CS205 Object Oriented Programming in Java

Module 2 - **Core Java Fundamentals (Part 6)**

Prepared by Renetha J.B.

Dept.of CSE, LMCST

Topics

- Core Java Fundamentals:

✓ **Constructors**

✓ **this Keyword**

✓ **Method Overloading**

✓ **Using Objects as Parameters**

Constructor

- A constructor **help to initialize an object**(give values) immediately upon creation.
- Constructor is a special method inside the class.
- Constructor has the same name as the class in which it resides.
- Once defined, the constructor is automatically called immediately after the object is created, before the new operator completes.

Constructor(contd.)

- Constructors have no return type, not even void.
  - This is because the implicit return type of a class' constructor is the class type itself.
- Two types of constructors
  - Default constructor – has no arguments
  - Parameterized constructor –has arguments(parameters)

Constructor(contd.)

- **Default constructor** has no arguments or parameters.

E.g.

```
class A
{
    A( )
    {
    //statements
    }
}
```

Default Constructor of class A

Default constructor(contd.)

```
class Box
{
int width ,length,height;
Box()
{
width=10;
length=10;
height=10;
}}
```

The following statement creates an object of class Box.

Box mybox1 = **new Box();**

Here **new Box( )** is calling the **Box( )** constructor**.**

Default constructor(contd.)

```
class Box
{
int width ,length,height;
Box()
{
width=10;
length=10;
height=10;
}}
```

The following statement creates an object of class Box.

```
Box mybox1 = new Box();
```

Here **new** **Box( )** is calling the **Box( )** constructor**.**

Default constructor(contd.)

- When we do not explicitly define a constructor for a class, then **Java creates a default constructor for the class.**

```
class Box {
    int width;
    int length;
    int height;
    Box()
    {
    System.out.println("Constructing Box");
    width = 10;
    length = 10;
    height= 10;
    }
    int volume()
    {
    return width * length * height;
    }
}
```

```
class Box {
        int length;
        int height;
        int width;
        Box()
        {
        System.out.println(“Constructor");
        width = 10;
        length = 10;
        height= 10;
        }
        int volume()
        {
        return width * length * height;
        }
}
```

```
class BoxDemo {
public static void main(String args[])
{
Box mybox1 = new Box();
Box mybox2 = new Box();
int vol;
vol = mybox1.volume();
System.out.println("Volume is " + vol);

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

OUTPUT

```
Constructor
Constructor
Volume is 1000
Volume is 1000
```

Parameterized Constructors

- Constructors with arguments are called parameterized constructors.

```
class Box
{
double width;
double height;
double length;
Box(double w, double h, double l)
{
width = w;
height = h;
length= l;
}
double volume()
{
return width * height * length;
}
}
```

Parameterized Constructor of class Box (Box constructor has arguments-> parameters)

```
class Box
{
double width;
double height;
double length;
Box(double w, double h, double l)
{
width = w;
height = h;
length = l;
}

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

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

Box mybox1 = new Box(10, 20, 15);
Box mybox2 = new Box(3, 6, 2);
double vol;
vol = mybox1.volume();
System.out.println("Volume is " + vol);

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

```
OUTPUT
Volume is 3000
Volume is 36
```

Parameterized constructor(contd.)

```
Box mybox1 = new Box(10, 20, 15);
```

- Here the values 10, 20, and 15 are passed to the **Box( )** constructor when new creates the object mybox1.
- The parameterized constructor is

```
Box(double w, double h, double l)
{
width = w;
height = h;
length = l;
}
```

- Thus, value of mybox1 object’s width, height, and depth will be set as 10, 20, and 15 respectively.

The this Keyword

- The **this** keyword can be used inside any method to refer to the **current object**.
- **this** is always a reference to the object on which the method was invoked.
- **this** can be used to refer current class instance variable.
- **this** can be used to invoke current class method (implicitly)
- **this**() can be used to invoke current class constructor.
- **this** can be passed as an argument in the method call.
- **this** can be passed as argument in the constructor call.
- **static methods** cannot refer to **this.**

this-Example

```
Box(double w, double h, double l)
 {
this.width = w;
this.height = h;
this.length = l;
}
```

Here **this** will always refer to the object invoking the method

```
class Box
{
double width;
double length;
double height;
Box(double w, double l, double h)
{
this.width = w;
this.length = l;
this.height = h;
}
}
```

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

Box mybox1 = new Box(10, 20, 15);
Box mybox2 = new Box(3, 6, 2);
}
}
```

Here in statement
Box mybox1 = new Box(10, 20, 15);
mybox1 object is created by calling parameterized constructor.
Box(double w, double l, double d)
Here **this** inside constructor refers to object mybox1.

Next when **mybox2** object is created, **this** refers to object mybox2.

Instance variable hiding-using this

- We can have **local variables**, including formal parameters to methods, which has the same name of the class' **instance variables(attributes**).
- But when a local variable has the same name as an instance variable, **the local variable *hides the* instance variable.**
  - **this** helps to solve this. Use **this.** along with instance variables.

Prepared by Renetha J.B.

Instance variable hiding-using this (contd.)

- // Use this to resolve name-space collisions.

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

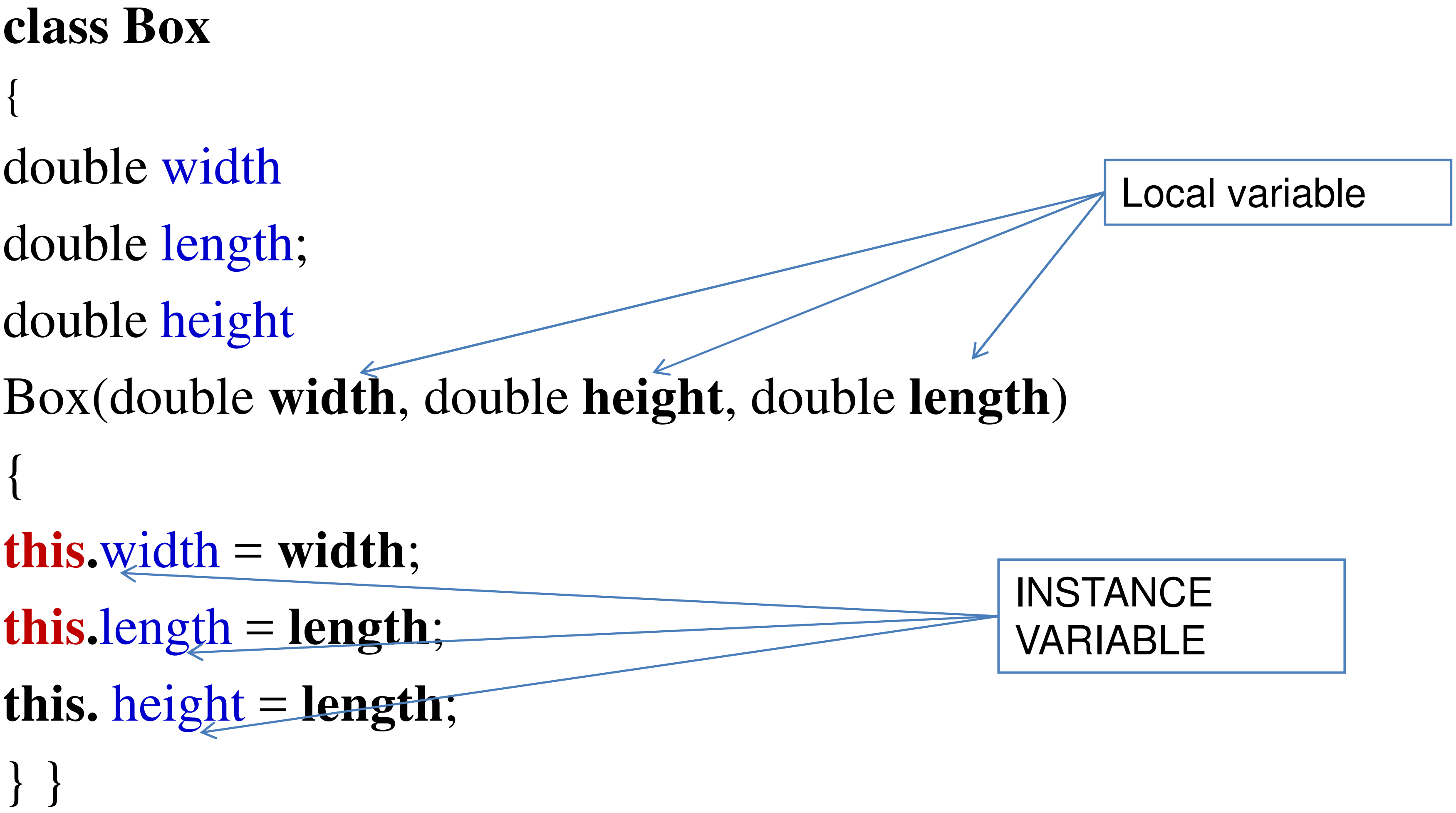
INSTANCE VARIABLE

CONSTRUCTOR

Prepared by Renetha J.B.

Instance variable hiding-using this (contd.)

- // Use **this** to resolve name-space collisions.



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

Local variable

INSTANCE VARIABLE

Method Overloading

- It is possible to define **two or more methods** with **same name** within the same class, but their parameter declarations should be different.
  - This is called **method overloading.**
  - This is a form of **polymorphism** (many forms)
- Overloaded methods must **differ in the type and/or number of their parameters.** (return types is not significant.)
- When an overloaded method is invoked, Java uses the type and/or number of arguments to determine which version of the overloaded method to actually call.

```
// Demonstrate method
    overloading.
class Over
{
void test()
{
    System.out.println(“Empty");
}
void test(int a) {
    System.out.println("a: " + a);
}
void test(int a, int b) {
    System.out.println(“a=”+a);
    System.out.println(“b=”+b);
    }
}
```

```
class Sample {
public static void main(String args[])
{
Over ob = new Over();

ob.test();
ob.test(10);
ob.test(2, 5);


}
}
```

OUTPUT

```
Empty
a=10
a=2
a=5
```

- *In the example* ,test( ) is overloaded three times.
  - The first version test() takes no parameters,
  - the second **test(int a)**takes one integer parameter
  - the thrd **test(int a,int b)** takes two integer parameters.

Method Overloading(contd.)

- When an overloaded method is called**, Java looks for a match between the arguments** used to call the method and the method’s parameters
- This **match need not always be exact**.
  - In some cases, Java’s automatic type conversions can play a role in overload resolution.

Overloading -through automatic type conversions

```
class Over{
void test() {
System.out.println(“Empty");
}

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

```
class Sample {
public static void main(String
    args[])
{
Over ob = new Over();


ob.test();
ob.test(10);
ob.test(2.5);


}
}
```

OUTPUT

```
Empty
a=10
a=2.5
```

Overloading -through automatic type conversions(contd.)

- In this example when **test( ) is called with an integer argument** inside .
  - Overload, no matching method is found with int as argument**.**
- However, Java can automatically **convert an integer into a double,** and this conversion can be used to resolve the call.
  - Therefore, when **test(int) is not found,** Java elevates int to double and then **calls test(double).**

Overloading Constructors

- Constructors can be overloaded. Because a class can have any number of constructors
  - one default constructor, many parameterized constructors

```
class A
{
A() { //statements}
A(int a) { //statements}
A(int a,float b) { //statements}


}
```

```
class Box
{
double width;
double length;
double height;
Box(double w, double l, double h)
{
width = w;
length = l;
height = h;
}
Box()
{
width = 0;
length =0;
height =0;
} }
```

```
class BoxDemo {
public static void main(String args[]) {
Box mybox1 = new Box();
Box mybox2 = new Box(3, 6, 2);
System.out.println("mybox1");
System.out.println(mybox1 .width + " “
    +mybox1 .length + " "+  mybox1 .height);

System.out.println("mybox2");
System.out.println(mybox2.width + " “ +
    mybox2.length + " " +  mybox2 .height);
} }
```

OUTPUT

```
mybox1
0.0   0.0   0.0
mybox2
3.0   6.0   2.0
```

```
class Box
{
double width
double length;
double height;
Box(double w, double l, double h)
{
this.width = w;
this.length = l;
this.height = h;
}
}
```

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

Box mybox1 = new Box(); //ERROR
Box mybox2 = new Box(3, 6, 2);
}
}
```

ERROR

Here following statement tries to create object mybox1 of class Box ,
Box mybox1 = new Box();
This should call default constructor **Box()** in class Box.
But Box class has constructor but no default constructor is there.

So ERROR occurs

```
class Box
{
double width
double length;
double height;
}
```

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

Box mybox1 = new Box();
}
}
```

NO ERROR in this code
The following statement creates object of Box class mybox1
Box mybox1 = new Box();
Since no constructors are not there,
Java provides the default constructor.

Using Objects as Parameters

- We can pas objects as arguments(parameters) to function(method).
- Objects are **passed by reference(call by refernce).**

Object as parameters

```
        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.equals(ob2));
System.out.println(ob1.equals(ob3));
}}
```

OUTPUT

```
true
false
```

Object as parameters

```
class Test {
int a, b;
Test(int i, int j)
{
a = i;
b = j;
}
boolean equals(Test o)
 {
if(o.a == this.a && o.b == this.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.equals(ob2));
System.out.println(ob1.equals(ob3));
}}
```

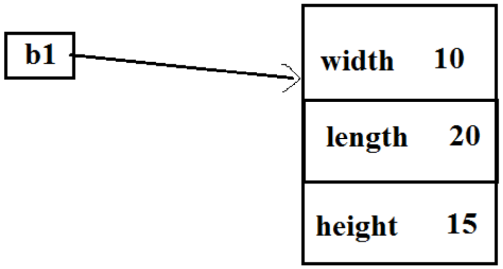
OUTPUT

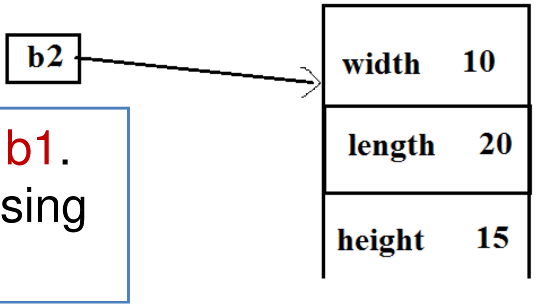
```
true
false
```

Object to initialize another object

```
class Box
{
double width
double length;
double height;
Box(double w, double l, double h)
{
width = w;
length = l;
height = h;
}
}
```

```
class BoxDemo {
public static void main(String args[])
{
Box b1 = new Box(10, 20, 15);
Box b2 = new Box(b1);
}
}
```





Here object b2 is a clone of b1. The object b2 is initialized using initial values of object b1

Passing arguments to function

- // Primitive types(int,char,double etc.) are passed by value.
- // Objects are **passed by reference.**

```
class Test {
int a;
Test(int i)
{
a = i;
}
void calc(Test o)
{
o.a *= 2;
}
void calc(int a)
{
a*=2;
}
}
```

```
class Obcall {
public static void main(String args[])
{
Test ob = new Test(15);
System.out.println("Object parameter");
System.out.println("Before call: " + ob.a );
ob.calc(ob);    // //Call by reference
System.out.println("After call: " + ob.a );


int a=15;
System.out.println("Integer parameter");
System.out.println("Before call: " + a);
ob.calc(a);      //Call by value
System.out.println("After call: " + a);  } }
```

```
OUTPUT
Object parameter
Before call: 15
After call: 30
Integer parameter
Before call: 15
After call: 15
```

Reference

- Herbert Schildt, Java: The Complete Reference, 8/e, Tata McGraw Hill, 2011.