DEFINING CLASSES

Pisit Nakjai

- What a class is, and how you defi ne a class
- How to implement class constructors
- How to defi ne class methods

WHAT IS A CLASS?

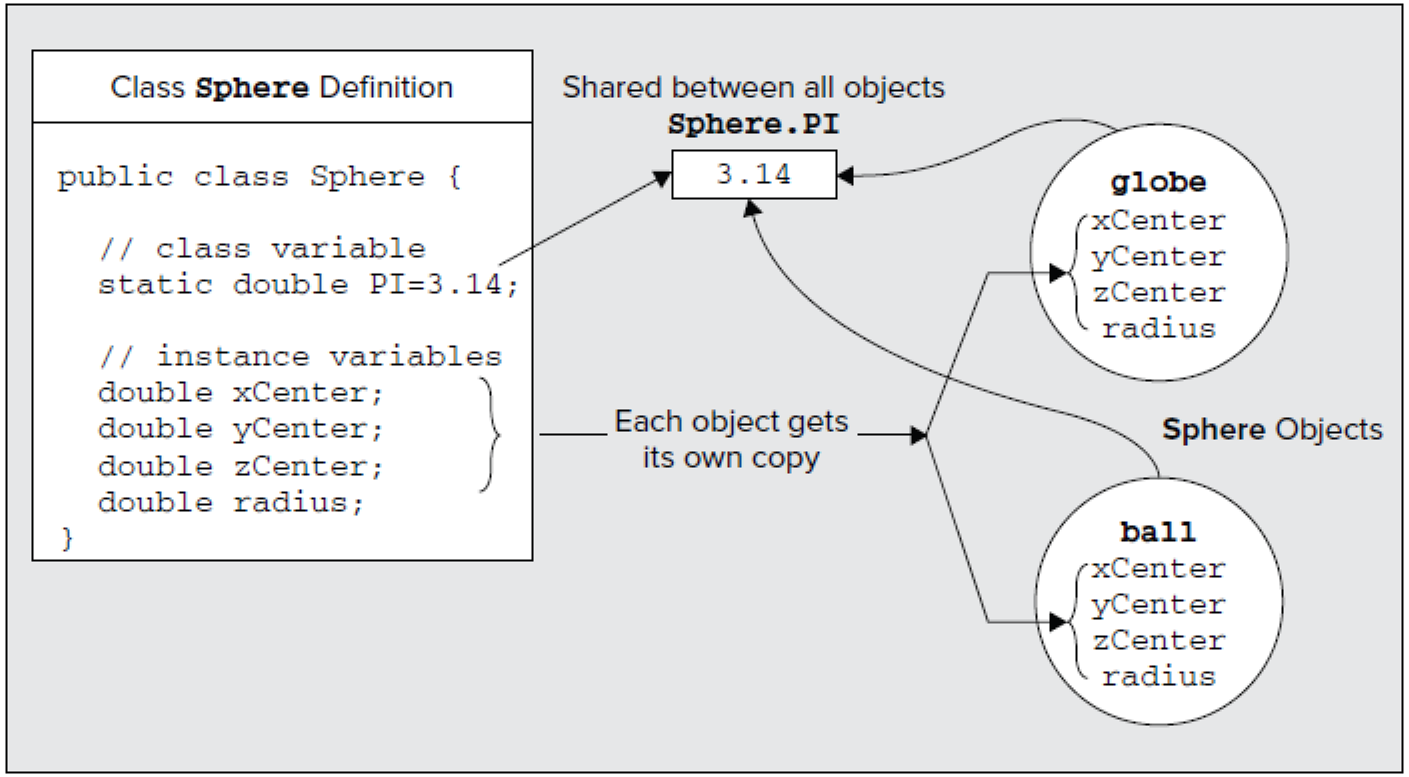
- a class is a prescription for a particular kind of object
- it defines a new *type*.
- the object contains all the fields that were included in the class definition

Fields in a Class Definition

- One kind of field is associated with the class and is shared by all objects of the class
- The other kind of field in a class is associated with each object uniquely
- summarize the two kinds of fields
  - **Non-static fields, also called instance variables** Each object has its own values for each instance variable
  - **Static fields, also called class variables :** given class has only one copy of each of its static fields or class variables, and these are shared between and among all the objects of the class

Fields in a Class Definition

- which illustrates the difference between class variables and instance variables



Class **Sphere** Definition

```
public class Sphere {

   // class variable
   static double PI=3.14;

   // instance variables
   double xCenter;
   double yCenter;
   double zCenter;
   double radius;
}
```

Methods in a Class Definition

- The methods for a class provide the actions that can be carried out using the variables specified in the class definition
- the variables in a class definition there are two varieties of methods
  - ***instance methods*** instance methods can be executed only in relation to a particular object,
  - ***class methods*.**
    - You can execute class methods even when no objects of a class exist
    - using the keyword **static**

Accessing Variables and Methods

- You can access a static member of a class using **the class name**, **followed by a period**, **followed by the member name**
- if you want to calculate the square root of π you can access the class method sqrt() and the class variable PI that are defined in the Math class as follows

```
double rootPi = Math.sqrt(Math.PI);
```

Final Fields

- If you declare a field in a class to be final the field cannot be modified by the methods in the class
- For example:

```
final double PI = 3.14;
```

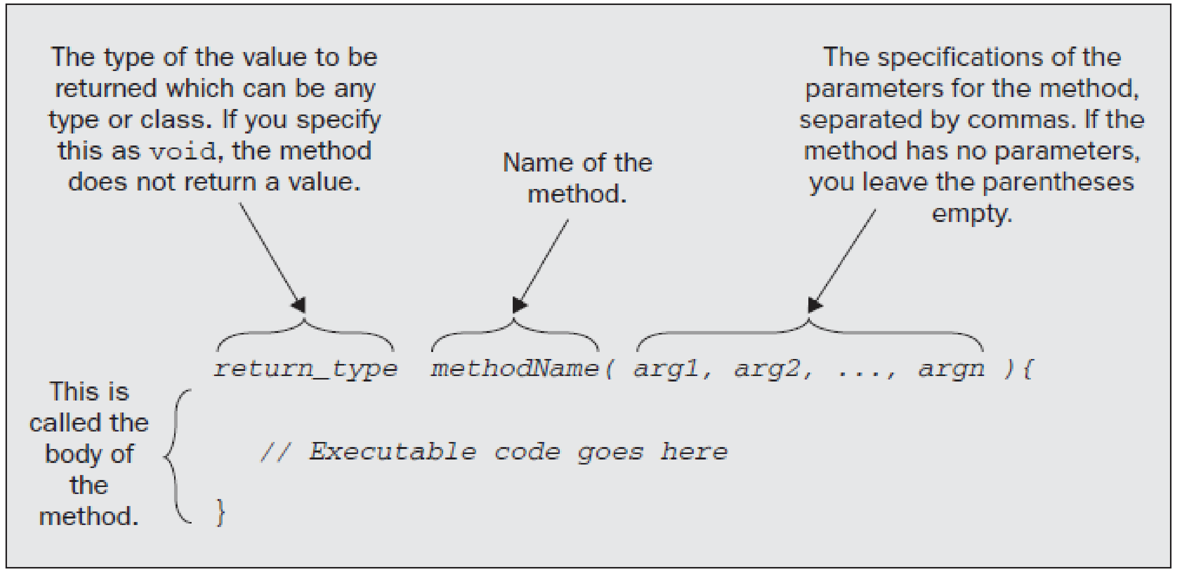
DEFINING CLASSES

- To define a class you use the keyword ***class*** followed by ***the name of the class***
- ***Example***

```
class Sphere {
  static final double PI = 3.14;       // Class variable that has a fixed value
  static int count = 0;                // Class variable to count objects

  // Instance variables
  double radius;                       // Radius of a sphere

  double xCenter;                      // 3D coordinates
  double yCenter;                      // of the center
  double zCenter;                      // of a sphere

  // Plus the rest of the class definition...
}
```

DEFINING METHODS

- A *method* is a self-contained block of code that has a name and has the property that it is reusable
- execute a method by *calling* it using its name
- the method may or may not return a value when its execution finishes

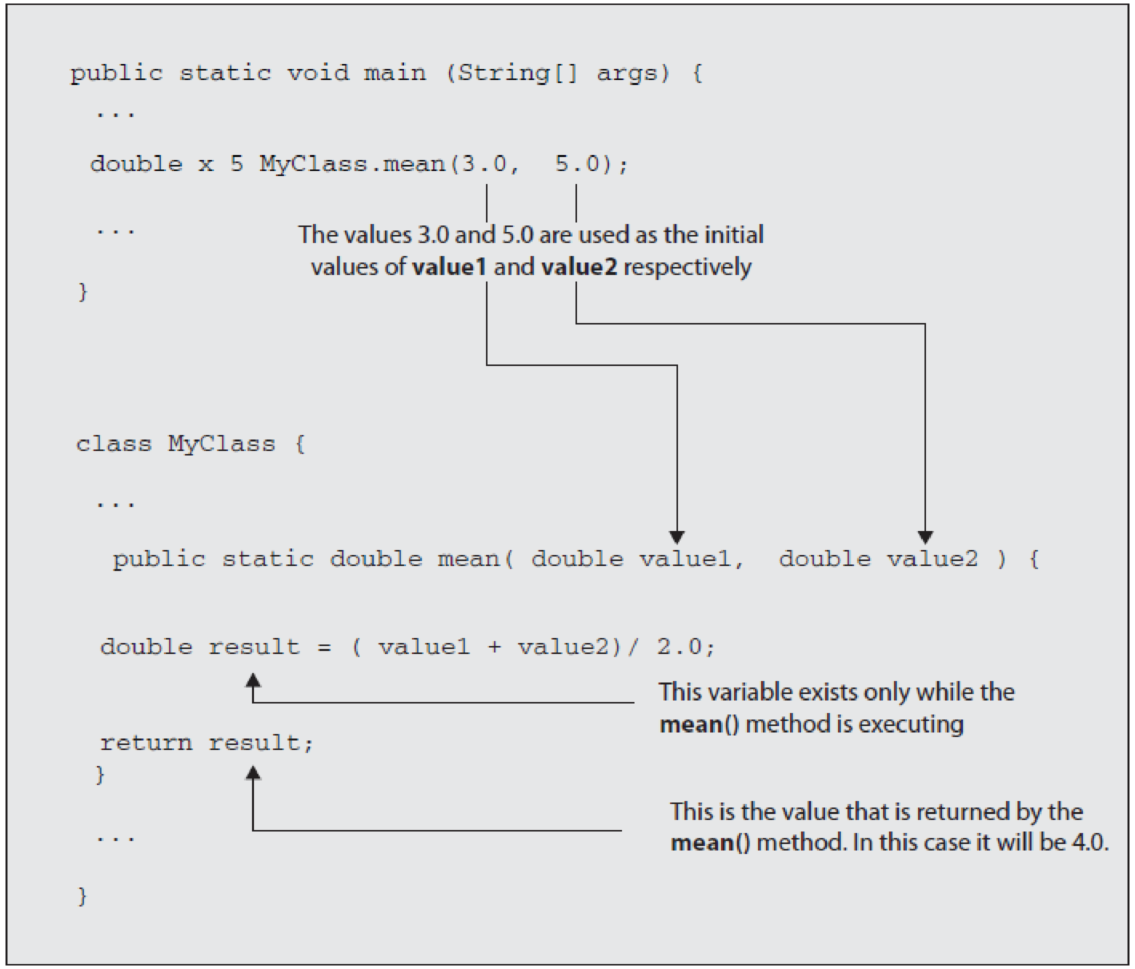

The type of the value to be returned which can be any type or class. If you specify this as void, the method does not return a value.
Name of the method.
The specifications of the parameters for the method, separated by commas. If the method has no parameters, you leave the parentheses empty.
This is called the body of the method.

```
return_type  methodName( arg1, arg2, ..., argn ){

    // Executable code goes here

}
```

The Parameter List

- The difference between a *parameter* and an *argument* is sometimes confusing
- A ***parameter*** has a name and a type and appears in the parameter list in the definition of a method
- An *argument* is a value that is passed to a method when it is executed

The Parameter List

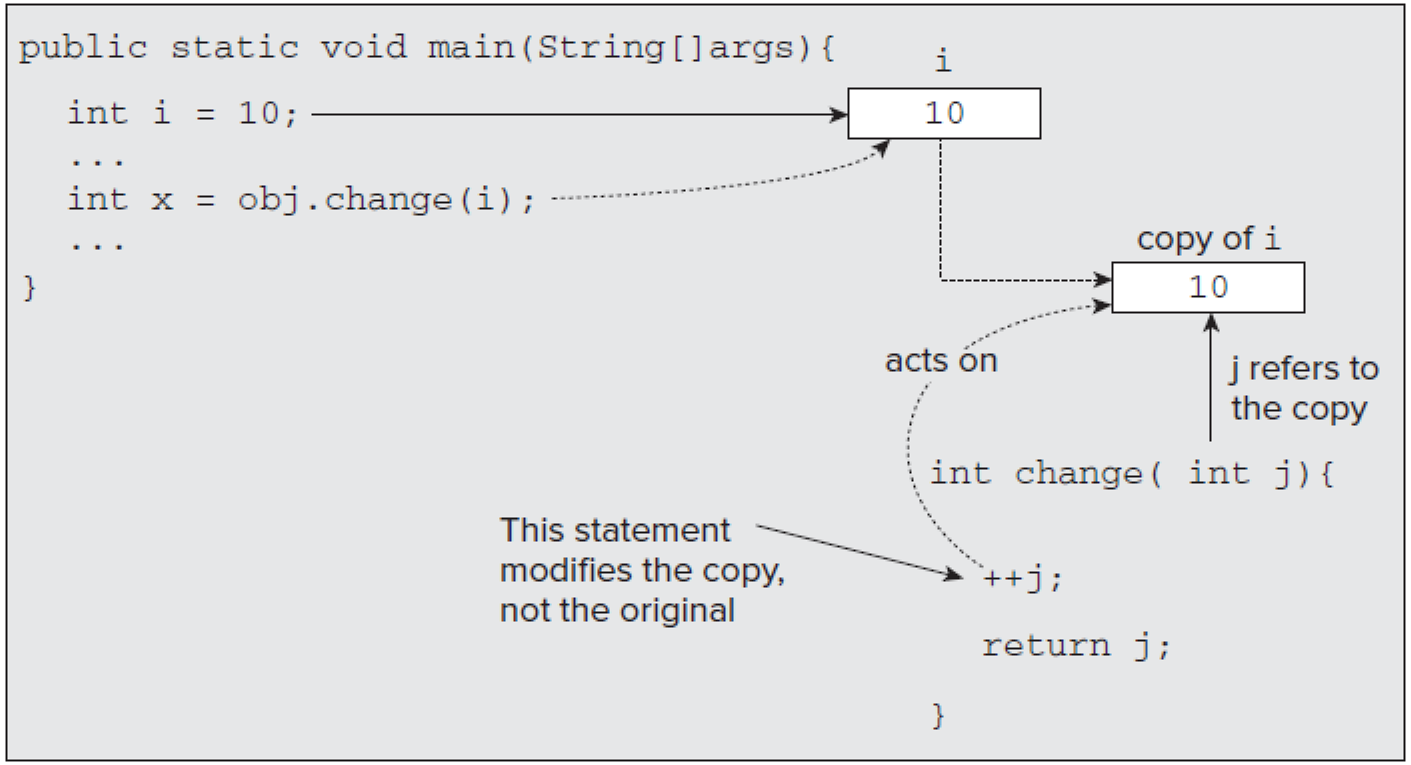


```
public static void main (String[] args) {
  ...
 double x 5 MyClass.mean(3.0,  5.0);

  ...
 }


 class MyClass {

  ...

   public static double mean( double value1,  double value2 ) {

  double result = ( value1 + value2)/ 2.0;

  return result;
  }

  ...

 }
```

The values 3.0 and 5.0 are used as the initial values of **value1** and **value2** respectively
This variable exists only while the **mean**() method is executing
This is the value that is returned by the **mean**() method. In this case it will be 4.0.

- the method has two parameters, value1 and value2
- The method mean() defines the variable result.
- All the variables that you declare within the body of a method are local to the method.
- Variables declared within a method are called ***local variables***

How Argument Values Are Passed to a Method



```
public static void main(String[]args){
   int i = 10;
   ...
   int x = obj.change(i);
   ...
}

int change( int j){
   ++j;
   return j;
}
```

- all argument values are transferred to a method using what is called the ***pass-by-value*** mechanism

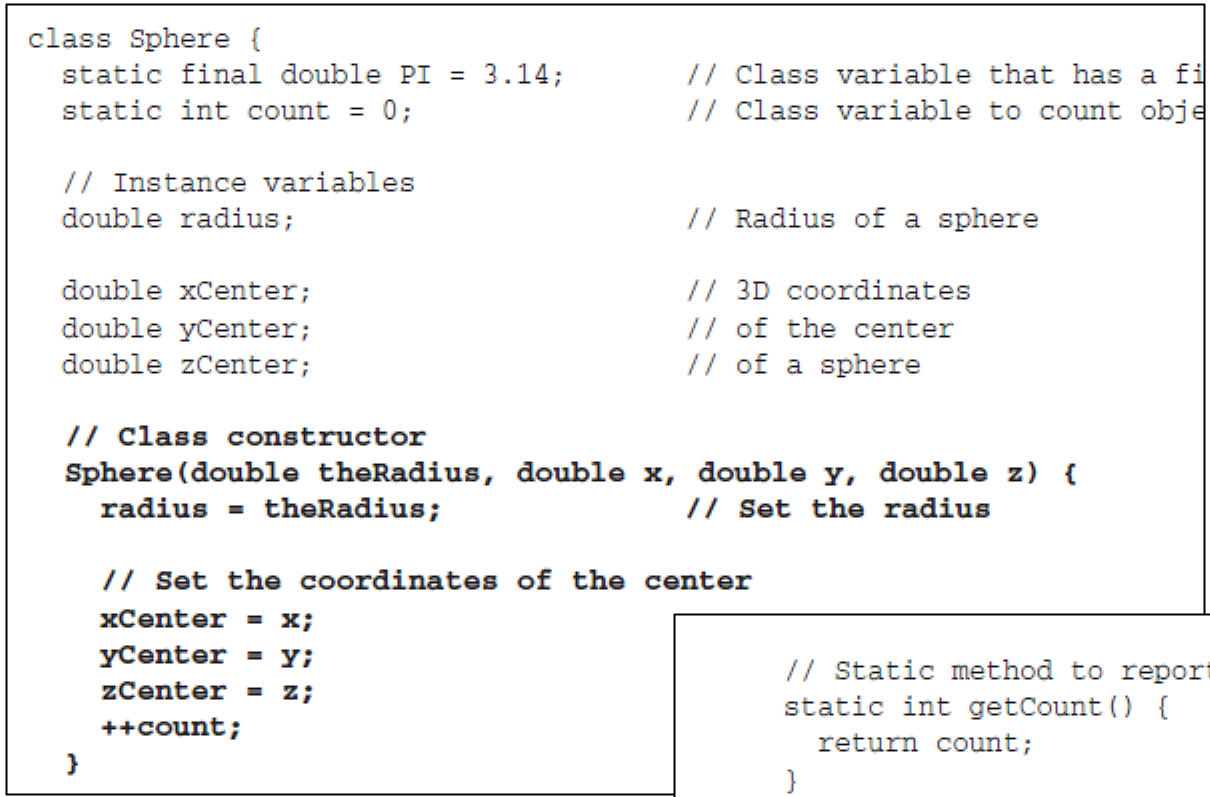
the effect for objects is different from that for variables of the primitive types.

when you use a variable of a class type as an argument to a method a copy of a ***reference*** to the object is passed to the method

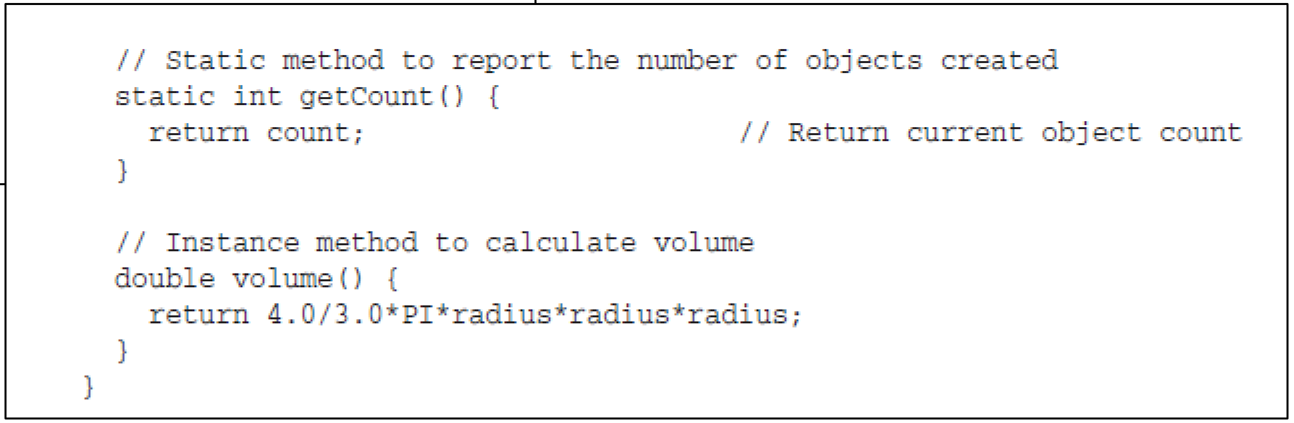
CONSTRUCTORS

- When you create an object of a class, a special kind of method called a ***constructor*** is always invoked
- If you don't define any constructors .the compiler supplies a ***default constructor*** in the class
- A constructor has two special characteristics that differentiate it from other class methods
  - A constructor never returns a value, and you must not specify a return type
  - A constructor always has the same name as the class.
- We must used constructor for declare initial value of object.

Class Sphere



```
class Sphere {
  static final double PI = 3.14;       // Class variable that has a fi
  static int count = 0;                // Class variable to count obje

  // Instance variables
  double radius;                       // Radius of a sphere

  double xCenter;                      // 3D coordinates
  double yCenter;                      // of the center
  double zCenter;                      // of a sphere

  // Class constructor
  Sphere(double theRadius, double x, double y, double z) {
    radius = theRadius;                // Set the radius

    // Set the coordinates of the center
    xCenter = x;
    yCenter = y;
    zCenter = z;
    ++count;
  }
```



```
  // Static method to report the number of objects created
  static int getCount() {
    return count;                      // Return current object count
  }

  // Instance method to calculate volume
  double volume() {
    return 4.0/3.0*PI*radius*radius*radius;
  }
}
```

Creating Objects of a Class

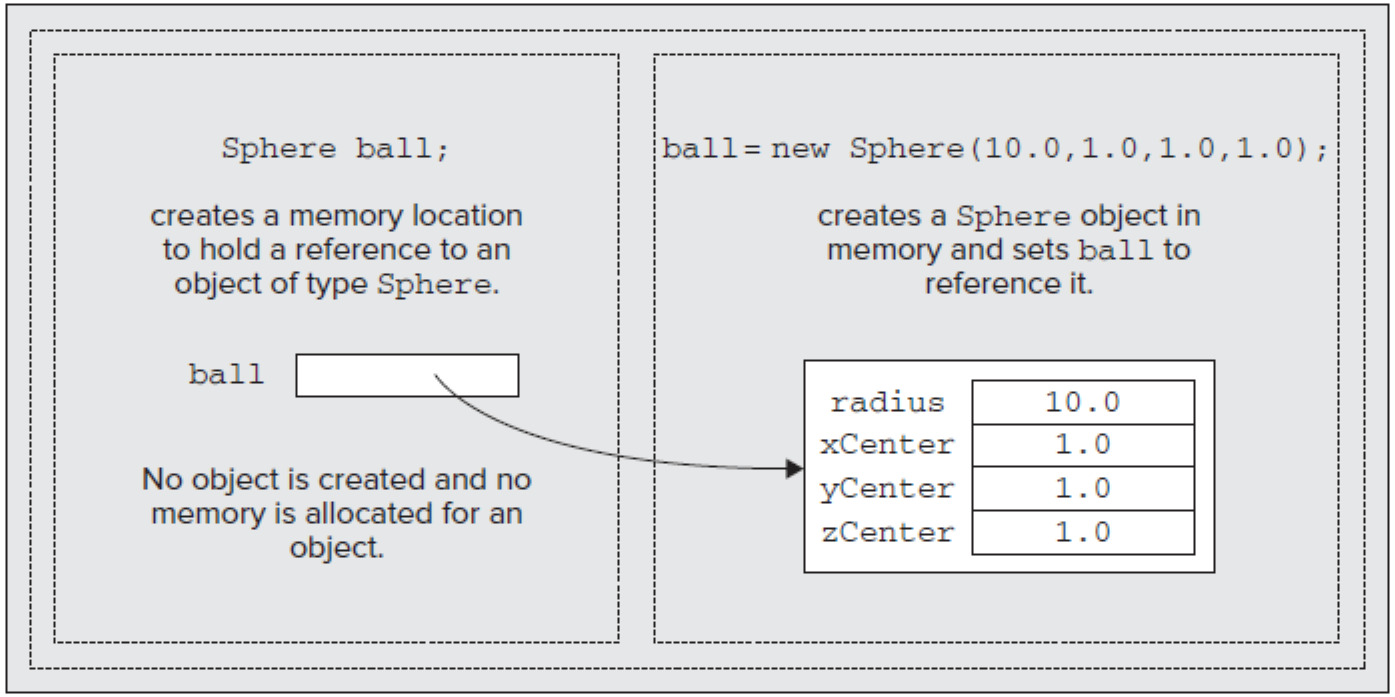
- When you declare a variable of type Sphere with the following statement

```
Sphere ball;
```

- no constructor is called because no object is created
- To create an object of a class you must use the keyword **new** followed by a call to a constructor.
- To initialize ball with a reference to an object, you could write

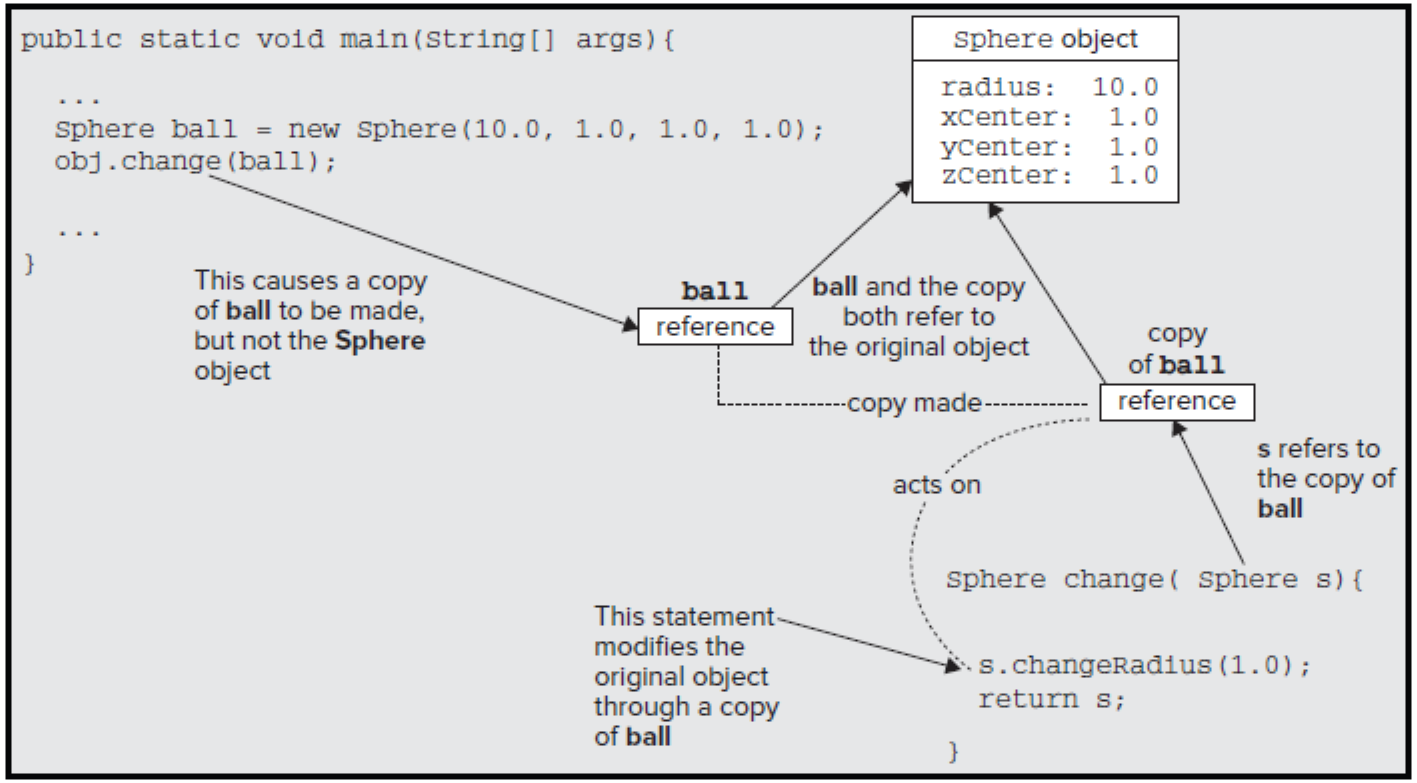
```
ball = new Sphere(10.0, 1.0, 1.0, 1.0);          // Create a sphere
```

Creating Objects of a Class


Sphere ball;
creates a memory location to hold a reference to an object of type Sphere.
ball
No object is created and no memory is allocated for an object.
ball = new Sphere(10.0,1.0,1.0,1.0);
creates a Sphere object in memory and sets ball to reference it.
radius
10.0
xCenter
1.0
yCenter
1.0
zCenter
1.0

Passing Objects to a Method

- When you pass an object as an argument to a method, the mechanism that applies is called ***pass-by-reference***

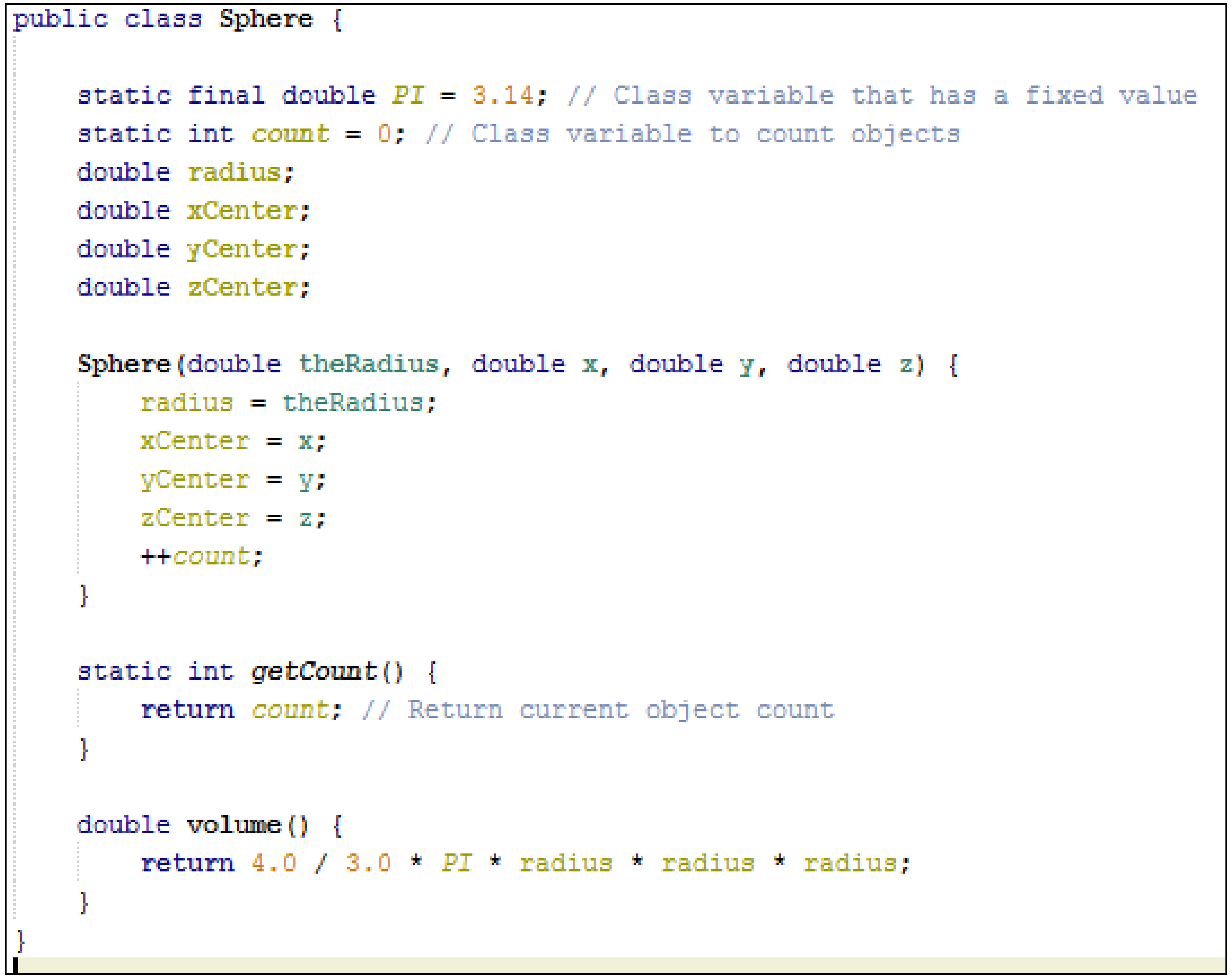


```
public static void main(String[] args){

  ...
  Sphere ball = new Sphere(10.0, 1.0, 1.0, 1.0);
  obj.change(ball);

  ...
}
```

```
Sphere change( Sphere s){

  s.changeRadius(1.0);
  return s;

}
```

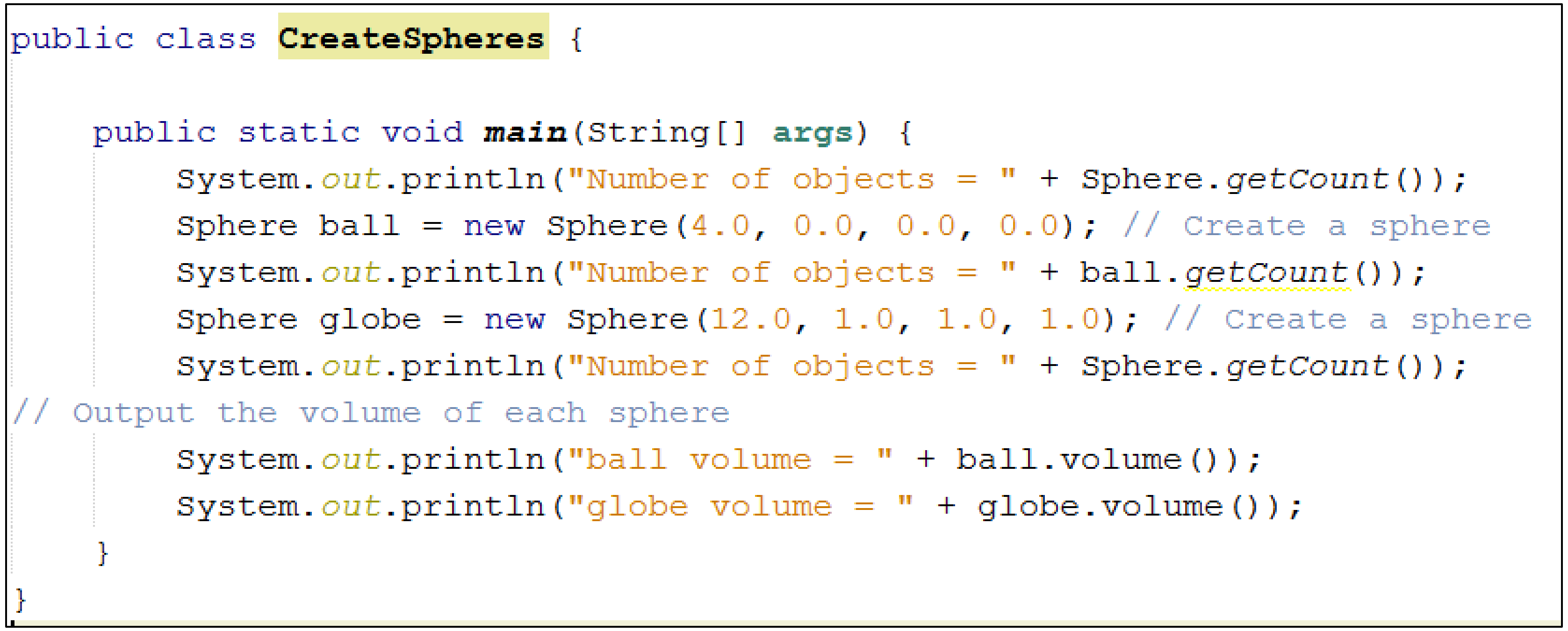
Sphere object
radius: 10.0
xCenter: 1.0
yCenter: 1.0
zCenter: 1.0
This causes a copy of **ball** to be made, but not the **Sphere** object
ball
reference
ball and the copy both refer to the original object
copy of **ball**
reference
copy made
acts on
s refers to the copy of **ball**
This statement modifies the original object through a copy of **ball**

TRY IT OUT : Using the Sphere Class



```
public class Sphere {

    static final double PI = 3.14; // Class variable that has a fixed value
    static int count = 0; // Class variable to count objects
    double radius;
    double xCenter;
    double yCenter;
    double zCenter;

    Sphere(double theRadius, double x, double y, double z) {
        radius = theRadius;
        xCenter = x;
        yCenter = y;
        zCenter = z;
        ++count;
    }

    static int getCount() {
        return count; // Return current object count
    }

    double volume() {
        return 4.0 / 3.0 * PI * radius * radius * radius;
    }
}
```

TRY IT OUT : Using the Sphere Class



```
public class CreateSpheres {

    public static void main(String[] args) {
        System.out.println("Number of objects = " + Sphere.getCount());
        Sphere ball = new Sphere(4.0, 0.0, 0.0, 0.0); // Create a sphere
        System.out.println("Number of objects = " + ball.getCount());
        Sphere globe = new Sphere(12.0, 1.0, 1.0, 1.0); // Create a sphere
        System.out.println("Number of objects = " + Sphere.getCount());
// Output the volume of each sphere
        System.out.println("ball volume = " + ball.volume());
        System.out.println("globe volume = " + globe.volume());
    }
}
```

Homework

- ให้นักศึกษาทำการสร้าง Method เพิ่ม ใน Class Sphere เพื่อทำการระยะห่างระหว่าง จุด center ของทรงกลมของ 2 ทรงกลม ว่ามีระยะห่างกันเท่าใด และให้ Method ส่งค่าที่ได้ออกจาก Method
- ให้นักศึกษาทำการสร้าง Method เพิ่มใน Class Sphere เพื่อทำการตรวจ ว่า ทรงกลม ทั้ง 2 ทรงกลมชนกันหรือไม่ หากตรวจสอบแล้วไม่มีการชนกันให้แสดง It’s Ok หากตรวจสอบแล้วชนกัน ให้แสดงว่า It’s crash