Classes

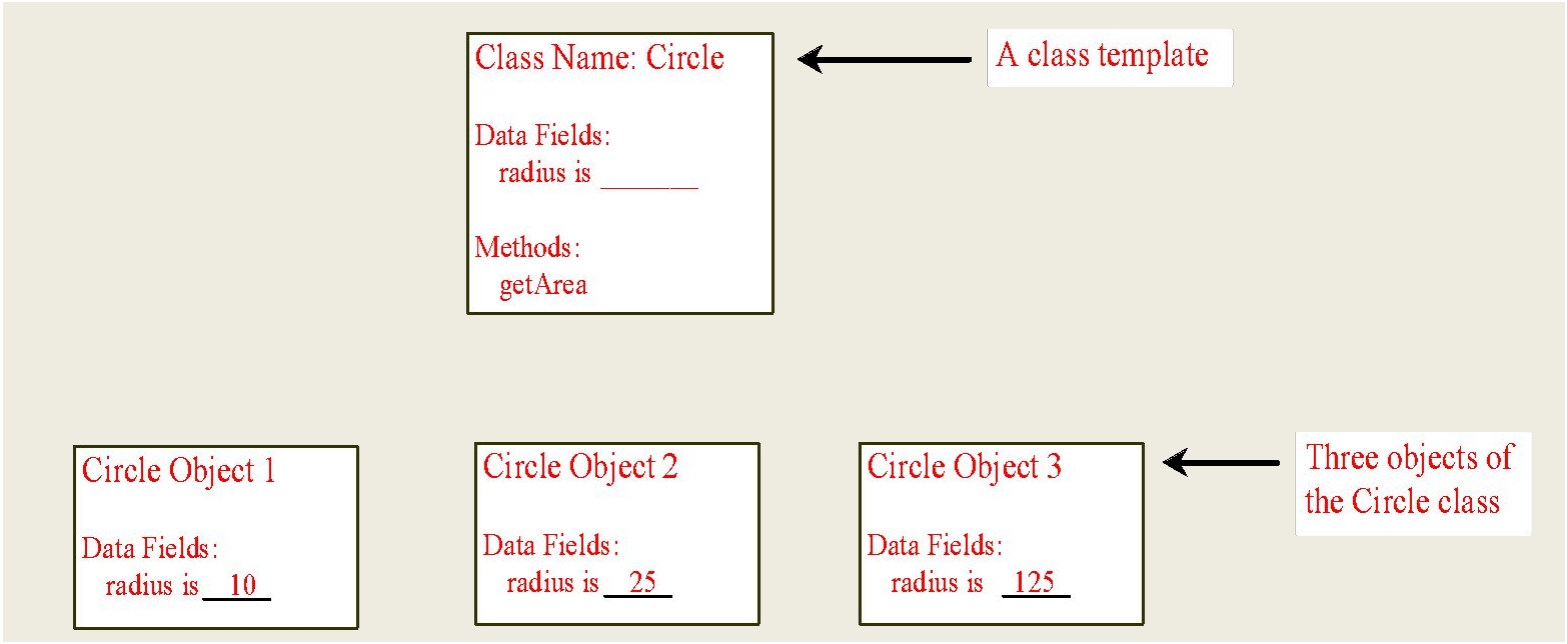
- A class is a user-defined data type. Once defined, this new type can be used to create variables of that type.

- These variables are termed as instances of classes, which are the actual objects.

- A class is a template for an object, and an object is an instance of a class.

By: RAJU PAL

Classes and Objects



An object has both a state and behavior. The state defines the object, and the behavior defines what the object does.

By: RAJU PAL

Classes Cont…

//Basic form of a class

```
class classname
{
        type variable1
        type variable2
        --------------
        ---------------
        type methodname1(parameter_list)
        {
                //body of method
        }
        type methodname2(parameter_list)
        {
                //body of method
        }
        --------------
}
```

//A Simple Class

```
class Box
{
        double width
        double height
        double depth
}
```

Data is encapsulated in a class by placing data fields inside the body of the class definition.

Declaring (creating) Objects

✔ **Declaring Object Reference Variables**

```
ClassName objectReference;
```

Example: `Box myBox;`

✔ **Creating Objects**

```
objectReference = new ClassName();
```

Example: `myBox = new Box();`

The object reference is assigned to the object reference variable.

✔ **Declaring/Creating Objects in a Single Step**

```
ClassName objectReference = new ClassName();
```

Example: `Box myBox = new Box();`

By: RAJU PAL

Accessing Objects

- Referencing the object's data:

```
objectRefVar.data
```

e.g., `myBox.width`

`myBox.height`

`myBox.depth`

- Invoking the object's method:

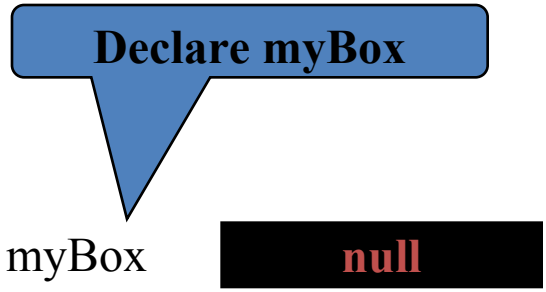
```
objectRefVar.methodName(arguments)
```

e.g., `myBox.getArea()`

Classes and Objects

Trace Code

```
Box myBox = new Box();

Box yourBox = new Box();

yourBox.width = 100;

yourBox.height = 10;

yourBox.depth= 50;
```

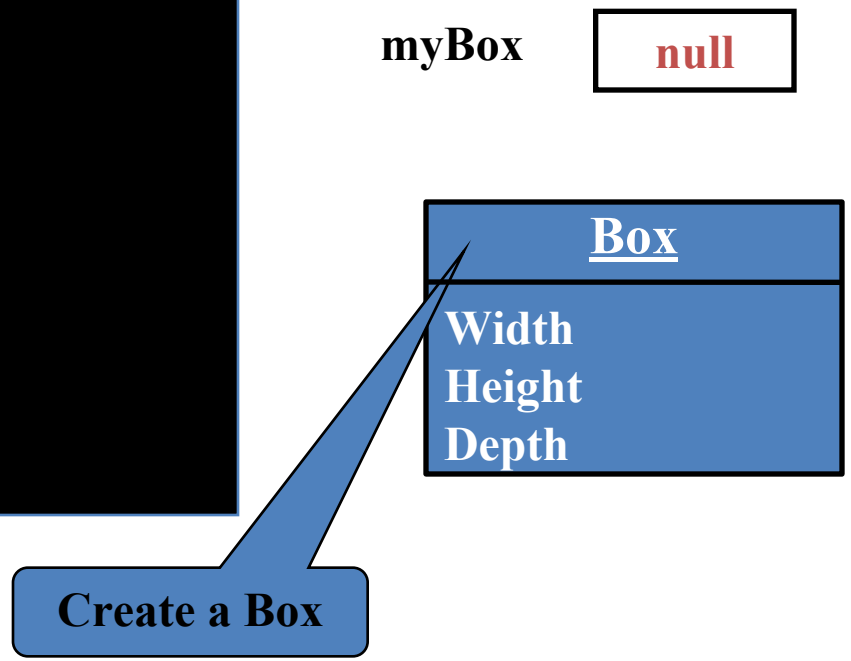


By: RAJU PAL

Classes and Objects

Trace Code, cont.

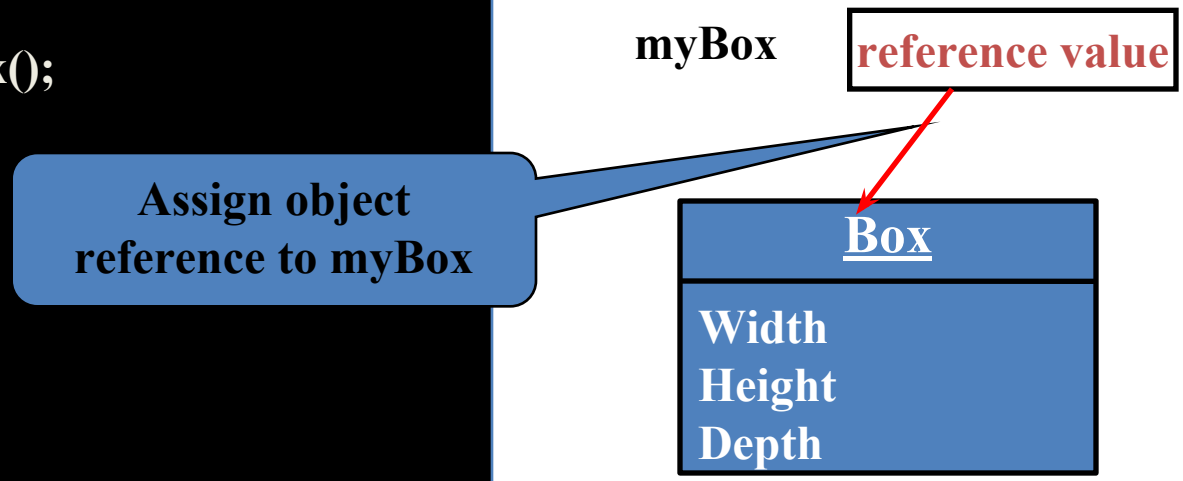
```
Box myBox = new Box();

Box yourBox = new Box();

yourBox.width = 100;

yourBox.height = 10;

yourBox.depth= 50;
```



Classes and Objects

Trace Code, cont.

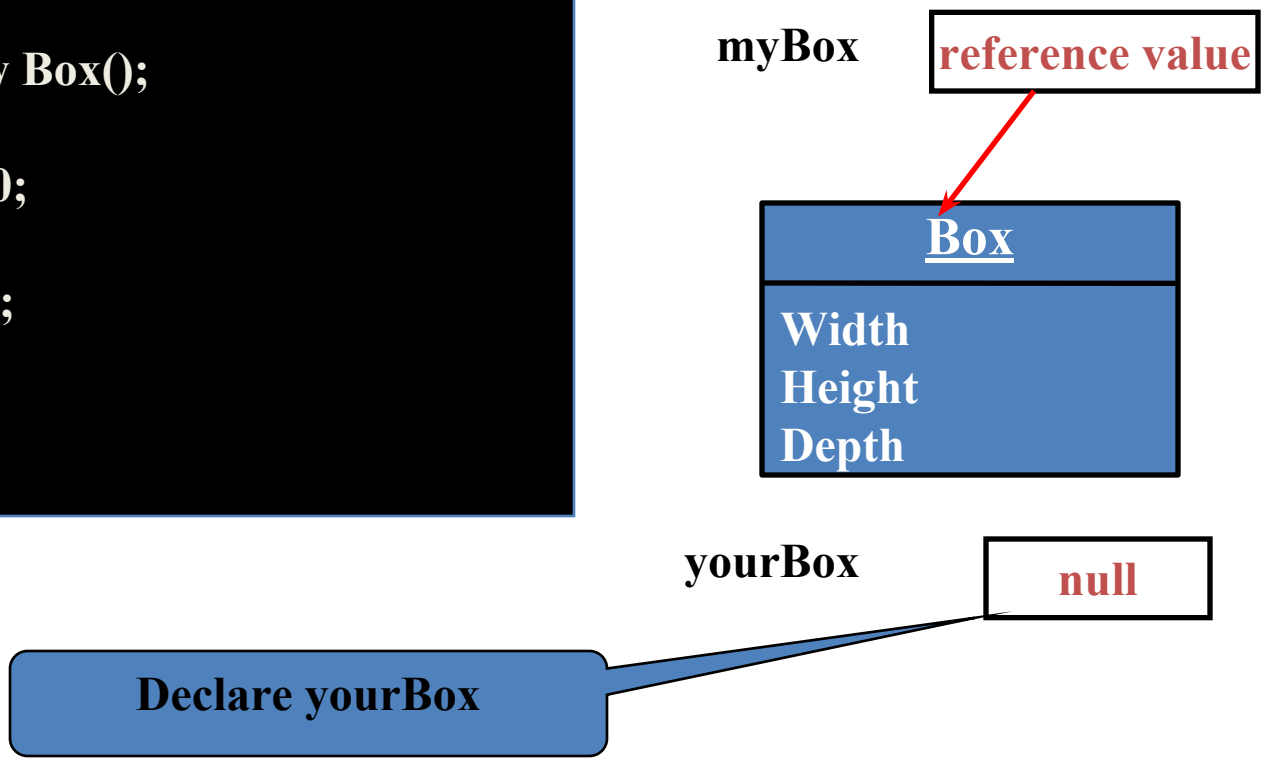
```
Box myBox = new Box();

Box yourBox =  new Box();

yourBox.width = 100;

yourBox.height = 10;

yourBox.depth= 50;
```



Classes and Objects

Trace Code, cont.

```
Box myBox  =  new Box();

Box yourBox  =  new Box();

yourBox.width = 100;

yourBox.height = 10;

yourBox.depth= 50;
```

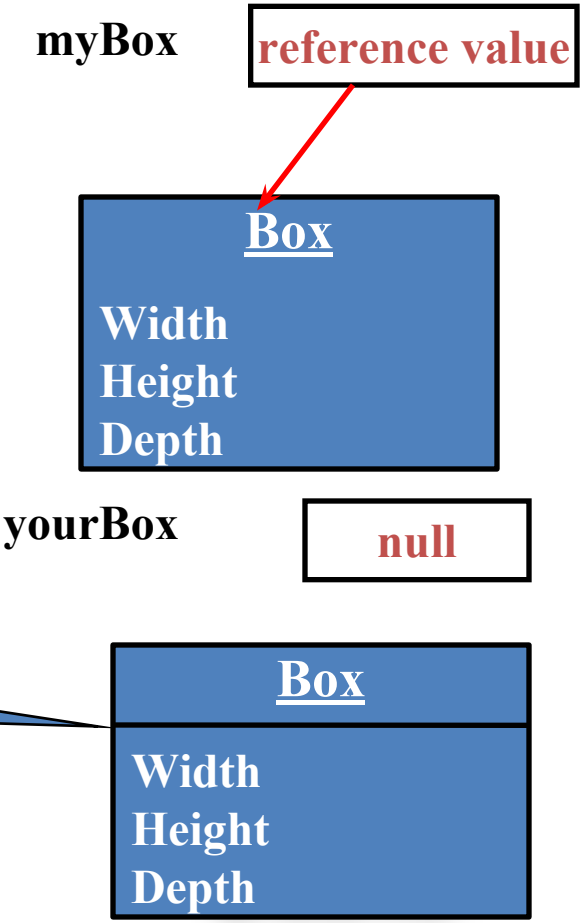


By: RAJU PAL

Classes and Objects

Trace Code, cont.

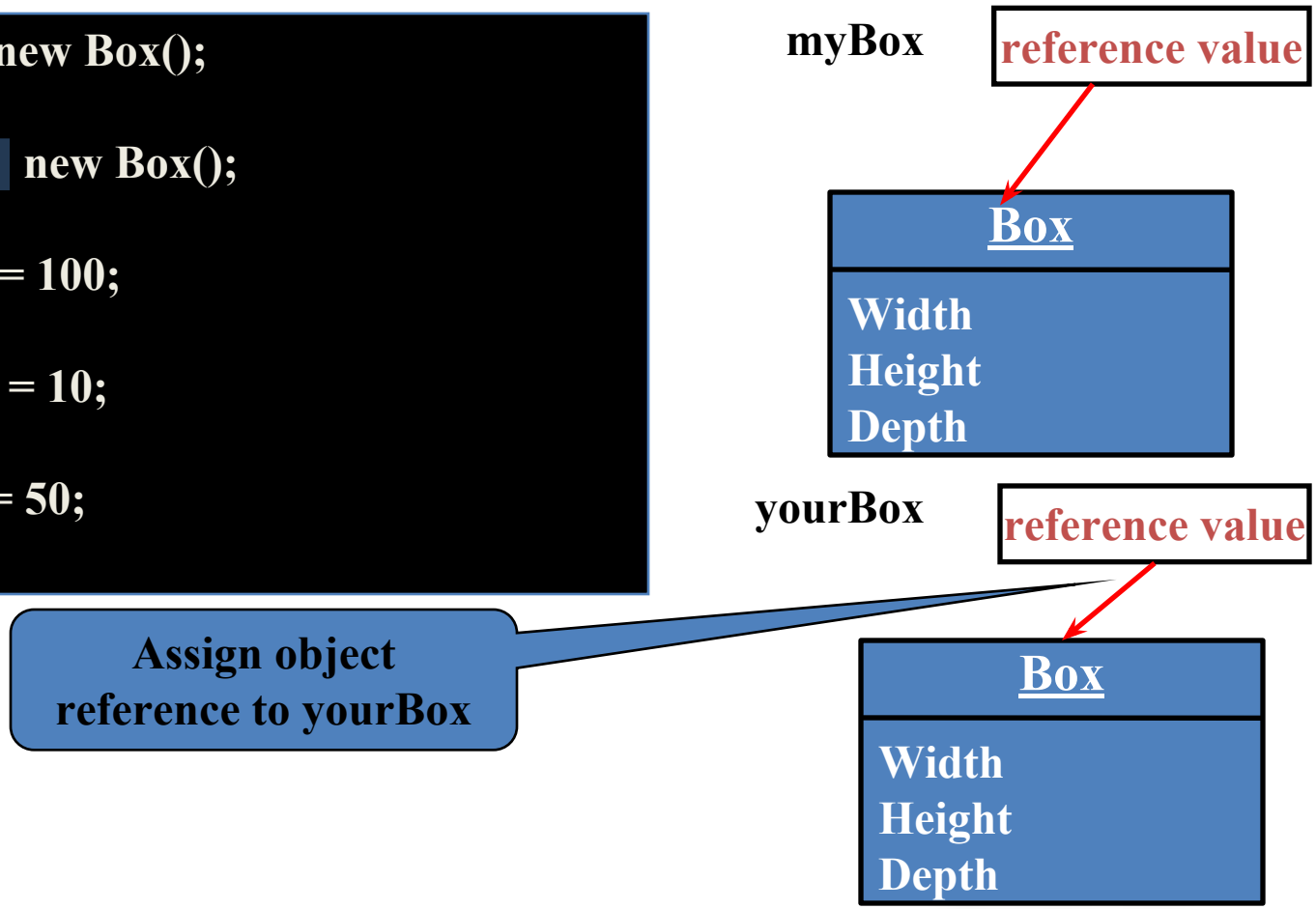
```
Box myBox  =  new Box();

Box yourBox  =  new Box();

yourBox.width = 100;

yourBox.height = 10;

yourBox.depth= 50;
```



Classes and Objects

Trace Code, cont.

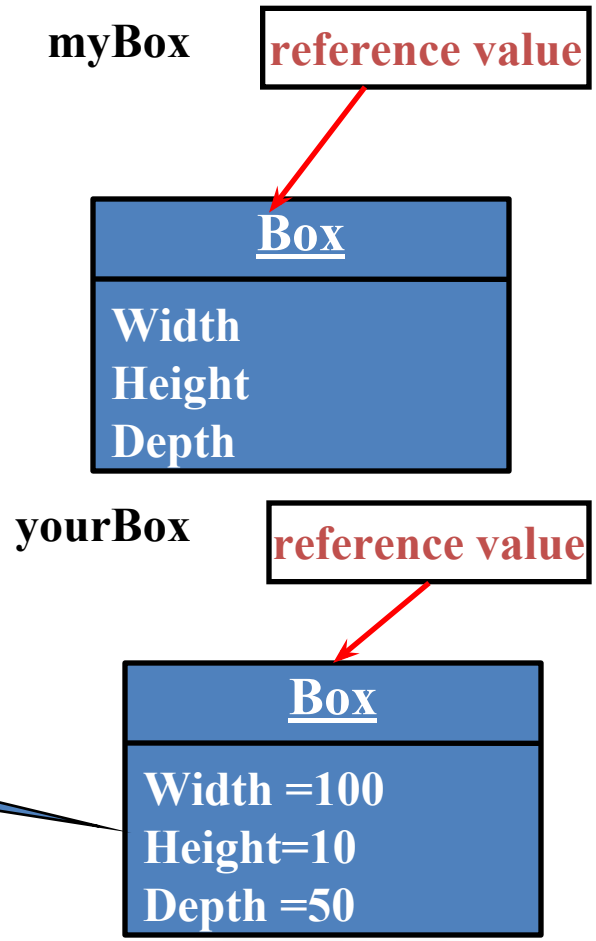
```
Box myBox  =  new Box();

Box yourBox = new Box();

yourBox.width = 100;

yourBox.height = 10;

yourBox.depth= 50;
```



Classes and Objects

Trace Code, cont.

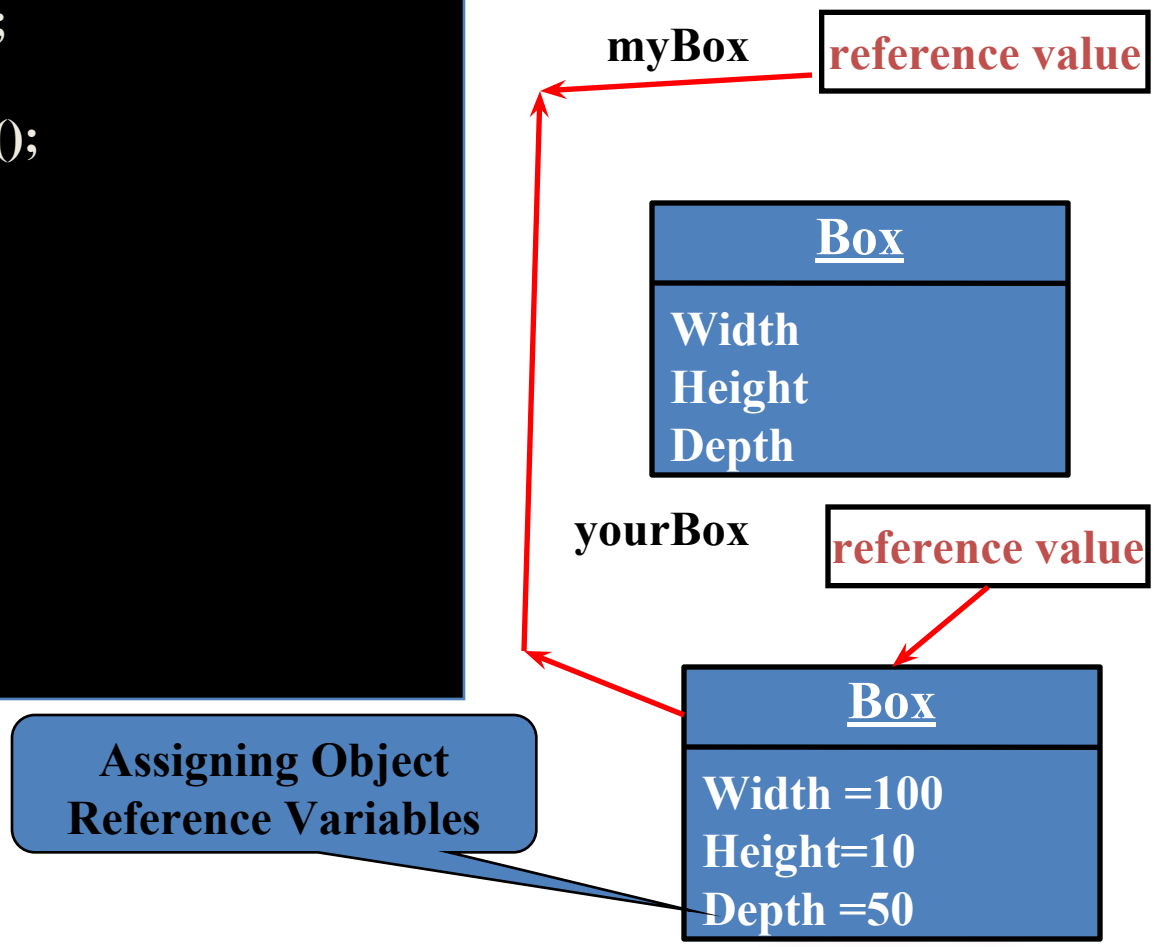
```
Box myBox  =  new Box();

Box yourBox  =  new Box();

yourBox.width = 100;

yourBox.height = 10;

yourBox.depth= 50;
```



Classes and Objects

Trace Code, cont.

```
Box myBox  =  new Box();

Box yourBox  =  new Box();

yourBox.width = 100;

yourBox.height = 10;

yourBox.depth= 50;

myBox = yourBox;
```



Garbage Collection

- When no references to an object exist, that object is assumed too be no longer needed, and the memory occupied by the object can be reclaimed.
- As shown in the previous figure, after the assignment statement myBox = yourBox, myBox points to the same object referenced by yourBox. The object previously referenced by myBox is no longer referenced. This object is known as garbage. Garbage is automatically collected by JVM.

Garbage Collection, cont…

- If you know that an object is no longer needed, you can explicitly assign null to a reference variable for the object. The JVM will automatically collect the space if the object is not referenced by any variable.

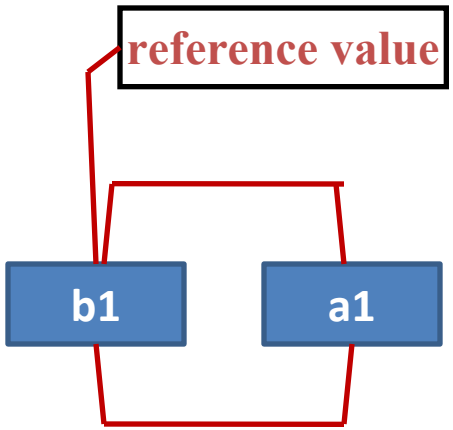
The *finalize()* Method

- Sometimes an object will need to perform some action when it is destroyed.
- The garbage collector calls a special method named *finalize* in your object if that method exists.
- If an object hold some non-Java resources (file handle or window character font) or any reference to other objects, these resources can be freed using *finalize* method.
- Avoiding circular reference.

```
protected void finalize()
{
   //finalization code here
}
```

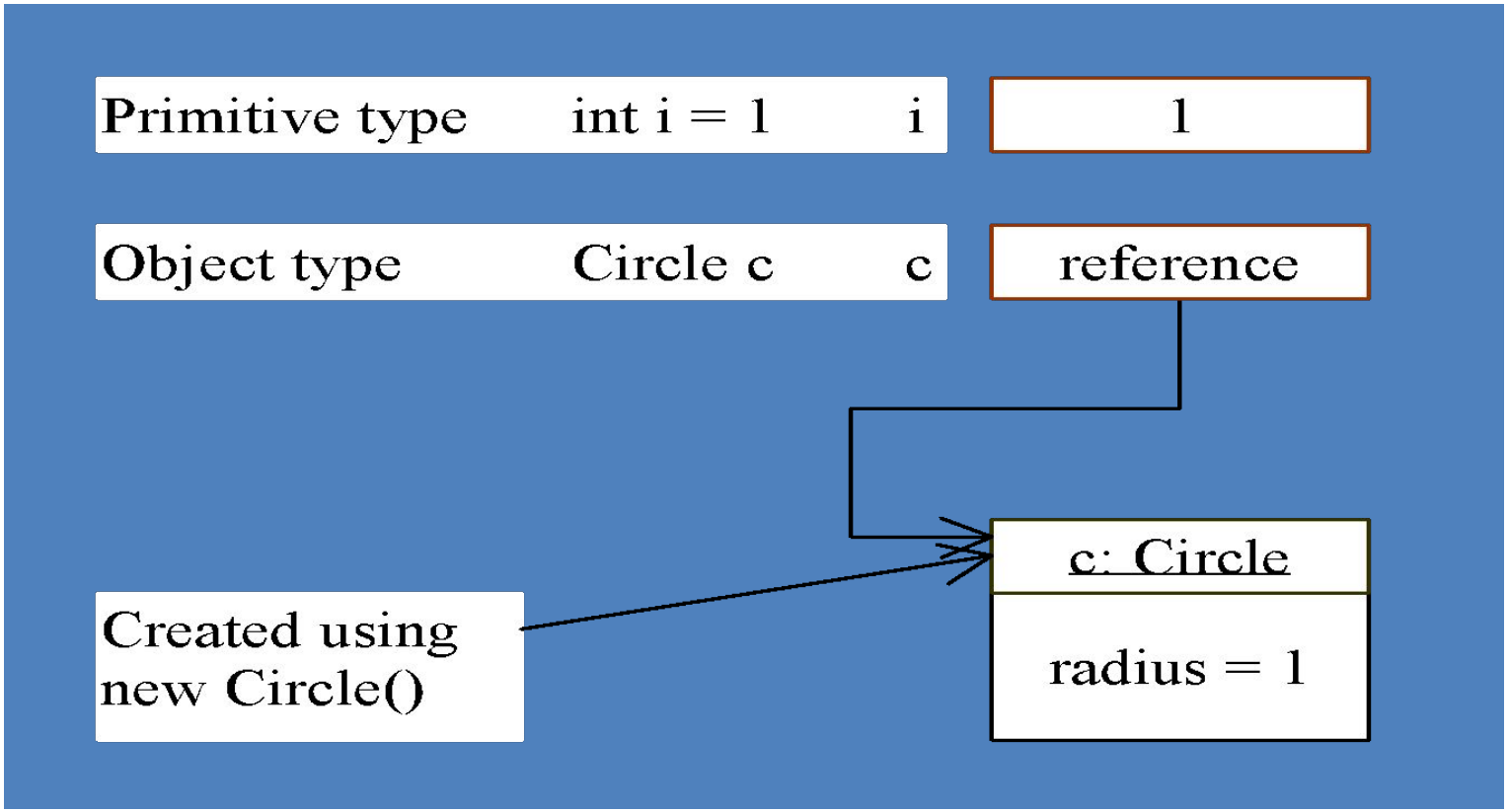
Circular References

```
Class a{
 b b1;
 a(){ b1 = new b();}
}
Class b{
 a a1;
 b(){ a1 = new a();}
}
public class app
{
   public static void main(String args[])
   {
      a obj = new a();
      obj = null;
   }
}
```



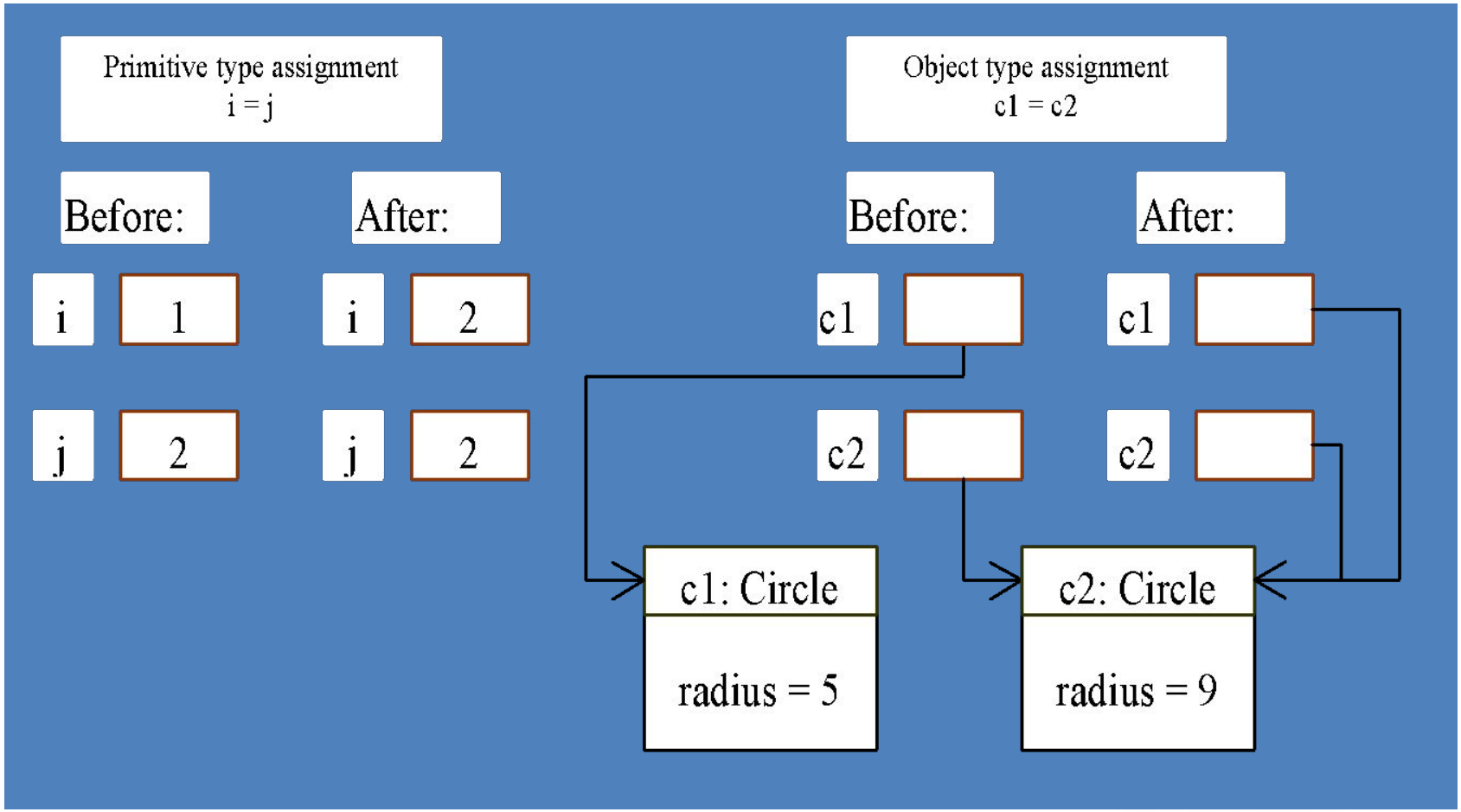
By: RAJU PAL

Differences between Variables of Primitive Data Types and Object Types



By: RAJU PAL

Copying Variables of Primitive Data Types and Object Types



By: RAJU PAL

Introducing Methods

type **name**(*parameter-list*)
{
    //body of method
    return value; (if type is not **void**)
}

- Define the interface to most classes.
- Hide specific layout of internal data structures(Abstraction)
- Add method to Box class

```
void volume()
{
   System.out.print("Volume is: ");;
   System.out.println(width*height*depth);
}
```

Returning a Value

```
double volume()
{
   return width*height*depth;
}
```

- The type of data returned by a method must be compatible with the return type specified by the method.

- The variable receiving the value returned by a method must also be compatible with the return type specified for the method.

Parameterized Methods

```
void setDim(double w, double h, double d)
{
   width = w;
   height = h;
   depth = d;
}
```

Parameters: variable defined by a method that receives a value when the method is called

Arguments: value that is passed to a method when it is invoked.

Constructors

```
Box()
{   //default constructor
}

Box()
{
  width = 10;
  height = 10;
  depth = 10;
}

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

}
Box myBox =new  Box();
Box yourBox = new Box(20,20,20);
```

- Constructors are a special kind of methods that are invoked to construct objects.
- A Constructor initializes an object immediately upon creation.
- Automatic initialization.

By: RAJU PAL

Constructors cont…

A constructor with no parameters is referred to as a *default constructor*.

- Constructors must have the same name as the class itself.
- Constructors do not have a return type—not even void.
- Implicit return type of a class' constructor is the class type itself.
- Constructors are invoked using the new operator when an object is created.
- Constructors play the role of initializing objects.

The *this* Keyword

- *this* can be used inside any method to refer to the current object.
- When you want to pass the current object to a method.
- To resolve any name space collisions that might occur between instance variables and local variables.

```
Box(double width, double height, double depth)
{
    this.width = width;
    this.height = height;
    this.depth = depth;
}
```

The *this* Keyword Cont…

```
class Data
{
     private String data_string;
     Data(String S){data_string = s;}
     public String getData(){return data_string;}
     public void printData()
     {
          Printer p = new Printer();
          p.print(this);
     }
}
class Printer
{
     void print(Data d){System.out.println(d.getData());}
}
public class app
{
     public static void main(String args[])
     {
           (new Data("Hello from Java")).printData();
     }
}
```

By: RAJU PAL

Overloading Methods

- *Method overloading* defines several different versions of a method all with the same name, but each with a different parameter list.
- At the time of method call, java compiler will know which one you mean by the number and/or types of the parameters.
- Overloaded methods must differ in the type and/or number of their parameters.
- Implements polymorphism
- May have different return types.

 To overload a method, you just define it more than once, specifying a new parameter list different from every other

Overloading Methods example

```
Class calculator
{
   int add(int op1, int op2)
   {
      return op1+op2;
   }
   int add(int op1, int op2, op3)
   {
      return op1+op2+op3;
   }
}
```

- *In C you have three functions to get the absolute value of different data types– **abs()** for integer, **fabs()** for floating point, and **lfabs()** for long integer.*

Overloading Constructors

- Works like overloading other methods.
- Define the constructor a number of times, each time with a different parameter list.

```
//when all dimensions specified
Box(double w, double h, double d){
 Width = w;
 Height = h;
 Depth = d;
}
//when cube is created
Box(double l){
 Width = Height =  Depth = l;
}
```

```
Box myBox = new Box(10,20,15);
Box yourBox = new Box(25);
```

By: RAJU PAL

Using Objects as Parameters

- An object of a class can be passed as parameter to both methods and constructors of a class.

```
/*construct a new object so that it is initially the same
  as some existing object.
Define a new constructor Box that takes an object of its
  class as a parameter.
*/
Box(Box ob){
  Width = ob.Width;
  Height = ob.Height;
  Depth = ob.Depth;
}
Box myBox = new Box(10,20,15);
Box yourBox = new Box(myBox);
```

Using Objects as Parameters cont…

```
//objects may be passed to methods
class Test{
int a,b;
Test(int I, int j){
 a = i;
 b = j;
 }
 //return true if obj is equal to the invoking obbject
 boolean equals(Test obj){
   if(obj.a == a && obj.b == b) return true;
   else return false;
 }
}
```

Test ob1 = new Test(100, 22);
Test ob2 = new Test(100, 22);
Test ob3 = new Test(-1,-1);
ob1.equals(ob2); true
Ob1.equals(ob3); false

By: RAJU PAL

Type of Argument Passing

- ***Call-by-value:***
  - When you pass an item of a simple data type to a method.
  - Method only gets a copy of the data item.
  - The code in the method cannot affect the original data item at all.

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

```
Test ob = new Test();
int a = 15, b = 20;
Systeem.out.println(" a and b before call: " +a+" "+b);  15  20
ob.meth(a,b);
Systeem.out.println(" a and b after call: " +a+" "+b);  15  20
```

Type of Argument Passing cont…

- ***Call-by-reference:***
  - When you pass an object to a method.
  - Java actually passes a reference to the object.
  - Code in the method can reach the original object.
  - Any change made to the passed object affect the original object.

```
class Test{
  int a,b;
  Test(int i, int j){
  a = i;
  b = j;
 }
  void meth(Test o){
     o.a = o.a*2;
     o.b = o.b/2;
  }
}
Test ob = new Test(15,20);
Systeem.out.println(" a and b before call: " +a+" "+b);  15  20
ob.meth(ob);
Systeem.out.println(" a and b after call: " +a+" "+b);   30  10
```

Returning Objects from Methods

- A method can return objects just like other data types.
- The object created by a method will continue to exist as long as there is a reference to it.
- No need to worry about an object going *out-of-scope* because the method in which it was created terminates.

Returning Objects Example

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

ob1.a = 2;
ob2.a = 12;

By: RAJU PAL

Visibility Modifiers Accessor Methods

- Visibility modifiers specify which parts of the program may see and use any particular class/method/field.
- How a member can be accessed is determined by the *access specifier* that modifies its declaration.
- Java has three visibility modifiers: **public**, **private**, and **protected.**

 *When no access specifier is used , then by default the member of a class is public within its own package but can not be accessed outside its package.**(default visibility)***

Visibility Modifiers - Classes

- A class can be defined either with the **public** modifier or without a visibility modifier **(default visibility).**
- If a class is declared as public it can be used by any other class
- If a class is declared without a visibility modifier it has a default visibility.

Visibility Modifiers - Members

- A member is a *field*, a *method* or a *constructor* of the class.
- Members of a class can be declared as **private**, **protected**, **public** or without a visibility modifier **(default)**:

```
private int hours;
int hours;
public int hours;
```

Public Visibility

- Members that are declared as public can be accessed from any class that can access the class of the member
- We expose methods that are part of the interface of the class by declaring them as public
- We do not want to reveal the internal representation of the object's data. So we usually do not declare its state variables as public (encapsulation)

Private Visibility

- A class member that is declared as private, can be accessed only by code that is within the class of this member.
- We hide the internal implementation of the class by declaring its state variables and auxiliary methods as private.
- Data hiding is essential for encapsulation.

The *static* Keyword

- Create a member that can be used by itself, without reference to a specific instance or object.
- A static member can be accessed before any objects of its class are created.
- You can declare both methods and variables to be **static.**
- These variables are, essentially, global variables.
- No copy of a **static** variable is made when objects of its class are declared.
- All instances of the class share the same **static** variables.

By: RAJU PAL

static Methods

✔ Methods declared as static have several restrictions:

- They can only call other **static** methods.
- They must only access **static** data.
- They cannot refer to **this** or **super** in any way.

final Keyword

- A variable can be declared as **final.**
- contents of the variable cannot can not be modified.
- Must initialize a **final** variable when it is declared.

```
Final int a=10;
Final flaot = 10.45f
```

Nested and Inner Classes

- Define a class within another class, known as nested class.
- Scope is bounded by its enclosing class.
- Nested class has access to the member of its enclosing class including private members but not reverse.
- Two types of nested classes:
  - **static** nested class: *can't access the member of its enclosing class directly.*
  - **Non-static** nested class **(Inner class):** *have direct access to its enclosing class*

Inner Class Example

```
class A{
  int var_a = 100;
  void abc(){
    B obj = B()
    obj.display();
  }
class B{
  void display(){
    System.out.println("The value of var_a: "+var_a);
  }
}
}
Class C{
  Public static void man(String args[]){
  A obj_outer = new A();
  obj_outer.abc();
  }
}
```

By: RAJU PAL

Thank you !!

By: RAJU PAL