UNIT I INTRODUCTION TO OOP AND JAVA FUNDAMENTALS 10

Object Oriented Programming - Abstraction – objects and classes - Encapsulation- Inheritance - 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.

1.1. Object Oriented Programming (or) OOP in Java

Object Oriented Programming is a programming approachwhich is based on the objects and similar will be used to write the implementation. It allows programmers to create the objects andfunctions to perform those objects. Manipulating these objects to get outcome is the goal of Object Oriented Programming.

Object Oriented Programming popularly known as OOP, is used in a present programming language like Java

1.1.1.Abstraction

- ✓ It refers to the act of representing necessary features without including the background details/explanation.
- ✓ It is a method of creating a new data type that is matched for a specific application. For example, while driving a car, you do not have to be worried with its internal working. Here you just need to worry about parts like steering wheel, Gears, accelerator, etc.

1.1.2.Objects and Classes

(Example program Refer any class based program in unit 1 and 2)

- ✓ Objects are real world entities in an object oriented system.
  Ex: person, place, bank account.

1. Instance of a class is called an Object, and there can be several instances of an object in a program. An Object contains both the data member and the function, which operates on the data. For example - chair, bike, marker, pen, table, car, etc.

Class:

- ✓ Collection of objects are called as objects
  Ex: Mango, apple & orange are member of the class fruits.

2. The class is a collection of similar entities. It is only logical component and not the physical unit. For example, if you had a class called "Expensive Cars" it could have objects similar to Mercedes, BMW, Toyota, etc. Its properties(data) can be price or speed of these cars. While the functions may be performed with these cars are driving, reverse, braking etc.

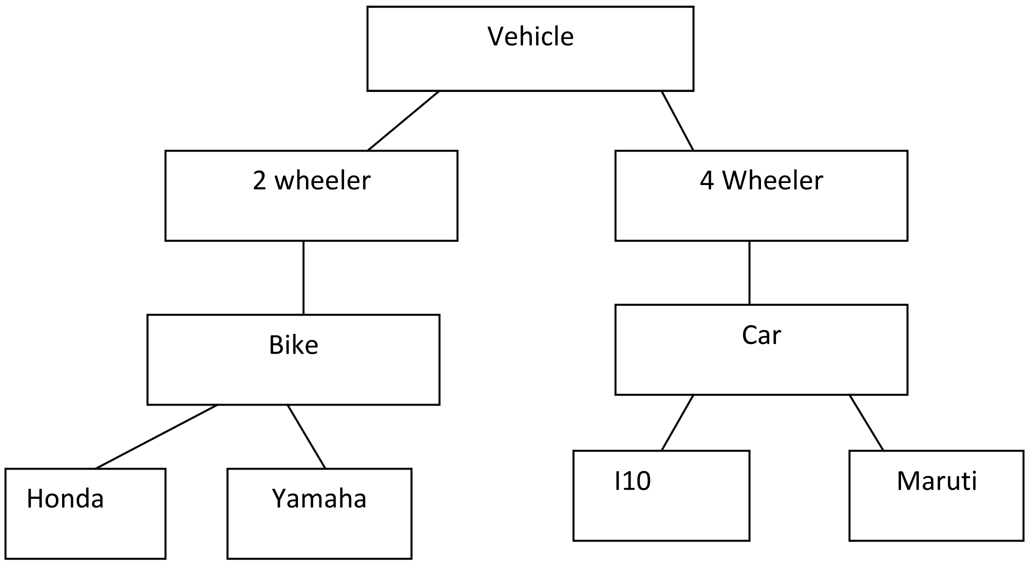
1.1.3. Encapsulation

- ✓ The wrapping up of data and functions in to a single unit is known as encapsulation.
- ✓ Encapsulation is an OOP method of wrapping the data and code. In this OOPS idea, the variables of a class are always concealed from other classes. It can only be accessed using the functions of their current class. For example - in school, a student cannot live without a class.

1.1.4. Inheritance

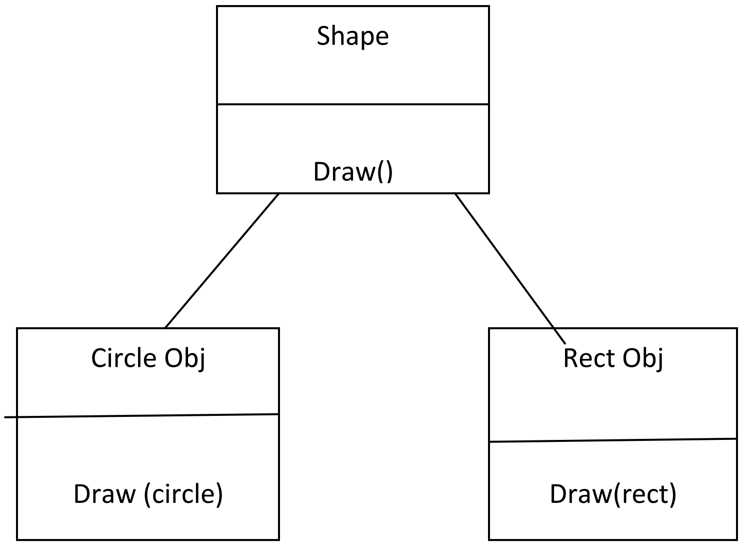
(Example program Refer unit-2)

- ✓ Objects of one class take the properties of objects of another class.
- ✓ Inheritance is an OOPS concept in which one object holds the properties and behaviors of the parent object. It's creating a parent-child relationship among two classes. It offers robust and natural method for organizing and structure of any software.
- ✓



1.1.5.Polymorphism

- ✓ The ability to take more than one form.
- ✓ Polymorphism refers to the capability of a variable, object or method to take on multiple forms. For example, in English, the verb "run" has a various meaning if you use it with "a laptop," "a foot race, and "business. Here, we recognize the meaning of "run" based on the other words used beside with it.



//compile time polymorphism
// Method overloading (add is overloading method –type, arguments and operations are different)

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}}
```

//Run time polymorphism example
// Method overriding (run is overriding method –type, arguments are same and operations is //different)

```
class Bike
{
 void run() //Method overriding
{
System.out.println("running");
}
}
class Splender extends Bike
{
 void run() //Method overriding
{
System.out.println("running safely with 60km");
}
```

```
 public static void main(String args[]){
   Bike b = new Splender();//upcasting
   b.run();
 }
}
```
Output: running safely with 60km.

1.1.6.Dynamic Binding

- ✓ The code related with a given method call is not known until the moment of call at runtime.

1.1.7.Message Communication

The process of programming in an OO language represented below
1.Creating classess that describe objects and their behavior.
2.Creating objects from class definitions.
Establishing communication among objects through message passing..

1.2. CHARACTERISTICS OF JAVA:

The various features of java programming are listed down below

Simple :

- Java is very comfortable to write and read.
- Java has a brief, cohesive set of concepts that makes it easy to learn and use.
- Most of the concepts are derived from C++ thus making Java learning process simple.

Secure :

- Java program cannot damage other system thus making it secure.
- Java presents a secure means of creating Internet applications.
- Java presents secure way to access web applications.

Portable :

- Java programs can execute in any system environment with the help of Java run-time system.(JVM-Java Virtual Machine)
- Java programs can be run on any platform (Linux,Window,Mac)
- Java programs can be used over world wide web (e.g applets)

Object-oriented :

- Java programming is object-oriented programming language.
- Like C++ java provides most of the object oriented concepts.
- Java is a complete OOP. Language. (while C++ is semi object oriented)

Robust :

- Java provides error-free programming by being severely typed and performing run-time checks.

Multithreaded :

- Java provides integrated support for multithreaded concepts.

Architecture-neutral :

- Java is not joined to a specific machine or operating system architecture.
- Machine Independent

Interpreted:

- Java provides cross-platform code through the use of Java bytecode.
- With the help of JVM,Bytecode can be interpreted on any platform

High performance:

- Bytecodes are extremely optimized.
- JVM can execute them much quicker.

Distributed :

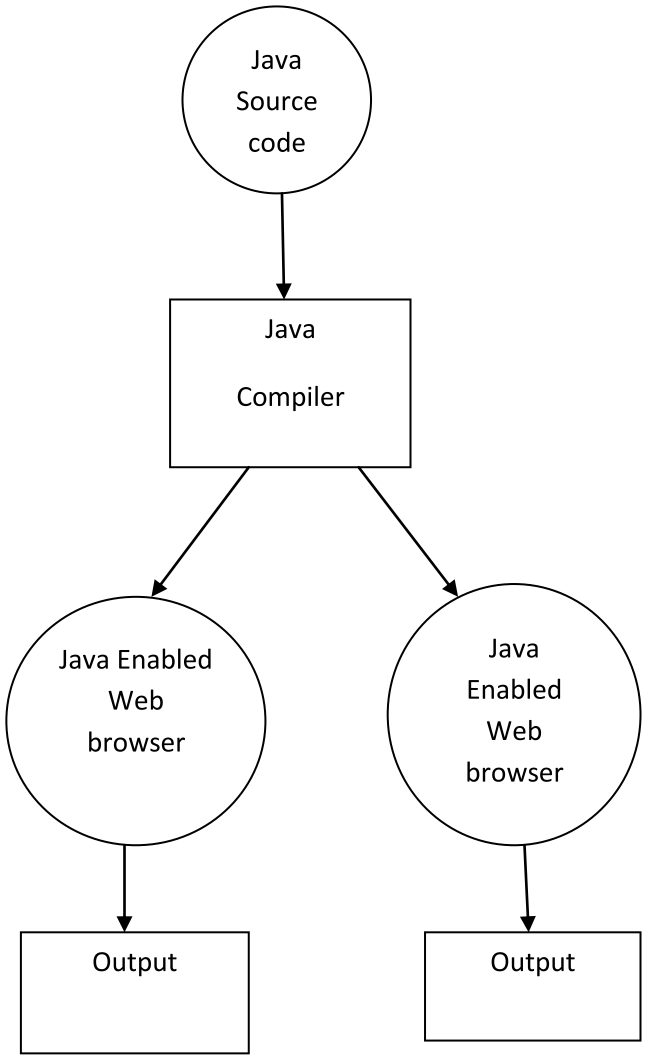
- Java was designed with the distributed background.
- Java can be transmitting,run over internet.

Dynamic :

Java was designed to adjust to an evolving environment:

- Even after binaries have been released, they can adjust to a changing environment
- Java loads in classes as they are wanted, even from across the network
- It defers lots of decisions (like object layout) to runtime, which solves several version problems like C++.

1.3.The Java Environment(java source file,structure,compilation)



JVM –Java Virtual Machine

A Java virtual machine (*JVM*) is a virtual machine that enables a computer to run Java programs as well as programs written in other languages and compiled to Java bytecode

JDK- Java Development Kit

he Java Development Kit (*JDK*) is a software development environment used for developing Java applications and applets

JRE-Java Runtime Environment

JDK includes the Java Runtime Environment (JRE), an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (javadoc) and other tools needed in Java development.

Overview:

Two types of Java Programs: 1.Stand alone applications 2. Web applets

1.Stand alone: Can read and write files and perform certain operations. Any **java** class with a main function can be considered a **standalone java application**.

2.Applet: Small java programs that will be used for internet application.An applet located on a remote computer (server) can be downloaded via internet & executed on a home computer (client) using Java capable browser.

Difference between Java & C

- Java has method to define classes & objects.
- Java does not consist of the C unique statements keywords goto, sizeof & typedef.
- Java does not have the data types struct, union and enum.
- Java does not describe the type modifiers keywords auto, extern, register, signed and unsigned.
- Does not contain a pre processor and we cannot use #define, #include & # ifdef statements.
- Java requires that the methods with no arguments must be declared with none parenthesis & not with the void keyword as done in c.

Difference between Java & C++

- Operator overloading is not possible in java programming
- Java does not have template classes as in c++.
- Java does not support multiple inheritance of classes.(indirectly used with new feature called interface).
- It does not support global variables. Every variable &function is declared within a class & forms part of that class.
- Pointer is not possible in java
- It replaces destructor function with a finalize() function
- No header files in Java like c++

1.3.1 JAVA SOURCE FILE

Simple Java Program

```
class Example
{
public static void main(String args[])
{
System.out.println("Hello");
}
}
```

Class : Example Declares a class[Every must be placed inside a class]
Class:Keyword, Example:identifier (specifies the name).

Main line:
main()-Starting point for the interpreter to start the execution of the program.

public: access specifier (making it reachable to all the other classes)

static: main must be declared as static; because the main interpreter uses this function before any objects are created.
void-main :function does not return any value(but simply prints a few text to the screen).
All the parameters to a function are declared inside a pair of parenthesis.
String args[] declares a argument named args, which hold Array of objects of the type string.

Output Line: println-function is a member of the out object,which is a static data field of system class.

- Println all the time appends a newline character to the close of the string.
- This means that any following output will start on a new line.
- Every java must close with a semi colon.

1.3.2.JAVA PROGRAM STRUCTURE

(Example program refer any program in unit 1 any 2)
The following table shows the structure of java program

Documentation Section
Package Statement
Import Statements
Interface Statement
Class Definition
Main Method Class
Main Method Definition

1.3.3.JAVA COMPILATION

The steps to execute java program

- preparation of the program text
- compilation of the program
- execution of the compiled program

1. Preparation of the program text
To write the program text we need to write a file containing code. For a Java program, The file name must be represented

***ClassName*.java**
where *ClassName*is the name of the class defined in the program. E.g.,Example.java
The program can be written by any program that allows one text file (editor). E.g., NotePad, Emacs, ...

2. Compilation of the program

The compilation of the program is essential to convert the program into a sequence of commands that can be openly executed by the computer. The standard Java compiler, which is part of the Java Standard Development Kit (Java SDK), is javac. To make use of it, you have to execute the following command:

javac *ClassName*.java

The compilation yields a file called *ClassName*.class, which contains the command that can be straight executed by the system. For example:

javac Example.java

- creates the file Example.class.

3. Execution of the compiled program
After compilation step, we have to execute the java program.In Java the execution of a program is done in following manner
java *ClassName*

(without .class). For example, the command

Java   Example
causes the execution of the code or programExample (or, more exactly, of the main method of the class First), and hence displays on the screen..

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;
}
}
```

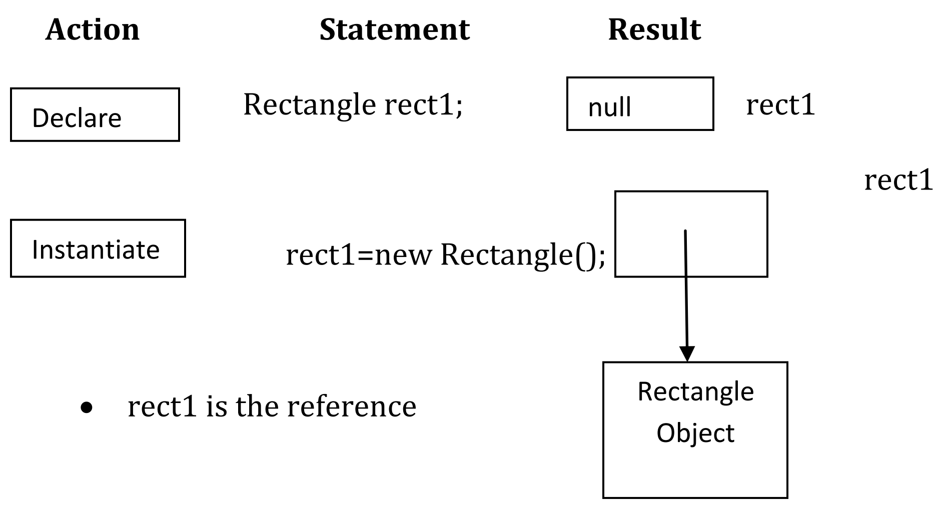
Creating objects

- 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.



- rect1 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:

```
rect1.length=15;
rect2.getData(20,10);
```

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.out.println(“Area1=”+area1);
System.out.println(“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.out.println("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.out.println("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 sum() {……….}
  ```
- 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.out.println("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.

1.4.7.Data Types in Java

Data type is a special keyword used to assignenough memory space for the data, in other words Data type is used for on behalf of the data in main memory (RAM) of the computer.

In java, there are two types of data types

- Primitive data types
- Non-primitive data types

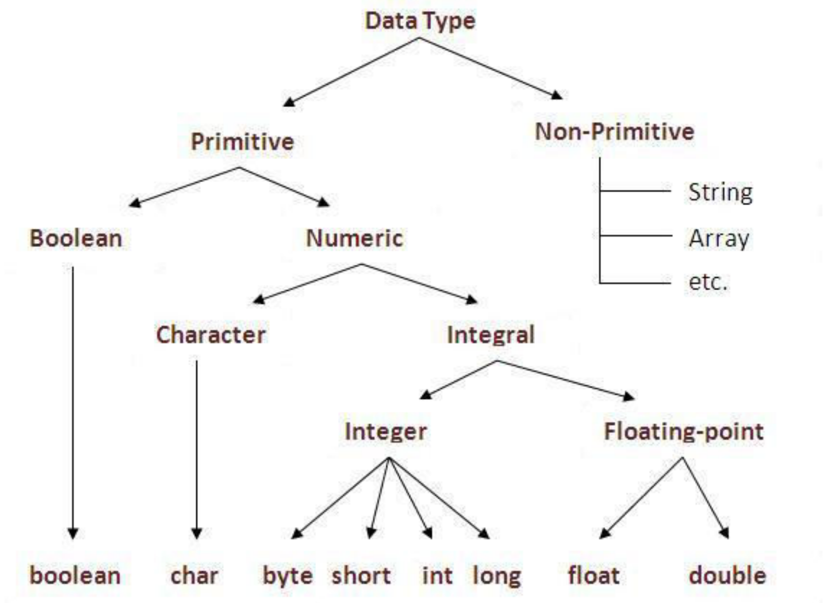


Figure 1: Classification of Data type

1.Primitive data types

Primitive data types are variables permits us to **store only one value** but they not at all allows us to store multiple values of similar type. This is a data type whose variable can contain maximum one value at a time.

Example:

```
int a;  // valid
```

a=10; // valid
a=10, 20, 30; // invalid

There are eight primitive data types given by Java. Primitive data types are predefined by the language and named through a keyword.

- **Numeric primitives:**short, int, long, float and double. These primitive data types can contain only numeric data. Operations related with such data types are those of easy arithmetic (addition, subtraction, etc.) or of comparisons (is greater than, is equal to, etc.)
  Example: double a=120.20;
  double b=50.20;
  double c= a+b;
- **Textual primitives:**byte and char. These primitive data types contain characters (that can be Unicode alphabets or even numbers). Operations related with such types are those of textual operation (comparing two words, joining characters to make words, etc.). though, byte and char can also support arithmetic operations.
  Example: char a='A';
  char b='B';
- **Boolean and null primitives:** boolean and null.
  Example: boolean flag=TRUE ;

Data Type	**Default Value**	**Default size**	**Range**
boolean	false	1 bit	True or False only
char	'\u0000'	2 byte	0 to 65,535
byte	0	1 byte	-128 to 127
short	0	2 byte	-32,768 to 32,767
int	0	4 byte	-2,147,483,648 to 2,147,483,647
long	0L	8 byte	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	0.0f	4 byte	1.40129846432481707e-45 to 3.40282346638528860e+38
double	0.0d	8 byte	4.94065645841246544e-324d to 1.79769313486231570e+308d

Table : Primitive Data type with Memory Size

2. Non-Primitive Data Types

It is Used to **store multiple values**.Reference variables are produced using defined constructors of the **classes**. They are used to contact objects. These variables are declared to be of a particular type that cannot be changed. Class objects and various types of array variables come below reference data type. Default value of any reference variable is null. A reference variable can be used to refer any other object of the declared type or any compatible type.

Objects and **Arrays** are the reference or non-primitive data types in Java. They are so called since they are handled “by reference” i.e. variables of their kind store the address of the object or array is stored in a variable. They are passed by reference. For Ex:

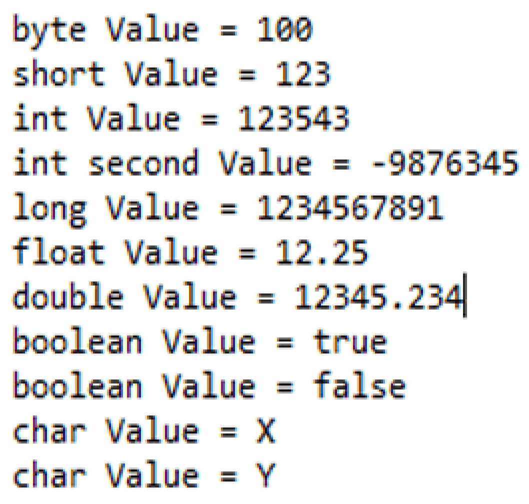
```
char []  arr = { 'a', 'b', 'c', 'd' };   //'arr' stores the references for the 4 values
```

Simple Example for Data type:

```
public class PrimitiveDemo
 {
    public static void main(String[] args)
    {
            byte b =100;
            short s =123;
            int v = 123543;
            int calc = -9876345;
            long amountVal = 1234567891;
            float intrestRate = 12.25f;
            double sineVal = 12345.234d;
            boolean flag = true;
            boolean val = false;
            char ch1 = 88; // code for X
            char ch2 = 'Y';
            System.out.println("byte Value = "+ b);
            System.out.println("short Value = "+ s);
            System.out.println("int Value = "+ v);
            System.out.println("int second Value = "+ calc);
            System.out.println("long Value = "+ amountVal);
            System.out.println("float Value = "+ intrestRate);
            System.out.println("double Value = "+ sineVal);
            System.out.println("boolean Value = "+ flag);
            System.out.println("boolean Value = "+ val);
            System.out.println("char Value = "+ ch1);
            System.out.println("char Value = "+ ch2);
    }

}
```

Output:



```
byte Value = 100
short Value = 123
int Value = 123543
int second Value = -9876345
long Value = 1234567891
float Value = 12.25
double Value = 12345.234
boolean Value = true
boolean Value = false
char Value = X
char Value = Y
```

The Primitive Types

The primitive types are also commonly referred to as simple types.Java defines eight primitive types of data: byte, short, int, long, char, float, double, and boolean.

These can be put in four groups:

Integers This group includes byte, short, int, and long, which are for whole-valued signed numbers.

Floating-point numbers This group includes float and double, which represent numbers with fractional precision.

Characters This group includes char, which represents symbols in a character set, like letters and numbers.

Boolean This group includes boolean, which is a special type for representing true/false values.

Types of Literals:

a. Integer Literals
b. Floating-Point Literals
c. Boolean Literals
d. Character Literals
e. String Literals

The Java Keywords:

There are 50 keywords currently defined in the Java language.

These keywords, combined with the syntax of the operators and separators, form the foundation of the Java language.

These keywords cannot be used as names for a variable, class, or method.

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Table : Java Keywords

1.4.8. VARIABLES

The variable is the fundamental unit of storage in a Java program. A variable is defined by the mixture of an identifier, a type, and an optional initializer. All variables have a range, which defines their visibility, and a lifetime. In Java, all variables should be declared before they can be used. The fundamental form of a variable declaration is shown here:

type identifier [= value][, identifier [= value] ...] ;

- type can be any one java data type, or the name of a class or interface.
- The identifier is the name of the variable.
- You can initialize the variable by mentioning an equal sign and a value.
- The initialization expression must answer in a value of the same type as that mentioned for the variable.

If we want to declare more than one variable of the specified type, we can use a comma separated list. Here are several examples of variable declarations of various types.

int a, b, c; **// declares three ints, a, b, and c.** 6

int d = 3, e, f = 5; // declares three more ints, initializing d and f.
byte z = 22; // initializes z.

Dynamic Initialization

Java permits variables to be initialized dynamically, any expression valid at the time the variable is declared.

Example Program: Demonstrate dynamic initialization.

```
//Demonstrate dynamic initialization.
class DynInit
{
public static void main(String args[])
{
double a = 3.0, b = 4.0;
// c is dynamically initialized
double c = Math.sqrt(a * a + b * b);
System.out.println("Hypotenuse is " + c);
}
}
```

The Scope and Lifetime of Variables

Scope:

- Each variables used have been declared at the beginning of the main( ) method. Java allows variables to be declared inside any block. A block is started with an opening curly brace and closed by a closing curly brace.
- A block defines a scope. Thus, every time you begin a new block, you are creating a new scope. A scope decides what objects are visible to other places of your program. It also decides the lifetime of those objects.
- Many other programming languages define two general types of scopes: global and local.
  In Java, there are two major scopes are defined in the class and those defined by a function. Variables declared within a scope are not visible (that is, accessible) to program that is given outside the scope.
- Nesting of scopes is possible. For example, Every time you create a block of code, you can create a new, nested scope. Whenever it happens the outer scope encloses the internal scope.

This means that objects declared on the outside scope will be visible to code within the internal scope.

Example Program: Scope of Variables

```
// Demonstrate block scope.
class Scope
{
public static void main(String args[])
{
int x; // known to all code within main
x = 10;
if(x == 10)
{
// start new scope
int y = 20;
// known only to this block 7
// x and y both known here.
System.out.println("x and y: " + x + " " + y);
x = y * 2;
}
// y = 100; // Error! y not known here
// x is still known here.
System.out.println("x is " + x);
}
}
```

Lifetime:

A variable declared inside a block will drop its value when the block is left. Thus, the life span of a variable is limited to its scope. If a variable declaration have aninitialize, then that variable can be reinitializedevery time and the block in which it is stated.

Example Program : Lifetime of a variable

```
// Demonstrate lifetime of a variable.
class LifeTime
{
public static void main(String args[])
{
int x;
for(x = 0; x < 3; x++)
{
int y = -1; // y is initialized each time block is entered
System.out.println("y is: " + y); // this always prints -1
y = 100;
System.out.println("y is now: " + y);
}
}
}
```

Output:

```
y is: -1
y is now: 100
y is: -1
y is now: 100
y is: -1
y is now: 100
```

1.4.9. OPERATORS

Operator in java is a symbol that is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in java which are given below:

- Unary Operator,

- Arithmetic Operator,
- Shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.
- **Java Operator Precedence**

Operator Type	**Category**	**Precedence**
Unary	postfix	*expr*++ *expr*--
	prefix	++*expr* --*expr* +*expr* -*expr* ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<<>>>>>
Relational	comparison	<><= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Java Unary Operator

The Java unary operators need only one operand. Unary operators are used to execute various operations i.e.:

- incrementing/decrementing a value by one
- negating an expression
- inverting the value of a Boolean

//Unary operator Example

```
class OperatorExample
{
public static void main(String args[])
{
int x=10;
System.out.println(x++); //10 (11)
System.out.println(++x); //12
System.out.println(x--); //12 (11)
System.out.println(--x); //10
}
}
```
Output:
```
10
12
12
10
```

Java Arithmetic Operators

Java arithmetic operators are used to perform various operations like addition, subtraction, multiplication, and division. They act as fundamental mathematical operations.

```
//Arithmetic operators
class OperatorExample
{
public static void main(String args[]){
System.out.println(10*10/5+3-1*4/2);
}
}
```
Output: 21

Java Assignment Operator:

Java assignment operator is one of the most general operators. It is used to allocate the value on its right to the operand on its left.

```
//Assignment operators
class OperatorExample
{
public static void main(String args[]){
int a=10;
int b=20;
a+=4;//a=a+4 (a=10+4)
b-=4;//b=b-4 (b=20-4)
System.out.println(a);
System.out.println(b);
}
}
```
Output:
```
14
16
```

Java Ternary Operator

Java Ternary operator is used as one liner replacement for if-then-else statement and used a lot in java programming. it is the only conditional operator which takes three operands.

// Ternary operator

```
class OperatorExample
{
public static void main(String args[]){
int a=2;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}
}
```

Output:

```
2
```

1.4.10 .CONTROL FLOW

A control statements can be any one of given categories: selection, iteration, and jump.
Java's Selection Statements Java provides two selection statements: if and switch.
These statements permit you to control the flow of your code execution based upon situation known only during run time.

Selection:

If
If else
Ladder if
Nested if else
Switch statement

//if else example

```
public class IfElseIfExample {
public static void main(String[] args) {
   int marks=65;

   if(marks<50){
      System.out.println("fail");
   }
   else if(marks>=50 && marks<60){
      System.out.println("D grade");
   }
   else if(marks>=60 && marks<70){
      System.out.println("C grade");
   }
   else if(marks>=70 && marks<80){
      System.out.println("B grade");
   }
   else if(marks>=80 && marks<90){
      System.out.println("A grade");
   }else if(marks>=90 && marks<100){
     System.out.println("A+ grade");
  }else{
      System.out.println("Invalid!");
   }
}
}
```
Output:
C grade

//switch
```
public class SwitchExample2 {
public static void main(String[] args) {
   int number=40;
   switch(number){
   case 10: System.out.println("10"); break;
   case 20: System.out.println("20");break;
   case 30: System.out.println("30");  break;
   default:System.out.println("Not in 10, 20 or 30");
   }
}
}
```
Output:
"Not in 10, 20 or 30

Iteration:
- While
- do while
- for

//while

```
public class WhileExample {
public static void main(String[] args) {
   int i=1;
   while(i<=10){
      System.out.println(i);
   i++;
   }
}
}
```
Output:1 2 3 4 5 6 7 8 9 10

//do-while
```
public class DoWhileExample {
public static void main(String[] args) {
   int i=1;
   do{
      System.out.println(i);
   i++;
   }while(i<=10);
}
}
```
Output:1 2 3 4 5 6 7 8 9 10

//for
```
public class ForExample {
public static void main(String[] args) {
   for(int i=1;i<=10;i++){
      System.out.println(i);
   }
}
}
```
Output:1 2 3 4 5 6 7 8 9 10

jump:

Java provides three jump statement:

- **break**
- **continue**
- **return**

// break and continue
```
        publicclassBreakAndContinue{
        publicstaticvoidmain(String args[]) {
    int[] numbers= newint[]{101,102,103,104,105,106,107,108,109,110};
                    intadd = 0;
        for(int i=0; i< numbers.length; i++){
                System.out.println("iteration: " + i);
        if(i == 5){
                  System.out.println("calling break statement");
                                        break;
                              }
        if(i%2 != 0){
        add = add + numbers[i];
                  System.out.println("calling continue statement");
                                        continue;
                }
                System.out.println("Last line of loop executed only for even number of
        iterations: " + numbers[i]);
              }
              System.out.println("This is outside the loop, sum: " + add);
           }
        }
```
Output:
```
Output is:
iteration: 0
Last line ofloop executed only for even number of iterations: 101
iteration: 1
```

```
calling continue statement
iteration: 2
Last line ofloop executed only for even number of iterations: 103
iteration: 3
calling continue statement
iteration: 4
Last line ofloop executed only for even number of iterations: 105
iteration: 5
calling break statement
This is outside the loop, sum: 206
```

1.4.11.ARRAYS

Array is a collection of contiguous or associated data items that share a general name.

Creating an Array

Arrays must be declared and created in the system memory before they are used

Creation of an array involves three steps

1. Declare the array
2. Create memory locations
3. Put values in to the memory locations.

Declaration of Arrays

Arrays in java may be declared in 2 forms

1. Type arrayname[];
2. Type[] arrayname;

Example

```
int number[];
int[] number;
```

Creation of Array

- After declaring an array we want to create memory.Java permits us to create arrays using new operator.

Syntax

```
arrayname=new type[size];
```

Ex:

```
number=new int[5];
average=new float[10];
```

Initialization of Arrays

- The initial step is to place values in to the array created. This process is called as initialization.

Syntax: arrayname[subscript]=value;

Ex: number[0]=10;
   number[1]=20;

Syntax: type arrayname[]={list of values};
Ex: int number[]={10,20,30,40,50};

One Dimensional Array

A list of items can be known one variable name with only one subscript and a variable is known as a single subscripted variable.

Array Length

- All arrays store the allocated size in a variable named length
- Access the length of the array a using a.length

Ex:int asize=a.length;

Example Program: Sorting a list of numbers

```
class Sorting
{
public static void main(String args[])
{
int number[]={10,20,45,5,15};
int n=number.length;
System.out.print("Given List");
for(int i=0;i<n;i++)
{
System.out.print(" "+number[i]);
}
System.out.println("\n");
for(int i=0;i<n;i++)
{
for(int j=i+1;j<n;j++)
{
if(number[i]<number[j])
{
int temp=number[i];
number[i]=number[j];
number[j]=temp;
}
}
}
System.out.println("Sorted List");
for(int i=0;i<n;i++)
{
System.out.print(""+number[i]);
}
System.out.println("");
}
}
```

Output:

```
Given List: 10    20    45    5     15
Sorted List:45    20    15    10    5
```

Two Dimensional Arrays

- A list of items can be known one variable name using 2 subscripts.
- That variable is called 2 dimensional array
    - First index=row
    - Second index=column

Ex: int myarray[][];
myarray=new int[3][4];

Example Program:

```
class Mutable
{
final static int rows=20;
final static int columns=20;
public static void main(String args[])
{
int product[][]=new int[rows][columns];
int rows,columns;
System.out.println("MUX table");
System.out.println("");
int i,j;
for(int i=10;i<rows;i++)
{
for(j=10;j<columns;j++)
{
product[i][j]=i*j;
System.out.println(""+product[i][j]);
}
System.out.println("");
}
}   }
```

Output:

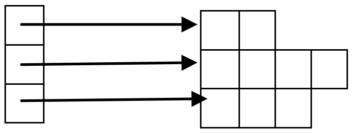
```
    100     110   120   130   ............................ 190
    110     121   132   143   ............................ 209
    190     209   228   247   ............................. 361
```

Variable Size Arrays

- Java considers multidimensional arrays as arrays of arrays
- It is feasible to declare a 2 dimensional array as
- int x[][]=new int[3][];
- These create 2d array as having different lengths for each row

X[0]=new int[2];

X[1]=new int[4];
X[2]=new int[3];



1.4.12.PACKAGES

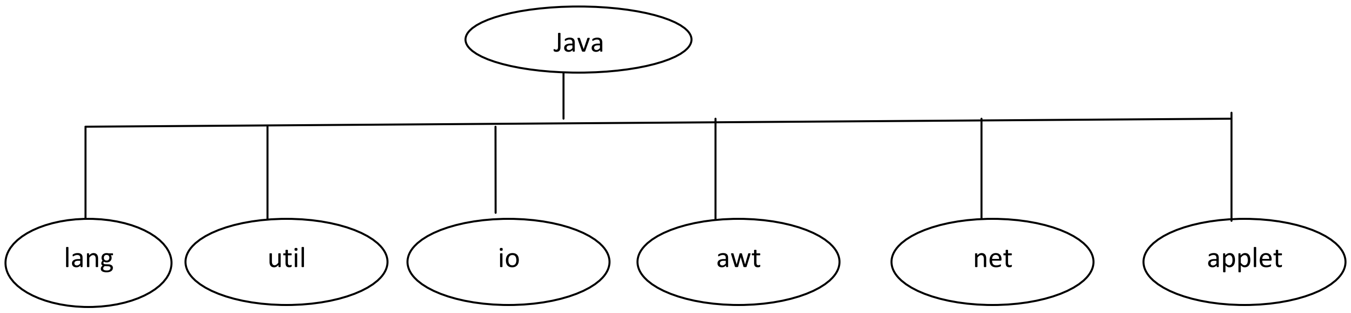
- Grouping mixture of classes and/or interfaces collectively
- Grouping is generally done according to functionality
- Packages act as containers for classes

Types:

1. Java API packages
2. User defined packages

Java API Packages

- Java API provides a large number of classes integrated in to various packages according to functionality



Package Name	Content
Java.lang [Primitive types, Strings, Mathfunctions, Threads]	Language Support class
Java.util [vectors,hash tables,random numbers,date]	Language utility classes
Java.io [input & output]	Input/output support classes
Java.awt [windows,buttons,lists,menus]	Set of classes for implementing GUI
Java.net	Classes for networking
Java.applet	Classes for creating and implementing applets

Using System Packages

There are 2 ways of accessing the classes available in a package

1. First Approach : fully qualified name

   Java.awt.color;

- Imports the color class & class name can now be openly used in the program.

2. Second Approach: Once or when we do not want to access any other classes of the package

- bring all the classes of java.awt package.

Use a class in number of times in a program/like to use different classes enclosed in a package.

import packagename.classname;

or

import packagename.*;

Naming Conventions

Can be named using standard naming rules

1. packages start with lowercase letters.
2. class names start with uppercase letters.
3. Methods start with lowercase letters.
   Ex: double y=java.lang.Math.sqrt(x)
   java.lang : package
   Math: class name
   sqrt: method name.

Benefits of Packages

- The classes enclosed in the packages of other programs can be simply reused
- They give a way to "hide" classes thus preventing from new programs or packages
- Also supply a way for separating "design" from "coding"
- Two different classes in 2 various packages can have similar name.

Java user defined package

Creating a user defined Package

- First declare the package name using the package keyword continued by a package name
- This must be the initial statement in a java source file.
- Then you can define a class just as we usually define a class

```
package firstpackage;//package declaration
public class Firstclass//class definition
{
Body
}
```

Creating our own package or user defined packages follows the following steps

1. Declare the package at the starting of a file
   package packagename;
2. Define the class that is to be place in the package & declare it public.
3. Create a subdirectory below the directory where the main source files are stored
4. Keep the listing as the classname.java file in the subdirectory created
5. Compile the file.This generates class file in the subdirectory.

- Java also provides the concept of package hierarchy
- This is done by specifying many names in a package statement,separated by dots.
  package firstpackage.secondpackage;

Accessing a Package

- In java programming package can be accessed either using a fully qualified class name or using another shortcut method through the import statement.
- The general form of import statement
  import package1[.package2][.package3].classname;
- The system must end with a semicolon(;)
- The import statement should become visible before any class definitions in a source file.

` **Ex:**
Importing a particular class
import firstpackage.secondpackage.Myclass;

- After defining, all the fields of the class Myclass can be straightly accessed using the class name or its objects can be used directly without specifying the package name.

Ex:
import packagename.*;

- May denote a single package or a hierarchy of packages. * represents that the compiler should look for this whole hierarchy when it encounters a class name

Example:

```
package package1;
public class ClassA
{
public void displayA()
{
System.out.println("Class A");
}
}

import package.ClassA;
class Test
{
public static void main(String args[])
{
ClassA objectA=new ClassA();
objectA.display();
}
}
```

Output:
ClassA

}

Example

```
package package2;
public  class ClassB
{
protected int m=10;
public  void display()
{
System.out.println("Class B");
System.out.println("m="+m);
}
}
```

Example:

```
import package1.ClassA;
import package2.*;
class Test2
{
public static void main(String args[])
{
ClassA objA=new ClassA();
ClassB objB=new ClassB();
objA.displayA();
objB.displayB();
}
}
}
```

Output:

```
ClassA
ClassB
m=10
```

Example

```
import package.ClassB;
class ClassC extends ClassB
{
int n=20;
void displayC()
{
System.out.println("Class C");
System.out.println("m="+m);
System.out.println("n="+n);
}
}

class Test2
```

```
{
public static void main(String args[])
{
ClassC objC=new ClassC();
objC.displayB();
objC.displayC();
}
}
```

Output
ClassB
M=10
ClassC
m=10
n=20

1.4.13.JAVADOC COMMENTS

TYPES

- Class Comments
- Method Comments
- Field Comments
- Package & Overview Comments

  Sample tags of javadoc:

  @author
  @exception
  @deprecated
  {@link}
  @param
  @return

Example:

```
/**
* <h1>Hello, World!</h1>
* The HelloWorld program implements an application that
* simply displays "Hello World!" to the standard output.
* <p>
* Giving proper comments in your program makes it more
* user friendly and it is assumed as a high quality code.
*
*
* @author  Zara Ali
* @version 1.0
* @since   2014-03-31
*/
public class HelloWorld {

   public static void main(String[] args) {
      /* Prints Hello, World! on standard output.
      System.out.println("Hello World!");
   }
}
```

2 MARKS QUESTIONS AND ANSWERS

1. **What is Programming Paradigm?**

The word paradigm to mean “any example or model”. Object-oriented programming paradigm suggests new ways of thinking for finding a solution to a problem. Hence the programmers should keep their mind tuned in such a manner that they are not to be blocked by their preconceptions experienced in other programming languages such as structured programming. Proficiency in object-oriented programming requires talent, creativity, intelligence, logical thinking and the ability to build and use abstractions and experience.

2. **What is Object Oriented Programming?**

Object-Orientation is a set of tools and methods that enable software engineers
to build reliable, user friendly, maintainable, well documented, reusable software systems that fulfills the requirements of its users.

3. **What are important features in object-oriented programming and design?**

- Improvement over the structured programming paradigm.
- Emphasis on data rather than algorithms.
- Procedural abstraction is complemented by data abstraction.
- Data and associated operations are unified, grouping objects with common attributes, operations and semantics.

4. **How could Java classes direct program messages to the system console, but error messages, say to a file?**

The class System has a variable out that represents the standard output, and the variable err that represents the standard error device. By default, they both point at the system console. This how the standard output could be re-directed: Stream st = new Stream(new FileOutputStream("output.txt")); System.setErr(st); System.setOut(st);

5. **What's the difference between an interface and an abstract class?**

An abstract class may contain code in method bodies, which is not allowed in an interface. With abstract classes, you have to inherit your class from it and Java does not allow multiple inheritance. On the other hand, you can implement multiple interfaces in your class.

6. **Why would you use a synchronized block vs. synchronized method?**

Synchronized blocks place locks for shorter periods than synchronized methods.

7. **Explain the usage of the keyword transient?**

This keyword indicates that the value of this member variable does not have to be serialized with the object. When the class will be de-serialized, this variable will be initialized with a default value of its data type (i.e. zero for integers).

8. Define object and object variable ?(Apr/May 2011)(May/June 2013)

An Objectis anything having crisply defined conceptual boundaries. Book, pen, train, employee, student, machine, etc., are examples of objects.

•Objects in java are created using the new operator

•New operator creates an object of the specified class and returns a reference to that object.

Ex:

Rectangle rect1;//declare

Rect1=new Rectangle();/instantiate

•First statement declares a variable to hold the object reference & the second one actually assigns the object reference to the variable.

9. What is meant by private access specifier?Nov/Dec 2011

Private members are accessible only within the class and cannot be inherited.

10. What is the need for javadoc multiline commentsNov/Dec 2011

To describe the program multiline comments are used. /*…..*/

11. Define Constructor.Nov/Dec 2011

- Constructor enables the object to initialize itself.
- Constructor also can not have any return type, constructor's are automatically chained by using this keyword and super.
- Two Types.Default Constructor and Parameterized constructor

12. What is a class? (Nov/Dec 2011)

Class is a data type. It generates object. It is the prototype or model. It does not occupy memory location. It cannot be manipulated because it is not available in the memory.

13. What are the fundamental features of object-oriented programming?

- Encapsulation
- Data Abstraction
- Inheritance
- Polymorphism
- Extensibility
- Persistence
- Delegation
- Genericity
- Object Concurrency
- Event Handling
- Multiple Inheritance
- Message Passing

14. What is Encapsulation?
The process, or mechanism, by which you combine code and the data it manipulates into a single unit, is commonly referred to as *encapsulation*. Encapsulation provides a layer of security around manipulated data, protecting it from external interference and misuse.

15. What is a Type?
A type is a set of values together with one or more operations that can be applied uniformly to all these values.

16. What is a class Members?
The non-static members of a class (variables and methods) are also known as
instance variables and methods while the non-static members are also known as class variables and class methods.

17. What is meant by Access Specifiers?
The purpose of access specifiers is to declare which entity cannot be accessed from where. Its effect has different consequences when used on a class, class member (variable or method), constructor.

18. Explain the Static Class Method.
Static methods typically take all they data from parameters and compute something from those parameters, with no reference to variables. This is typical of methods which do some kind of generic calculation. A good example of this are the many utility methods in the predefined Math class.

19. Explain the accessing of static call method.
A common use of static variables is to define "constants". Examples from the Java library are Math.PI or Color.RED. They are qualified with the class name, so you know they are static. Any method, static or not, can access static variables. Instance variables can be accessed only by instance methods.

20. Explain the Constructor method in Java.
When you create a new instance (a new object) of a class using the new keyword, a *constructor* for that class is called. Constructors are used to initialize the instance variables (fields) of an object.

21. How to Write a Finalize method using Java.(May/June 2013)
Before an object is garbage collected, the runtime system calls its finalize() method. The intent is for finalize() to release system resources such as open files or open sockets before getting collected. Your class can provide for its finalization simply by defining and implementing a method in your class named finalize(). Your finalize() method must be declared as follows:
protected void finalize () throws throwable

22. What is an array? How to declare an array in java.

Array is the most important thing in any programming language. By definition, array is the static memory allocation. It allocates the memory for the same data type in sequence. It contains multiple values of same types. It also store the values in memory at the fixed size. Multiple types of arrays are used in any programming language such as: one - dimensional, two - dimensional or can say multi - dimensional.

Declaration of an array:

int num[]; or int num = new int[2];

23. What is the use of Strings in Java?

The String class is commonly used for holding and manipulating strings of text in Java programs. It is found in the standard java.lang package which is automatically imported, so you don't need to do anything special to use it.

24. Explain about String Tokenizer?

*StringTokenizer*class objects may be created by one of three constructor methods depending on the parameters used. The first parameter string is the source text to be broken at the default set of *whitespace* delimiters (space, tab, newline, cr, formfeed).

25. Explain the Java Packages.

A *package* is a grouping of related types providing access protection and name space management. Note that *types* refer to classes, interfaces, enumerations, and annotation types. Enumerations and annotation types are special kinds of classes and interfaces, respectively, so *types* are often referred to in this lesson simply as *classes and interfaces*.

26. What is JavaDoc? (Nov/Dec 2011)

Javadoc is a convenient, standard way to document your Java code. Javadoc is actually a special format of comments. There are some utilities that read the comments, and then generate HTML document based on the comments. HTML files give us the convenience of hyperlinks from one document to another. Most class libraries, both commercial and open source, provide Javadoc documents.

27. Mention the types of the JavaDoc.

There are two kinds of Javadoc comments: class-level comments, and member-level comments. Class-level comments provide the description of the classes, and member-level comments describe the purposes of the members.

28. Mention the purpose of finalize method.May/June 2013

Reclaim the Object, its last chance for any object to perform cleanup activity i.e. releasing any system resources held, closing connection if open etc

29.How to access member variables from the class?

ObjectName.VariableName=Value;
ObjectName.MethodName(Parameters List);

30. Explain about Java I/O Package?

The Java I/O Package (java.io) provides a set of input and output streams used to read and write data to files or other input and output sources. The classes and interfaces defined in java.io are covered fully in Input and Output Streams.

31. Explain about Java Utility Package?
This Java package, java.util, contains a collection of utility classes. Among them are several generic data structures (Dictionary, Stack, Vector, Hashtable) a useful object for tokenizing a string and another for manipulating calendar dates. The java.util package also contains the Observer interface and Observable class, which allow objects to notify one another when they change. The java.util classes aren't covered separately in this tutorial although some examples use these classes.

32. Explain about Applet Package?
This package contains the Applet class -- the class that you must subclass if you're writing an applet. Included in this package is the AudioClip interface which provides a very high level abstraction of audio. Writing Applets explains the ins and outs of developing your own applets.

33. Explain about the Java Packages?(NOV/DEC2010)
Several packages of reusable classes are shipped as part of the Java development environment. Indeed, you have already encountered several classes that are members of these packages: **String, System, and Date**, to name a few. The classes and interfaces contained in the Java packages implement various functions ranging from networking and security to graphical user interface elements.

13 MARK QUESTIONS

1. Explain OOP Principles.
2. Explain the features of Java Language.
3. Compare and Contrast Java with C.
4. Compare and Contrast Java with C++.
5. Explain Constructors with examples.
6. Explain the methods available under String and String Buffer Class.
7. Explain the Date Class methods with examples.
8. Discuss in detail the access specifiers available in Java.
9. Explain the different visibility controls and also compare with each of them.
10. Explain the different methods in java.Util.Arrays class with example.
11. Explain Packages in detail.
12. Discuss the methods under Array Class.

PART-C (15 MARKS)

13. Discuss some of the classes available under Lang package and develop your own applications..
14. Illustrate with examples: static and final.
15. Explain method overriding with example program.
16. What is javaDoc? Explain the comments for classes, methods, fields and link.
17. Develop library application Programs using fundamental concepts in Java.