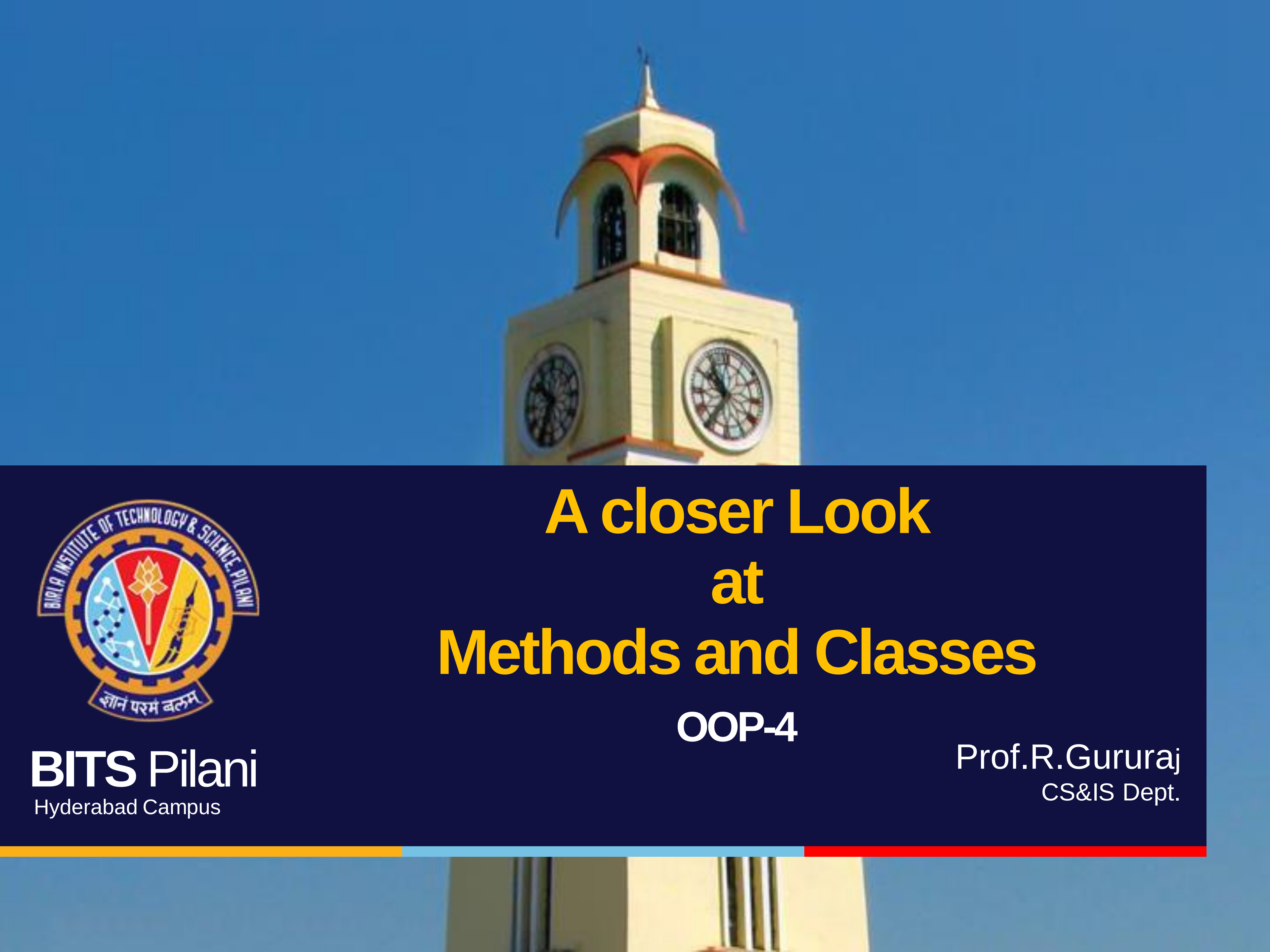
CS F213
Object Oriented Programming

BITS Pilani
Hyderabad Campus

Prof.R.Gururaj
CS&IS Dept.



BITS Pilani
Hyderabad Campus

A closer Look at Methods and Classes

OOP-4

Prof.R.Gururaj
CS&IS Dept.

Ch.6 & 7 of T2.

The Complete Reference- Java, 11th Edition, Herbert Schildt, Tata McGraw Hill Publishing.

And also refer to Class notes.

Classes

- The class forms the basis for object oriented programming in Java.

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

General form of a class:

```
class classname
{
    type instance-variable1;
    type instance-variable2;
    ....
    type instance-variableN;
    type methodname1(parameter-list)
    { body }
    ......
    type methodnameN(parameter-list)
    { body }
}
```

Members of a class

instance variables

methods

A simple class:

```
class Box
{
        int width;
        int length;
        int depth;
}
```

To instantiate an object of Box

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

To access variables of an object

```
        myBox.length=12;
```

Hungarian notation

(Charles Simonyi from Microsoft)

Naming convention used in Java OOP

Class names - start with caps and and new words will start with caps

Ex: Box, BoxDemo

Variables – start with small case and new words will start with caps

Ex: length, boxVolume

methods – start with small case and new words will start with caps

Ex: volume(), boxVolume(), findSum()

A program that uses Box class

```
class Box
{
      int width; int length;  int depth;
}
class BoxDemo
{
      public static void main(String args[])
      {
                  Box myBox=new Box();
                  int volume;
                  myBox.length=2; myBox.width=3; myBox.height=3;
                  volume= myBox.length *  myBox.width *  myBox.height;
                  System.out.println(“Box volume is : “+volume);
      }
 }
```

A program that instantiates more than one Box class objects

```
class BoxDemo
{
        public static void main(String args[])
        {
                  Box myBox1=new Box();
                  Box myBox2=new Box();
                  int volume;
                 myBox1.length=2; myBox1.width=3; myBox1.height=3;
                 volume= myBox1.length *  myBox1.width *  myBox1.height;
                 System.out.println(“Box1  volume is : “+volume);
                 myBox2.length=5; myBox2.width=6; myBox2.height=7;
                 volume= myBox2.length *  myBox2.width *  myBox2.height;
                 System.out.println(“Box2  volume is : “+volume);
        }
   }
```

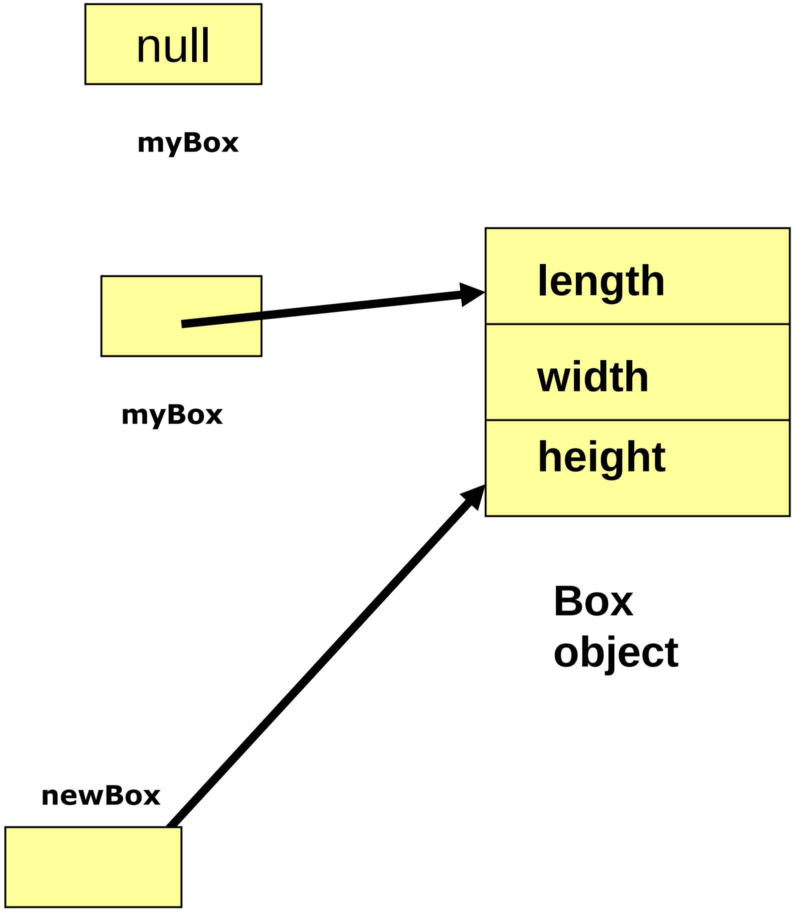
Declaring class objects

```
Box myBox;

myBox=new Box();

Box newBox;

newBox= myBox;
```



Adding methods

```
type methodname1(parameter-list)
{
        body
 }
```

Type is return value type and could be *void*

We return values from a method using the syntax

return *value*;

We can return

- any java predefined type value or an object.

- the type specified in return type of method declaration and actual return value type must match.

Adding a method to Box class that returns some value:

```
class Box
{
      int width;
      int length;
      int depth;
      int volume()
      {
         return (length *  width * height);
      }
}
```

Adding a method to Box class that take parameters:

```
class Box
{
        int width;
        int length;
        int depth;
        int volume()
        {   return (length *  width * height);   }
        void setDimensions(int l, int w, int h)
        {
                        length=l;
                        width=w;
                        height=h;
        }
}
```

Constructors

- ❑ Initialize java objects when they are created

  they have no return type.
- ❑ Constructor is automatically called after object is created and new operator completes.
- ❑ A default constructor is created by java, instance variables will be initialized to zero or null.
- ❑ If we explicitly define a constructor the default constructor is no more available.

```
//Class Box with default constructors
class Box
{
      int width;
      int length;
      int depth;
      int volume()
      {
         return (length *  width * height);
      }
}
```

A program that uses Box class constructor

```
class BoxDemo
{    Public static void main(String args[])
     {
               Box myBox1=new Box();
                int vol;
               vol= myBox1.volume();
               System.out.println("Box volume is : " + vol);
      }
}
```

Adding a parameterized constructor to Box class :

```
class Box
{
     int width;  int length;    int depth;
     Box(int l, int w, int h)
     {System.out.println(“Inside box parameterized constructor  : " );
               length=l;
               width=w;
               height=h;
     }
     int volume()
     {    return (length *  width * height);    }
}
```

A program that uses parameterized constructor

```
class BoxDemo
{
     Public static void main(String args[])
     {          Box myBox1=new Box(2,3,4);
                 int vol;
                vol= myBox1.volume();
                System.out.println(“Box volume is : “ + vol);
       }
}
```

```
Inside box parameterized constructor:
Box volume is :24
```

Use of *this* keyword

The keyword *this* used in a method to refer to the object invoked it.

```
class Box
{
      int width; int length; int height;
      Box(int w, int l, int h)
      {System.out.println("Inside box parameterized constructor  : " );
                  this.length=l;
                  this.width=w;
                  this.height=h;
      }
}
```

But the above usage is redundant

Instance variable hiding

When local variable and instance variable names are same, then local variable hides the instance variable.

```
class Box
{
     int width; int length; int height;
     Box(int length, int width, int height)
     {System.out.println("Inside box parameterized constructor  : " );
               length=length;
               width=width;
               height=height;
     }
}
```

Instance variable hiding

When local variable and instance variable names are same, then local variable hides the instance variable. The keyword ‘this’ will resolve the problem

```
class Box
{
     int width; int length; int height;
     Box(int length, int width, int height)
     {System.out.println(“Inside box parameterized constructor  : “ );
               this.length=length;
               this.width=width;
               this.height=height;
     }
}
```

```
class Line
{
    int  length;
    Line()
    {
    System.out.println("Inside Line no argument constructor  : " );
    length=12;
    }
    void print()
    {
    int length=30;
    System.out.println("Line length is : "+length );
    }
}
```

```
class Demo
{
        public static void main(String args[])
        {
        Line l1=new Line();
        System.out.println(" Line length is : "+ l1.length );
        l1.print();
        }
}
//what is the output?
```

```
class Line
{
    int  length;
    Line()
    {
    System.out.println("Inside Line no argument constructor  : " );
    length=12;
    }
    void print()
    {
    int length=30;
    System.out.println("Line length is : "+ this.length );
    }
}
```

Garbage Collection in Java

- In Java we don’t destroy objects that are created.
- Java has a mechanism called ‘***garbage collection***’.
- It is daemon thread.
- It reclaims space occupied by unused objects.

Daemon thread in java

Garbage collector thread is **a daemon thread**. Daemon thread is a low priority thread which runs intermittently in the back ground doing the garbage collection operation or other requests for the java runtime system.

Garbage collection in Java:

When no references to an object exist, it is assumed that it is no longer needed, and the memory occupied by that object can be reclaimed. No explicit need to destroy it, like in C++. It happens sporadically.

finalization mechanism in Java

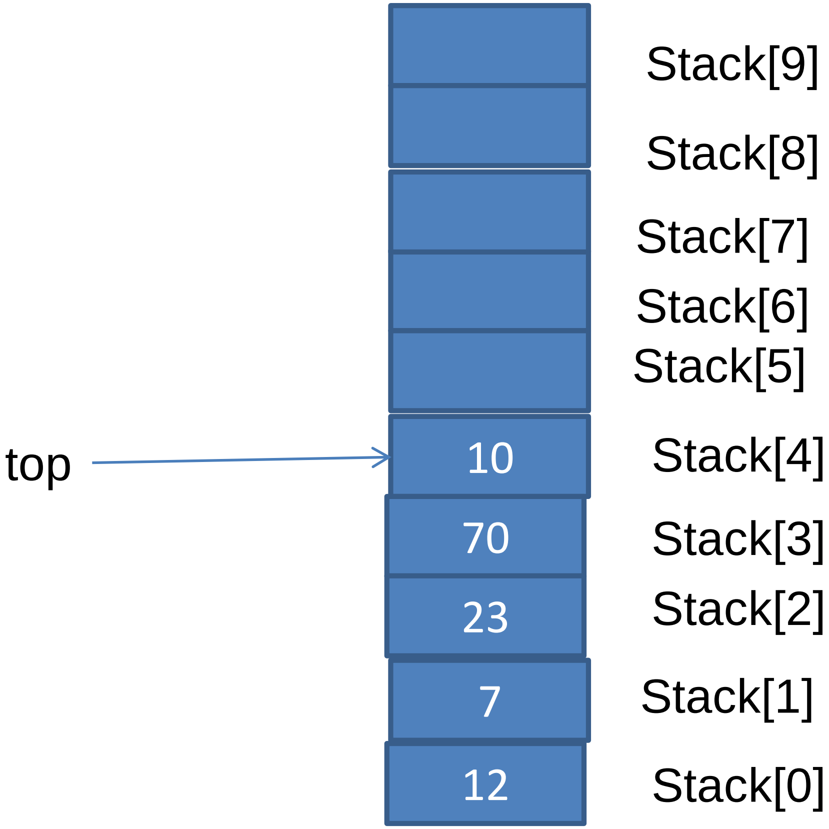
it facilitates performing certain actions just before an object is destroyed by garbage collector.

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

```
     class Box
     {
     public void finalize()
      {System.out.println("object is garbage collected");}
     }
public class Demo
{
    public static void main(String args[]){
            Box b1=new Box();
             b1=null;
            System.gc();
    }
}
// but execution of gc() anf finalize() are not guaranteed.
```

Stack Example

A stack stores data using first-in, last-out ordering. Stacks are controlled through two operations pop and push.



int stack[ ]=new int[10];
tos=-1;

Implementation as discussed in the Lect session.

Method and constructor with same name?

```
class LineDemo
{
        public static void main(String args[])
        {
        Line s=new Line ();
        System.out.println(" Line Length is:"+s.l);
        s. Line(23);
        }
}
class Line
{
        int l;
        Line()
        {            System.out.println(" In constructor");       }
        void Line(int a)
        {System.out.println(" Inside Line method : "+a);}
}
        //output
In constructor
 Line Length is:0
 In Line method : 23
```

Summary

- ❑ Class structure
- ❑ Methods
- ❑ Constructors
- ❑ Parameterized constructor
- ❑ Use of *this* keyword
- ❑ Garbage collection
- ❑ Method *finalize()*
- ❑ Stack Example