UNIT I INTRODUCTION TO OOP AND JAVA FUNDAMENTALS 10

Object Oriented Programming - Abstraction - objects and classes - EncapsulationInheritance - Polymorphism- OOP in Java - Characteristics of Java - The Java Environment **-** Java Source File - Structure - Compilation. Fundamental Programming Structures in Java - Defining classes in Java - constructors, methods -access specifiers - static members -Comments, Data Types, Variables, Operators, Control Flow, Arrays , Packages - JavaDoc comments.

1.4 FUNDAMENTAL PROGRAMMING STRUCTURES IN JAVA

1.4.1.OBJECTS AND CLASSES

Defining a Class

- Class is a user defined data type with a model serves to define its properties.
- Once the class type is defined we can make variables of that type using declarations.
- These variables are known as instances of classes, which are the real objects.

```
class Classname [extends superclass name]
{
[Varibale declaration;]
[Methods
declaration;]

}
```

Adding Variables

- Data is wrapped in a class by placing data memberswithin the body of the class definition.
- Variables are called instance variables for the reason that they are created every time an object of the class is instantiated.

Example:

```
class Rectangle
{
int length;
int width;
}
```

Adding Methods

- Functions are declared within the body of the class but directly after the declaration of instance variables.

```
Type method name (parameter list)
{
Method-body;
}
```

Method declarations have 4 basic parts

1. Name of the method(**method name**)
2. Type of value the method returns(**type**)
3. List of parameters(**parameter list**)
4. The body of the method

Example:

```
class Rectangle
{
int length;
int width;
void getData(int x,int y)
{
length =x;
width =y;
}
}
```

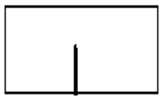
Creating objects

- In Java programming objects are created by the new operator
- New operator creates an object of the precise class and returns a reference to that object.

Ex:

Rectangle rect1;//**declare**
Rect1=new Rectangle();// **instantiate**

First statement declares a variable to have the object reference & the second one really assigns the object reference to the variable.

Action	**Statement**	**Result**
Declare	Rectangle recti;	**null** recti
Instantiate	recti=new Rectangle();	rect1  **Rectangle Object**

• recti is the reference

Accessing Class Members

- Objects have been created, each containing its hold set of variables, and we should allocate values to these variables.
- All variables must be allocated values before they are used.
- We are outside the class; we cannot right to use the instance variables & the functions directly.

Objectname.Variablename=value;
Objectname.methodname(parameter list);

Ex:

```
recti.length=i5;
rect2.getData(20,i0);
```

Example

```
class Rectangle
{
int length;//Declaration of varibles
void getData(int x,int y)//Definition of method
{
length=x;
width=y;
}
int rectArea()//Definition of another method
{
int area=length*width;
return(area);
}
} class RectArea
```

```
{
Public static vod main(String args[])
{
int area1,area2;
Rectangle rect1=new rectangle();//Objects
Rectangle rect2=new Rectangle();// Objects

rect1.length=15;
rect1.width=10;

area1=rect1.length*rect1.width;

rect2.getData(20,12);
area2=rect2.rectArea();

System.outprintln("Area1="+area1);
System.outprintln("Area2="+area2);
}
}
```

1.4.2.CONSTRUCTORS
Java supports a special kind of the function, called "constructor", that enables an object to initialize itself when object is created.Constructors have the similar name as the class itself. They don't have a return type,void is also not used as return type.

Example1:Constructor

```
class Rectangle
{
int length;
int width;

Rectangle(int x,int y)//constructor method

{
length=x;
width=y;
}
int rectArea()
{
return(length*width);

class RectangleArea
{
public static void main (String args[])
{
Rectangle rect1=new Rectangle(15,10);//calling constructor
int area1=rect1.rectArea();
System.outprmtln("Area =" +area1);
}
}
```

Example 2: (default Constructor and Parameterized Constructor)
Default constructor will not have any parameters where as parameterized constructor can have one or more parameters. copy constructor is a special kind of constructor where old object is passed as a argument for new object.

```
class complex
{
double real,imag;
complex() //simple or default constructor
{
real=0.0;
imag=0.0;
}
complex(double x,double y) //parameterized constructor
{
real=x;
imag=y;
}
complex add(complex a) //copy constructor
{
complex n=new complex();
n.real=this.real+a.real;
n.imag=this.imag+a.imag;
return n;
}
void display()
{
System.outprmtln("The result is"+real+"+i"+imag);
}
}
class comclass
{
public static void main(String args[])
{
complex c1=new complex(10,20);
complex c2=new complex(10,20);
complex c1=new complex();
c3=c1.add(c2);
c3.display();
}
}
```

Output:

```
The result is 20.0+i40.0
```

1.4.3. Methods

Methods are also called as functions or sub programs.The syntax for method declaration is given below

```
dataType methodname(parameter lists or argument lists)
{
// body of method
}
```

- dataType denotes the type of data returned by the function. This can be any valid data type, including class types that you have created.
- If the function does not return a value, its return type should be void.
- The name of the method is specified by name.
- The parameter-list is a sequence of data type and identifier pairs specified by commas.
- Parameters are essentially variables that obtain the value of the arguments passed to the function when it is called.
- If the function has no parameters, then the argument list will be empty.
- Functions that have a return type other than void should return a value to the calling routine with the following form of the return statement:

return value;

Here, return is a java keyword and value is the value returned.

Adding a Method to the Class

Use functions to call the instance variables defined by the class.

In fact, functions define the interface to the majority classes. This permits the class implementer to conceal the specific layout of inner data structures behind cleaner function abstractions.

Example Program : Methods

```
// This program includes a method inside the box class.
class Box
{
double width;
double height;
double depth;
// display volume of a box
void volume()
{
System.out.print("Volume is ");
System.out.println(width * height * depth);
}
}
class BoxDemo3
{
public static void main(String args[])
{
Box mybox1 = new Box();
Box mybox2 = new Box();
// assign values to mybox1's instance variables
mybox1.width = 10;
mybox1.height = 20;
mybox1.depth = 15;
/* assign different values to mybox2's instance variables */
mybox2.width = 3;
mybox2.height = 6;
mybox2.depth = 9; 14
// display volume of first box
mybox1.volume();
// display volume of second box
mybox2.volume();
}
}
```

Output:

```
Volume is 3000.0
Volume is 162.0
```

Returning a Value

A best way to implement volume( ) is to have it calculate the volume of the box and return the outcome to the caller.

```
// Now, volume() returns the volume of a box.
class Box
{
double width;
double height;
double depth;
// compute and return volume
double volume()
{
return width * height * depth;
}
}

class BoxDemo4
{
public static void main(String args[])
{
Box mybox1 = new Box();
Box mybox2 = new Box();
double vol;
// assign values to mybox1's instance variables
mybox1.width = 10;
mybox1.height = 20;
mybox1.depth = 15;
/* assign different values to mybox2's instance variables */
mybox2.width = 3;
mybox2.height = 6;
mybox2.depth = 9;
// get volume of first box
vol = mybox1.volume();
System.out.println("Volume is " + vol);
// get volume of second box
vol = mybox2.volume();
System.out.println("Volume is " + vol);
}
}
```

Adding a Method That Takes Parameters

Parameters allow a method to be generalized. 15

This program uses a parameterized method.

```
class Box
{
double width;
double height;
double depth;
// compute and return volume
double volume()
{
return width * height * depth;
}
// sets dimensions of box
void setDim(double w, double h, double d) {
width = w;
height = h;
depth = d;
}
}
class BoxDemo5
{
public static void main(String args[])
{
Box mybox1 = new Box();
Box mybox2 = new Box();
double vol;
// initialize each box
mybox1.setDim(10, 20, 15);
mybox2.setDim(3, 6, 9);
// get volume of first box
vol = mybox1.volume();
System.out.println("Volume is " + vol);
// get volume of second box
vol = mybox2.volume();
System.out.println("Volume is " + vol);
}
}
```

As you can see, the setDim( ) function is used to put the dimensions of each box. For- example, when
mybox1. setDim(10, 20, 15);

1.4.4. ACCESS SPECIFIERS

Visibility modifiers are also referred as access modifiers.

Three Types

1. Public
2. Private
3. Protected

Public Access

- Any variable or function is visible to the full class in which it is defined

  public int number;
  public void sumQ {..........}
- A variable or function declared as public has the feasible visibility & accessible in all places.

Protected Access

- The protected modifier makes the members visible not only to all classess and derived classes in the same package but also to derived class in other packages.

Private Access

- They are accessible only inside their own class
- They cannot be inherited by derived classes and not accessible in derived classes

Private Protected Access
Ex: private protected int codenumber;

- This modifier makes the field visible in all subclasses regardless of what package they are in.
- These members are not accessible by other classes in the same package.

	Public	**Protected**	**Private Protected**	**Private**
Same Class	Yes	Yes	Yes	Yes
Sub class in same package	Yes	Yes	Yes	No
Other class in same package	Yes	Yes	No	No
Subclass in other package	Yes	Yes	Yes	No
Non subclasses in other packages	Yes	No	No	No

1.4.5.STATIC MEMBERS

- Class consists of 2 sections:
  - Declare variables (instance variables)
  - Declare methods (instance methods)
- This is because each time the class is instantiated a new replica of each of them is created
- They can be accessed using the objects.
- Assume define a field that is regular to all the objects & accessed without using a specific object

```
static int count;
static int max(int x,int y);
```

- These members are called as static members.
- Static methods are called using class members

Static methods have several restrictions

1. Static method can only call other static methods
2. Static method can only access static data
3. Static method cannot refer to “this” or “super” in any way.

Finalize Method

- finalize()-added to any class
- Java calls that method whenever it is about to get back the space for that object
- The finalize method should openly define the tasks to be performed.

Example

```
class Mathoperation
{
static float mul(float x,float y) //static method
{
return x*y;
}
static float divide(float x,float y) //static method
{
return x/y;
}
}
class Mathapplication
{
public static void main(String args[])
{
float a=Mathoperation.mul(4.0,5.0); //static method mul is called by class name Mathoperation
float b=Mathoperation.divide(a,2.0); //staticmethod divide is called by class name
Mathoperation
System.outprmtln("b=”+b);
}
}
```

1.4.6. JAVA COMMENTS

The java comments are exactly statements which are never executed by the compiler and interpreter. The comments can be used to give information or details about the variable, function, class or any statement. It can also be used to wrap program code for exact time.

Types of Java Comments

There are 3 types of comments in java.

- Single Line Comment
- Multi Line Comment
- Documentation Comment

1 .Java Single Line Comment

The single line comment is used to comment only one line.

Syntax:

```
//This is single line comment
```

Example:

```
public class CommentExample1 {
public static void main(String[] args) {
   int i=10;//Here, i is a variable (Single line comment) System.out.println(i);
} }
```

Output: 10

2 .Java Multi Line Comment

The multi line comment is used to comment many lines of code.

Syntax:

```
/*
This is multi line comment */
```

Example:

```
public class CommentExample2 {
public static void main(String[] args) { /* Let's declare and print variable in java. */
   int i=10;
   System.out.println(i);
} }
```

Output: 10

3 .Java Documentation Comment

The documentation comment is used to make documentation API. To generate documentation API, you want to use **javadoc tool**.

Syntax:
```
/**
This is documentation comment */
```
Example:
```
/** The Calculator class provides
functions to get addition and subtraction of given 2 numbers.*/ public class Calculator {
/** The add() method returns addition of given numbers.*/ public static int add(int a, int
b){return a+b;}
/** The sub() method returns subtraction of given numbers.*/ public static int sub(int a, int
b){return a-b;} }
```

Compile it by javac tool:

javac Calculator.java

Create Documentation API by javadoc tool:

javadoc Calculator.java

Now, there will be HTML files formed for your Calculator class in the present directory. Open the HTML files and observe the details of Calculator class provided through documentation comment.