Programming with JAVA – Overview of Java Language, Classes Objects and Methods, Method Overloading and Inheritance, Overriding Methods, Final Variables and Methods. Interfaces, Packages, Multithreaded programming, Managing Errors and Exceptions.

PROGRAMMING WITH JAVA

JAVA is an object oriented language developed by Sun Microsystems of US A in 1991

Features of JAVA

1. Compiled and Interpreted:- Java compiler translates source code into bytecode instructions. Bytecode are not machine instructions and in the next stage, Java interpreter generates machine code that can be directly executed by the machine that is running the Java program. We can thus say that Java is both a compiled and an interpreted language.

2. Platform – Independent:- Java programs can be easily moved from one computer system to another, anywhere and anytime. Changes and upgrades in operating systems, processors and system resources will not force any changes in Java program. Java ensures portability in two ways. First, Java compiler generates bytecode instructions that can be implemented on any machine. Secondly, the sizes of the primitive data types are machine independent.

3. Object – Oriented:- Java is a true object – oriented language. All program code and data reside within objects and classes. Java comes with an extensive set of classes, arranged in packages that we can use in our programs.

4. Robust and Secure :- Robust simply means strong. Java puts a lot of emphasis on early checking for possible errors, as Java compilers are able to detect many problems that would first show up during execution time in other languages. Java has the strong memory allocation and automatic garbage collection mechanism. It provides the powerful exception handling and type checking mechanism as compare to other languages. Compiler checks the program whether there any error and interpreter checks any run time error and makes the system secure from crash. All of the above features make the Java language robust.

5. Distributed:- Java is designed as a distributed language for creating applications on networks. It has the ability to share both data and programs. Java applications can open and access remote objects on internet as easily as they can do in local system.

6. Simple :- Java is a simple language. Many features of C++ that are either redundant or
sources of unreliable code are not part of Java. For example, Java does not use pointers, preprocessor header files etc. It also eliminates operator overloading and multiple inheritance.

7. Multithreaded:- Multithreaded means handling multiple tasks simultaneously. Java supports multithreaded programs. This means that we need not wait for the application to finish one task before beginning another.

8. High Performance :- Java architecture is designed to reduce overheads during runtime. The incorporation of multithreading enhances the overall execution speed of Java programs.

9. Extensible :- Extensible code means java provides inheritance and with the help of inheritance we reuse the code that is pre-defined and also uses all the built in functions of java and classes.

10. Dynamic:- Java is a dynamic language. Java is capable of dynamically linking in new class

OVERVIEW OF JAVA LANGUAGE

We can develop two types of Java programs :

1. Standalone applications :- Programs written in Java to carry out certain tasks on a
standalone local computer.

2. Web applets:- S mall Java programs developed for internet applications.

Simple Java Program

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

Output: Hello World

The first line declares a class, which is an object – oriented construct. Everything must be placed inside a class.

public static void main(String []args):- This line defines a method named main.

public:- This keyword is an access specifier that declares the main method as unprotected and therefore making it accessible to all other classes.

static:- This keyword declares this method as one that belongs to t he entire class and not part of any object of the class. The main method must be declared as static since the interpreter uses this method before any objects are created.

void:- This keyword states that the main method does not return any value.
System.out.println("Hello World"); :- This line prints a string on the output screen and appends a newline character at the end of the string.

Java Program Structure

A Java program may contain one or more sections as given below:

Documentation Section
Package Statement
Import Statements
Interface Statements
Class Definitions
Main Method Class
{
Main Method Definition
}

Eg. : import student.test;
This statement instructs the interpreter to load the test class contained in the package student. Using import statements, we can have access to classes that are part of other named packages.

Documentation Section:-
It comprises a set of comment lines giving the name of the program, the author and other details, which the programmer would like to refer to at a later stage. Comments must explain the details of classes and algorithms in the file. This would help in maintaining the program.

Package Statement:- This statement declares a package name and informs the compiler that the classes defined here belong to this package. (eg. : package student;)

Import Statements :- This statement is similar to #include statement in C++.

Interface Statements :- An interface is like a class but includes a group of method declarations. This section is used only when we need to implement the multiple inheritance feature in the program.

Class Definitions :- A Java program may contain multiple class definitions.

Main Method Class:- Since every program requires a main method as its starting point, this class is the essential part of a Java program. A simple Java program may contain only this part. The main method creates objects of various classes and establishes communications between them. O n reaching the end of main, the program terminates and the control passes back to the operating system

CLASSES, OBJECTS AND METHODS
Classes provide a convenient method for packing together a group of logically related data
items and functions that work on them.

In Java data items are called fields, and functions are called methods.

Defining a Class
The syntax to define a class is as given below:

```
class classname [extends superClassName]
 {
field  declaration;
```

method declaration;
}

Field Declaration

Data is encapsulated in a class by placing data fields inside the body of the class definition. These variables are called instance variables because they are created whenever an object is created.

Method Declaration and Constructors Example:

```
class Room {
float length, breadth;
Room(float a, float b ) // Defining Construtor
 {
length = a; breadth = b;
}
void displayArea()  // Defining a method
 {
System.out.println("Area =  " + (length*breadth));
}
}
class RoomArea {
public static void main(String []args)
 {
Room room1;
room1 = new Room(100, 200); // object creation and calling constructor
room1.displayArea();
}
}
```

Output: Area = 20000

Creating Object Also known as instantiating an object.

Objects in Java are created using the new operator. The new operator creates an object of the
specified class and returns a reference to that object.

Room room1;

This statement declares a variable to hold the object reference.

room1 = new Room(100, 200);

This statement assigns the object reference to the variable. The variable room1 is now an

object of the class Room.
Both statements can be combined into one as shown below:
Room room1 = new Room(100, 200);In the above example the class Room has a parameterized constructor. Therefore the constructor is invoked explicitly by passing arguments in the object creation statement.

If there was no such constructors explicitly defined inside the class, then the object creation statement would be like:
Room room1 = new Room();
The method Room() is the default constructor of the class.

- We can create any number of objects of Room.

Any effect in the variable of one object has no effect on the variables of another object.

Accessing Class Members

We cannot access the instance variables and the methods directly from outside the class.
To do this, we must use the concerned object and dot operator as shown below.
objectName.variableName =value;
objectName.methodName(parameters);
Here objectName is the name of the object, variableName is the name of the instance variable inside the object, and methodName is the method that we wish to call.

METHOD OVERLOADING

1. It is possible to create methods that have the same name, but different parameter lists and different definitions. This is called method overloading.
2. When we call a method in an object, Java matches up the method name first and then the number and type of parameters to decide which one of the definitions to execute. This process is known as polymorphism
3. To create an overloaded method, all we have to provide several different method definitions in the class, all with the same name, but with different parameter lists.
4. The difference may either be in the number or type of arguments.

Example:

```
 class S um
 {
int add(int a, int b)
{
return (a+b);
}
int add(int a, int b, int c)
{ return (a+b+c);
}
}
 class Addition
{
public static void main(String args[ ])
 {
Sum obj = new Sum(); int s1 = obj.add(1, 2, 3); // invokes the function  int
add(int a, int b, int c)
int s2 = obj.add(10, 20); //  invokes the function  int add(int a, int b);
System.out.println("S um =  " + s1 );
System.out.println("S um =  "+ s2 );
}
}
```

Output:

```
Sum = 6
Sum = 30
```