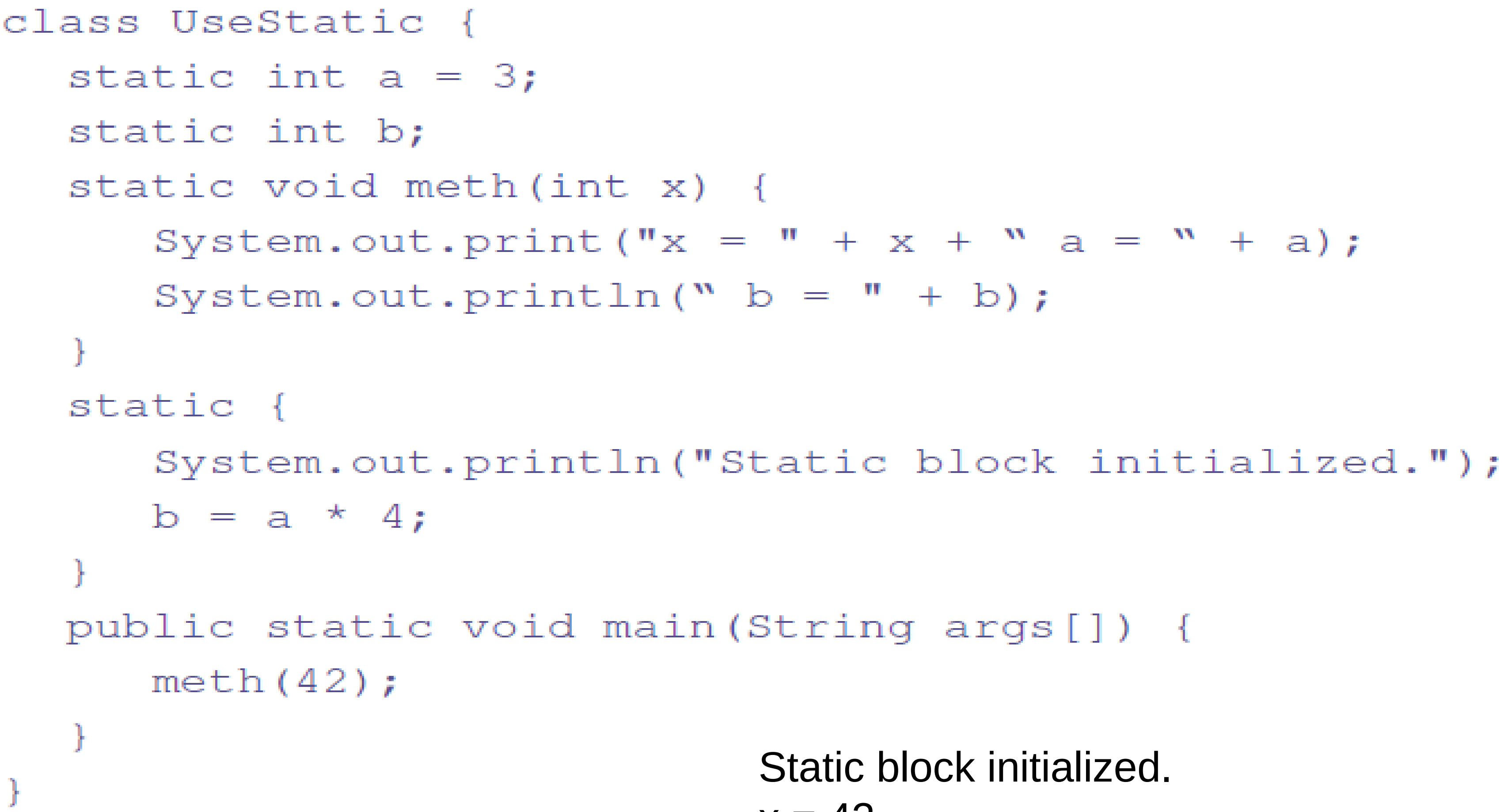
Static block:

- static { … }

This is where the static variables are initialized.

The block is executed exactly once, when the class is first loaded.

Example: Static



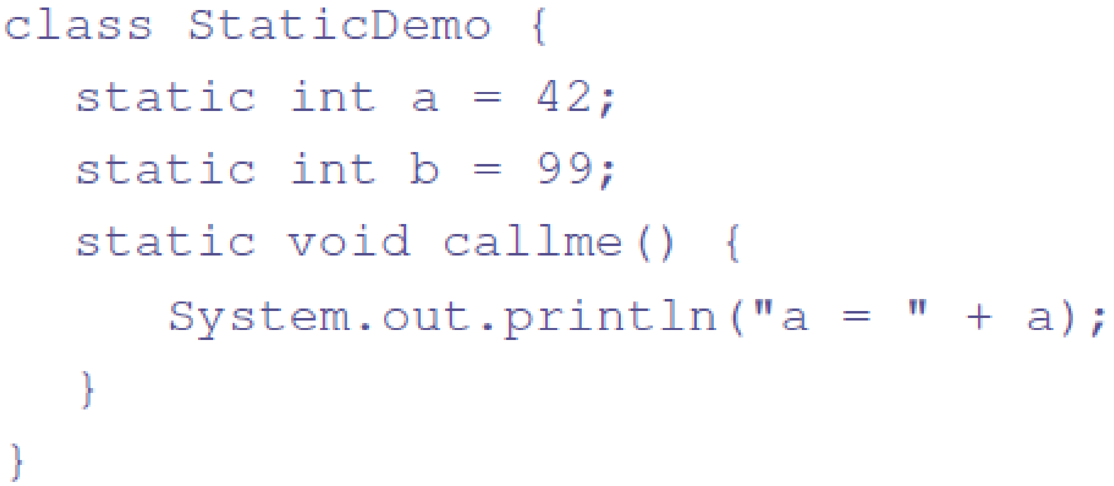
```
class UseStatic {
  static int a = 3;
  static int b;
  static void meth(int x) {
     System.out.print("x = " + x + “ a = “ + a);
     System.out.println(“ b = " + b);
  }
  static {
     System.out.println("Static block initialized.");
     b = a * 4;
  }
  public static void main(String args[]) {
     meth(42);
  }
}
```

Static block initialized.
x = 42
a = 3
b = 12

Static Member Usage

How to use static members outside their class?

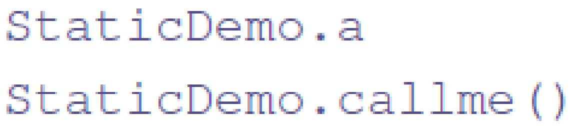
Consider this class:



```
class StaticDemo {
  static int a = 42;
  static int b = 99;
  static void callme() {
     System.out.println("a = " + a);
  }
}
```

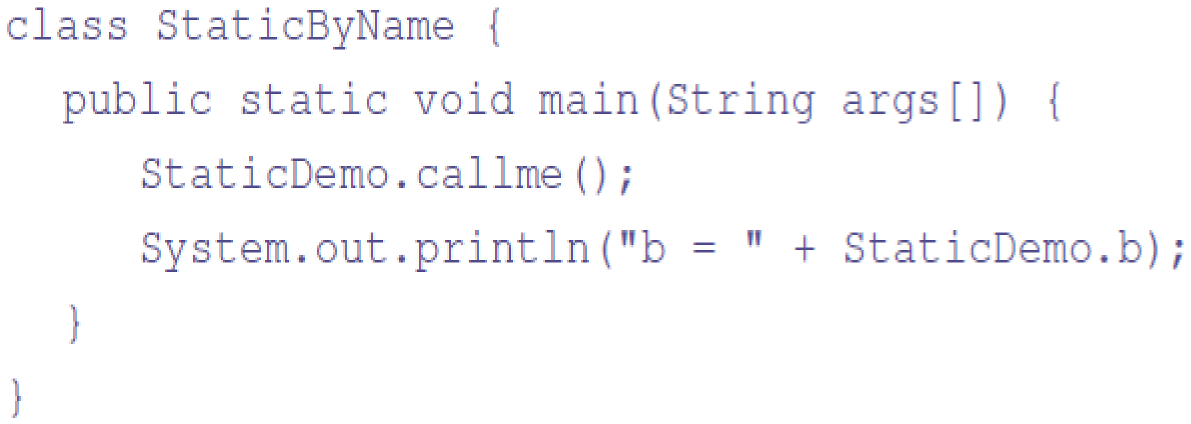
Static Member Usage

Static variables/method are used through the class name:



```
StaticDemo.a
StaticDemo.callme()
```

Example



```
class StaticByName {
  public static void main(String args[]) {
     StaticDemo.callme();
     System.out.println("b = " + StaticDemo.b);
  }
}
```

What happens in this code??

```
class demo
{
  public static void main(String args[])
  {
    demo d = new demo();
    d.add(10,20); // to call the non-static method
  }
  public void add(int x ,int y)
  {
     int a = x; int b = y; int c = a + b;
     System.out.println("addition" + c);
   }
 }
```

Class Participation

```
class Test1 {

public static void main(String[] args)

            {           int x = 20;

                        System.out.println(x);

            }

static

            {           int x = 10;

                        System.out.print(x + " ");

            }

}
```

Option
A) 10 20
B) 20 10
C) 10 10
D) 20 20

Class Participation

```
class Test1 {

          int x = 10;

public    static void main(String[] args)

          { System.out.println(x);

          }

static

          { System.out.print(x + " ");

          }

}
```

Option

A) 10 10
B) Error
C) Exception
D) none

Class Participation

```
class Test1 {
          int x = 10;
public static void main(String[] args)
          {          Test1 t1 = new Test1();
                      System.out.println(t1.x);
          }
          static
          {           int x = 20;
                      System.out.print(x + " ");
          } }
```

Option

A) 10 20
B) 20 10
C) 10 10
D) Error

QUESTIONS