MODULE 2: INTRODUCTION TO JAVA

Syllabus:

Objects and Arrays, Namespaces, Nested Classes

Java and Java applications; Java Development Kit (JDK); Java is interpreted, Byte Code, JVM; Object-oriented programming; Simple Java

Data types and other tokens: Boolean variables, int, long, char, operators, arrays, white spaces, literals, assigning values; Creating and destroying objects; Access specifiers.

Operators and Expressions: Arithmetic Operators, Bitwise operators, Relational operators, The Assignment Operator, The ? Operator; Operator Precedence; Logical expression; Type casting; Strings.

Control Statements: Selection statements, iteration statements, Jump Statements.

Arrays of Objects:

- It is possible to have arrays of objects.
- The syntax for declaring and using an object array is exactly the same as it is for any other type of array.
- A array variable of type class is called as an array of objects.

Program:

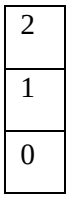
```
#include<iostream.h>
 class  c1
 {
                int i;
      public
      :         void get_i(int j)
                 {
                        i=j;
                }
                void show( )
                {
                        cout<<i<<endl;
                }
};
void main( )
{
              c1  obj[3];    // declared array of objects
            for(int i=0;i<3;i++)
                    obj[i].get_i(i);

            for(int i=0;i<3;i++)
                     obj[i].show( );
}
```


In the above program,we have declared object obj as an array of objects[i.e created 3 objects].

The following statement:

obj[i].get_i(i);

invokes get_i( ) function 3 times,each time it stores value of i in the index of obj[i].that is after the execution of complete loop,the array of object "obj" looks like this:



The following statement

obj[i].show( );

displays the array of objects contents:0,1,2

output: 0

1

2

Namespace

What is Namespace in C++?

Namespace is a new concept introduced by the ANSI C++ standards committee. For using identifiers it can be defined in the namespace scope as below.

Syntax:

```
using namespace std;
```

In the above syntax "std" is the namespace where ANSI C++ standard class libraries are defined. Even own namespaces can be defined.

Syntax:

```
namespace namespace_name
{
    //Declaration of variables, functions, classes, etc.
}
```

Example :

```
#include <iostream.h>
```

```
using namespace std; namespace Own
{
      int a=100;
}
 int main()
{
      cout << "Value of a is:: " << Own::a;
      return 0;
}
```

Result :

Value of a is:: 100

In the above example, a name space "Own" is used to assign a value to a variable. To get the value in the "main()" function the "::" operator is used.

Nested Classes in C++

Nested class is a class defined inside a class that can be used within the scope of the class in which it is defined. In C++ nested classes are not given importance because of the strong and flexible usage of inheritance. Its objects are accessed using "Nest::Display".

Example :

```
#include <iostream.h>
class Nest
{
      public:
            class Display
           {
               private:
                        int s;
               public:
```

```
                void sum( int a, int b)
                {
                        s =a+b;
                }
                void show( )
                {
                        cout << "\nSum of a and b is:: " << s;
                }
        }; //closing of inner class
}; //closing of outer class


void main()
{
      Nest::Display x;            // x is a object, objects are accessed using "Nest::Display".
        x.sum(12, 10);
       x.show();
}
```

Result : Sum of a and b is::22

In the above example, the nested class "Display" is given as "public" member of the class "Nest".

Constructors

- ✓ Constructors are special class functions which performs initialization of every object. The Compiler calls the Constructor whenever an object is created. Constructor"s initialize values to data members after storage is allocated to the object.

```
class A
{
          int x;
           public: A();  //Constructor
};
```

- ✓ While defining a constructor you must remember that the name of constructor will be same as the name of the class, and constructors never have return type.

- ✓ Constructors can be defined either inside the class definition or outside class definition using class name and scope resolution :: operator.

```
class A
{
        int i;
        public:
                A(); //Constructor declared
};
```

```
A::A() // Constructor definition
{
        i=1;
}
```

Types of ConstrUCTors

Constructors are of three types :

1. Default Constructor
2. Parametrized Constructor
3. Copy Constructor

DeFAULT ConstrUCTor

Default constructor is the constructor which doesn't take any argument. It has no parameter.

Syntax :


```
class_name ()
{
        Constructor Definition
}
```

Example :

```
class Cube
{
        int side;
        public:  Cube()                 //constructor
                 {
                        side=10;
                  }
};

int main()
{
        Cube c;       //constructor is going to call
        cout << c.side;
```

```
}
```
Output : 10

In this case, as soon as the object is created the constructor is called which initializes its data members.

A default constructor is so important for initialization of object members, that even if we do not define a constructor explicitly, the compiler will provide a default constructor implicitly.

```
class Cube
{
 int side;
};
```

```
int main()
{
 Cube c;
 cout << c.side;
}
```
Output : 0

In this case, default constructor provided by the compiler will be called which will initialize the object data members to default value, that will be 0 in this case.

Parameterized Constructor

These are the constructors with parameter. Using this Constructor you can provide different values to data members of different objects, by passing the appropriate values as argument.

Example :

```
class Cube
{
        int side;
        public:
                Cube(int x)
                {
                        side=x;
                }
};

int main()
{
        Cube c1(10);
        Cube c2(20);
        Cube c3(30);
        cout << c1.side;
        cout << c2.side;
        cout << c3.side;
}
```
OUTPUT : 10 20 30

By using parameterized construcor in above case, we have initialized 3 objects with user defined values. We can have any number of parameters in a constructor.

copy constructor

What is copy constructor and how to use it in C++?

A **copy constructor** in C++ programming language is used to reproduce an identical copy of an original existing object.It is used to initialize one object from another of the same type.

Example :

```
#include<iostream>
using namespace std;
class copycon
{
      int copy_a,copy_b; // Variable Declaration
      public:
            copycon(int x,int y)
          {
            //Constructor with Argument
                copy_a=x;
                copy_b=y; // Assign Values In Constructor
           }
           void Display()
          {
                        cout<<"\nValues :"<< copy_a <<"\t"<< copy_b;
          }
};
int main()
 {
          copycon obj(10,20);
          copycon obj2=obj; //Copy Constructor
         cout<<"\nI am Constructor";
         obj.Display(); // Constructor invoked.
         cout<<"\nI am copy Constructor";
         obj2.Display();
          return 0;
}
```

Result :

```
I am Constructor
Values:10 20
I am Copy Constructor
Values:10 20
```

Constructor Overloading

- ✓ Just like other member functions, constructors can also be overloaded. In fact when you have both default and parameterized constructors defined in your class you are having Overloaded Constructors, one with no parameter and other with parameter.
- ✓ You can have any number of Constructors in a class that differ in parameter list.

```
class Student
{
        int rollno;
        string name;
        public:
                Student(int x)
                {
                        rollno=x;
                        name="None";
                }
                Student(int x, string str)
                {
                        rollno=x ;
                        name=str ;
                }
};

int main()
{
        Student A(10);
        Student B(11,"Ram");
}
```

In above case we have defined two constructors with different parameters, hence overloading the constructors.

One more important thing, if you define any constructor explicitly, then the compiler will not provide default constructor and you will have to define it yourself.

In the above case if we write Student S; in **main()**, it will lead to a compile time error, because we haven't defined default constructor, and compiler will not provide its default constructor because we have defined other parameterized constructors.

Destructors

- ✓ Destructor is a special class function which destroys the object as soon as the scope of object ends. The destructor is called automatically by the compiler when the object goes out of scope.
- ✓ The syntax for destructor is same as that for the constructor, the class name is used for the name of destructor, with a **tilde** ~ sign as prefix to it.


```
class A
{
 public:
          ~A();
};
```

Destructors will never have any arguments.

Example to see how ConstrUCTor and DestrUctor is called

```
class A
{
A()
 {
  cout << "Constructor called";
 }
```

```
~A()
 {
  cout << "Destructor called";
 }
};

int main()
{
 A obj1; // Constructor Called
 int x=1
 if(x)
  {
   A obj2; // Constructor Called
  } // Destructor Called for obj2
} // Destructor called for obj1
```

MODULE 2: Introduction to JAVA

Basic concepts of object oriented programming

Object:

This is the basic unit of object oriented programming. That is both data and method that operate on data are bundled as a unit called as object. **It is a real world entity (Ex:a person, book, tables, chairs etc…)**

Class:

Class is a **collection of objects** or class is a **collection of instance variables and methods**. When you define a class, you define a blueprint for an object. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

Abstraction:

Data abstraction refers to, **providing only essential information** to the outside word and hiding their background details ie. to represent the needed information in program without presenting the details.

For example, a database system hides certain details of how data is stored and created and maintained. Similar way, C++ classes provides different methods to the outside world without giving internal detail about those methods and data.

Encapsulation:

Encapsulation is ***placing the data and the methods/functions*** that work on that data in the same place. While working with procedural languages, it is not always clear which functions work on which variables but object-oriented programming provides you framework to place the data and the relevant functions together in the same object.

Inheritance:

One of the most useful aspects of object-oriented programming is **code reusability**. As the name suggests Inheritance is the process of forming a new class from an existing class that is from the existing class called as base class, new class is formed called as derived class.

This is a very important concept of object oriented programming since this feature helps to reduce the code size.

Polymorphism:

The ability to use a method/function in different ways in other words giving different meaning for method/ functions is called polymorphism. Poly refers many. That is a single method/function functioning in many ways different upon the usage is called polymorphism.

Java History:

Java is a general-purpose object oriented programming language developed by sun Microsystems of USA in the year 1991. The original name of Java is Oak. Java was designed for the development of the software for consumer electronic devices like TVs, VCRs, etc.

Introduction: Java is a general purpose programming language. We can develop two types of Java application. They are:

(1). Stand alone Java application.
(2). Web applets.

Stand alone Java application: Stand alone Java application are programs written in Java to carry out certain tasks on a certain stand alone system. Executing a stand-alone Java program contains two phases:

(a) Compiling source coded into bytecode using javac compiler.
(b) Executing the bytecodede program using Java interpreter.

Java applet: Applets are small Java program developed for Internet application. An applet located on a distant computer can be downloaded via Internet and execute on local computer.

Java and Internet:

Java is strongly associated with Internet. Internet users can use Java to create applet programs and run them locally using a "Java enabled Browser" such as "hotjave". They can also use a Java enabled browser to download an applet locating on any computer any where in the internet and run them locally.

Internet users can also set their web-sites containing Java applets that could be used by other remote users of Internet. The ability of the Java applets to hitch a ride on the information makes Java a unique programming language for Internet.

Java Environment:

Java environment includes a large number of development tools and hundreds of classes and methods. The Java development tools are part of the systems known as Java development kit (JDK) and the classes and methods are part of the Java standard library known as Java standard Library (JSL) also known as application program interface (API).

Java Features:

(1) Compiled and Interpreted
(2) Architecture Neutral/Platform independent and portable
(3) Object oriented
(4) Robust and secure.
(5) Distributed.
(6) Familiar, simple and small.
(7) Multithreaded and interactive.
(8) High performance
(9) Dynamic and extendible.

1. **Compiled and Interpreted**
   Usually a computer language is either compiled or interpreted. Java combines both these approaches; first java compiler translates source code into bytecode instructions. Bytecodes are not machine instructions and therefore, in the second stage, java interpreter generates machine code that can be directly executed by the machine that is running the java program.

2. **Architecture Neutral/Platform independent and portable**
   The concept of Write-once-run-anywhere (known as the Platform independent) is one of the important key feature of java language that makes java as the most powerful language. Not even a single language is idle to this feature but java is closer to this feature. The programs written on one platform can run on any platform provided the platform must have the JVM.

3. **Object oriented**
   In java everything is an Object. Java can be easily extended since it is based on the Object model.java is a pure object oriented language.

4. **Robust and secure.**
   Java is a robust language; Java makes an effort to eliminate error situations by emphasizing mainly on compile time error checking and runtime checking. Because of absence of pointers in java we can easily achieve the security.

5. **Distributed.**
   Java is designed for the distributed environment of the internet.java applications can open and access remote objects on internet as easily as they can do in the local system.

6. **Familiar, simple and small.**
   Java is designed to be easy to learn. If you understand the basic concept of OOP java would be easy to master.

7. **Multithreaded and interactive.**
   With Java's multi-threaded feature it is possible to write programs that can do many tasks simultaneously. This design feature allows developers to construct smoothly running interactive applications.

8. **High performance**
Because of the intermediate bytecode java language provides high performance

9. **Dynamic and extendible.**
Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

Java Development kits(java software:jdk1.6): Java development kit comes with a number of Java development tools. They are:

(1) Appletviewer: Enables to run Java applet.
(2) javac: Java compiler.
(3) java : Java interpreter.
(4) javah : Produces header files for use with native methods.
(5) javap : Java disassembler.
(6) javadoc : Creates HTML documents for Java source code file.
(7) jdb : Java debugger which helps us to find the error.

Java Building and running Process:

1. **Open the notepad and type the below program**

Simple Java program:

Example:

```
class Sampleone
{
        public static void main(String args[])
        {
                System.out.println("Welcome to JAVA");
        }
}
```

Description:

(1) **Class declaration:** "class sampleone" declares a class, which is an object-oriented construct. Sampleone is a Java identifier that specifies the name of the class to be defined.

(2) **Opening braces:** Every class definition of Java starts with opening braces and ends with matching one.

(3) **The main line:** the line " public static void main(String args[]) " defines a method name main. Java application program must include this main. This is the starting point of the interpreter from where it starts executing. A Java program can have any number of classes but only one class will have the main method.

(4) **Public:** This key word is an access specifier that declares the main method as unprotected and therefore making it accessible to the all other classes.

(5) **Static:** Static keyword defines the method as one that belongs to the entire class and not for a particular object of the class. The main must always be declared as static.

(6) **Void:** the type modifier void specifies that the method main does not return any value.

(7) **The println:** It is a method of the object out of system class. It is similar to the printf or cout of c or c++.

2. Save the above program with .java extension, here file name and class name should be same, ex: Sampleone.java,
3. Open the command prompt and **Compile** the aboveprogram

   javac Sampleone.java

   From the above compilation the java compiler produces a bytecode(.class file)
4. Finally run the program through the **interpreter**

   java Sapleone.java

Output of the program:

Welcome to JAVA

Implementing a Java program: Java program implementation contains three stages. They are:

1. Create the source code.
2. Compile the source code.
3. Execute the program.

(1) Create the source code:

1. Any editor can be used to create the Java source code.
2. After coding the Java program must be saved in a file having the same name of the class containing main() method.
3. Java code file must have .Java extension.

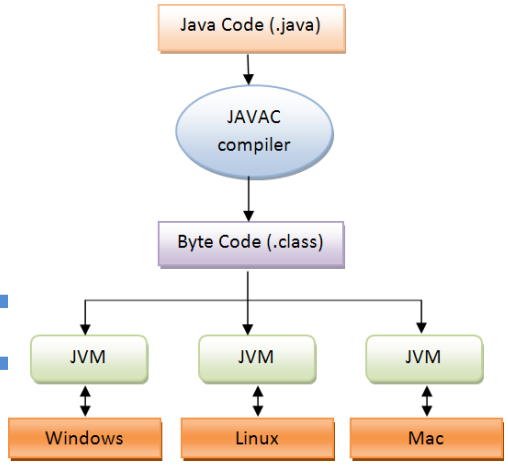
(2) Compile the source code:

1. Compilation of source code will generate the bytecode.
2. JDK must be installed before completion.
3. Java program can be compiled by typing javac <filename>.java
4. It will create a file called <filename>.class containing the bytecode.

(3) Executing the program:

1. Java program once compiled can be run at any system.
2. Java program can be execute by typing Java <filename>

JVM(Java Virtual Machine)



The concept of Write-once-run-anywhere (known as the Platform independent) is one of the important key feature of java language that makes java as the most powerful language. Not even a single language is idle to this feature but java is closer to this feature. The programs written on one platform can run on any platform provided the platform must have the JVM(**Java Virtual Machine**). A Java virtual machine (JVM) is a virtual machine that can execute Java bytecode. It is the code execution component of the Java software platform.

More Examples:

Java program with multiple lines:

Example:

```
import java.lang.math;
class squreroot
{
        public static void main(String args[])
        {
                double x = 5;
                double y;
                y = Math.sqrt(x);
                System.out.println("Y = " + y);
        }
}
```

Java Program structure: Java program structure contains six stages. They are:

(1) Documentation section: The documentation section contains a set of comment lines describing about the program.

(2) Package statement: The first statement allowed in a Java file is a package statement. This statement declares a package name and informs the compiler that the class defined here belong to the package.

```
Package student;
```


(3) Import statements: Import statements instruct the compiler to load the specific class belongs to the mentioned package.

```
Import student.test;
```

(4) Interface statements: An interface is like a class but includes a group of method deceleration. This is an optional statement.

(5) Class definition: A Java program may contain multiple class definition The class are used to map the real world object.

(6) Main method class: The main method creates objects of various classes and establish communication between them. On reaching to the end of main the program terminates and the control goes back to operating system.

Java command line arguments: Command line arguments are the parameters that are supplied to the application program at the time when they are invoked. The main() method of Java program will take the command line arguments as the parameter of the args[ ] variable which is a string array.

Example:

```
Class Comlinetest
{
        public static void main(String args[ ] )
        {
                int count, n = 0;
                string str;
                count = args.length;
                System.out.println ( " Number of arguments :" + count);
                While ( n < count )
                {
                        str = args[ n ];
                        n = n + 1;
                        System.out.println( n + " : " + str);
                }
        }
}
```

Run/Calling the program:

```
javac Comlinetest.java
java Comlinetest Java c cpp fortran
```

Output:

```
1 : Java
2 : c
3 : cpp
4 : fortran
```


Java API:

Java standard library includes hundreds of classes and methods grouped into several functional packages. Most commonly used packages are:

(a) Language support Package.
(b) Utilities packages.
(c) Input/output packages
(d) Networking packages
(e) AWT packages.
(f) Applet packages.

Java Tokens

Constants: Constants in Java refers to fixed value that do not change during the execution of program. Java supported constants are given below:

(1) **Integer constants:** An integer constant refers to a sequence of digits. There are three types of integer namely decimal integer, octal integer and hexadecimal integer. For example: 123 -321

(2) **Real constants:** Any real world number with a decimal point is known as real constants. For example : 0.0064 12e-2 etc.

(3) **Single character constants:** A single character constant contains a single character enclosed with in a pair of single quotes. For ex: „m' „5"

(4) **String constants :** A string constant is a sequence of character enclosed with double quotes. For ex: "hello" "java" etc.

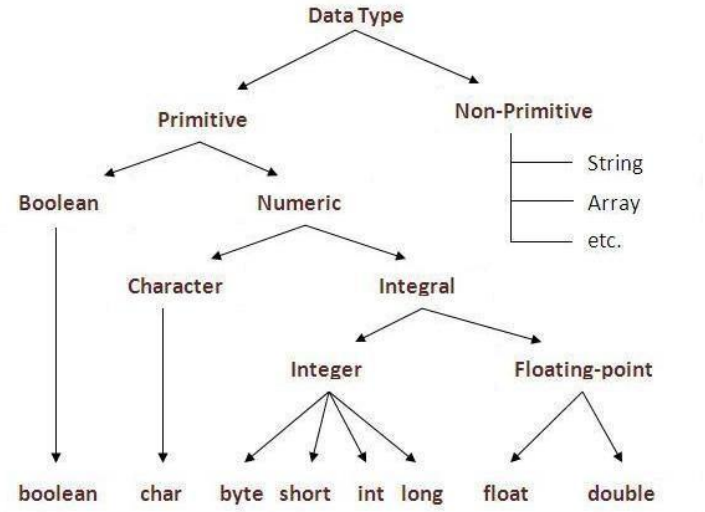
(5) **Backslash character constants:** Java supports some backslash constants those are used in output methods. They are :

1. \b	Backspace
2. \f	Form feed
3. \n	New Line
4. \r	Carriage return.
5. \t	Horizontal tab.
6. \'	Single quotes.
7. \"	Double quotes
8. \\	Back slash

Data Types in Java:

In java, data types are classified into two catagories :

1. Primitive Data type
2. Non-Primitive Data type



Data Type	Default Value	Default size
boolean	False	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Integers Type: Java provides four types of Integers. They are byte, sort, Int, long. All these are sign, positive or negative.

Byte: The smallest integer type is byte. This is a signed 8-bit type that has a range from –128 to 127. Bytes are useful for working with stream or data from a network or file. They are also useful for working with raw binary data. A byte variable is declared with the keyword "byte".

```
byte b, c;
```

Short: Short is a signed 16-bit type. It has a range from –32767 to 32767. This data type is most rarely used specially used in 16 bit computers. Short variables are declared using the keyword short.

```
short a, b;
```

int: The most commonly used Integer type is int. It is signed 32 bit type has a range from –2147483648 to 2147483648.

```
int a, b, c;
```

long: Long is a 64 bit type and useful in all those occasions where Int is not enough. The range of long is very large.

long a, b;

Floating point types: Floating point numbers are also known as real numbers are useful when evaluating a expression that requires fractional precision. The two floating-point data types are float and double.

float: The float type specifies a single precision value that uses 32-bit storage. Float keyword is used to declare a floating point variable.

float a, b;

double: Double DataTips is declared with **double** keyword and uses 64-bit value.

Characters: The Java data type to store characters is char. **char data type** of Java uses Unicode to represent characters. Unicode defines a fully international character set that can have all the characters of human language. Java char is 16-bit type. The range is 0 to 65536.

Boolean: Java has a simple type called **boolean** for logical values. It can have only one of two possible values. They are true or false.

Key Words: Java program is basically a collection of classes. A class is defined by a set of declaration statements and methods containing executable statements. Most statement contains an expression that contains the action carried out on data. The compiler recognizes the tokens for building up the expression and statements. Smallest individual units of programs are known as tokens. Java language includes five types of tokens. They are

(a) Reserved Keyword
(b) Identifiers
(c) Literals.
(d) Operators
(e) Separators.

(1) **Reserved keyword:** Java language has 60 words as reserved keywords. They implement specific feature of the language. The keywords combined with operators and separators according to syntax build the Java language.

(2) **Identifiers:** Identifiers are programmer-designed token used for naming classes methods variable, objects, labels etc. The rules for identifiers are

1. They can have alphabets, digits, dollar sign and underscores.
2. They must not begin with digit.
3. Uppercase and lower case letters are distinct.
4. They can be any lengths.
5. Name of all public method starts with lowercase.

6. In case of more than one word starts with uppercase in next word.
7. All private and local variables use only lowercase and underscore.
8. All classes and interfaces start with leading uppercases.
9. Constant identifier uses uppercase letters only.

(3) **Literals:** Literals in Java are sequence of characters that represents constant values to be stored in variables. Java language specifies five major types of Literals. They are:

1. Integer Literals.
2. Floating-point Literals.
3. Character Literals.
4. String Literals.
5. Boolean Literals.

(4) **Operators:** An operator is a symbol that takes one or more arguments and operates on them to produce an result.

(5) **Separators:** Separators are the symbols that indicates where group of code are divided and arranged. Some of the operators are:

1. Parenthases()
2. Braces{ }
3. Brackets [ ]
4. Semicolon ;
5. Comma ,
6. Period .

Java character set: The smallest unit of Java language are its character set used to write Java tokens. This character are defined by unicode character set that tries to create character for a large number of character worldwide.

The Unicode is a 16-bit character coding system and currently supports 34,000 defined characters derived from 24 languages of worldwide.

Variables: A variable is an identifier that denotes a storage location used to store a data value. A variable may have different value in the different phase of the program. To declare one identifier as a variable there are certain rules. They are:

1. They must not begin with a digit.
2. Uppercase and lowercase are distinct.

3. It should not be a keyword.
4. White space is not allowed.

Declaring Variable: One variable should be declared before using. The syntax is

Data-type variblaname1, variablename2, ... variablenameN;

Initializing a variable: A variable can be initialize in two ways. They are

(a) Initializing by Assignment statements.
(b) Initializing by Read statements.

Initializing by assignment statements: One variable can be initialize using assignment statements. The syntax is :

Variable-name = Value;

Initialization of this type can be done while declaration.

Initializing by read statements: Using read statements we can get the values in the variable.

Scope of Variable: Java variable is classified into three types. They are

(a) Instance Variable
(b) Local Variable
(c) Class Variable

Instance Variable: Instance variable is created when objects are instantiated and therefore they are associated with the object. They take different values for each object.

Class Variable: Class variable is global to the class and belongs to the entire set of object that class creates. Only one memory location is created for each class variable.

Local Variable: Variable declared inside the method are known as local variables. Local variables are also can be declared with in program blocks. Program blocks can

be nested. But the inner blocks cannot have same variable that the outer blocks are having.

Arrays in Java

Array which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Declaring Array Variables:

To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable:

dataType[] arrayRefVar; or dataType arrayRefVar[];

Example:

The following code snippets are examples of this syntax:

int[] myList; or int myList[];

Creating Arrays:

You can create an array by using the new operator with the following syntax:

arrayRefVar = new dataType[arraySize];

The above statement does two things:

- It creates an array using new **dataType[arraySize];**
- It assigns the reference of the newly created array to the variable **arrayRefVar**.

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below:

dataType[] arrayRefVar = new dataType[arraySize];

Alternatively you can create arrays as follows:

dataType[] arrayRefVar = {value0, value1, ..., valuek};

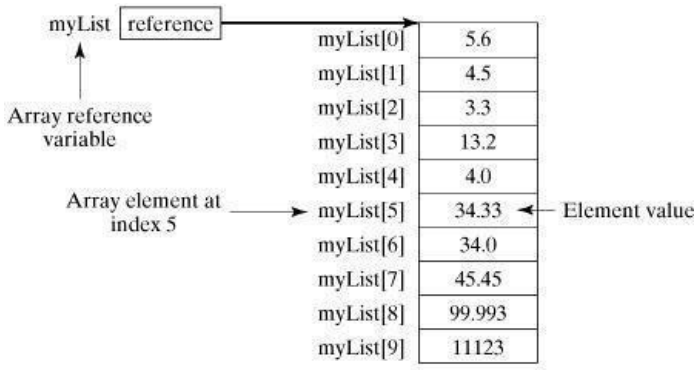
The array elements are accessed through the **index**. Array indices are 0-based; that is, they start from 0 to **arrayRefVar.length-1**.

Example:

Following statement declares an array variable, myList, creates an array of 10 elements of double type and assigns its reference to myList:

```
double[] myList = new double[10];
```

Following picture represents array myList. Here, myList holds ten double values and the indices are from 0 to 9.



Processing Arrays:

When processing array elements, we often use either for loop or foreach loop because all of the elements in an array are of the same type and the size of the array is known.

Example:

Here is a complete example of showing how to create, initialize and process arrays:

```
public class TestArray
{
        public static void main(String[] args)
        {
                double[] myList = {1.9, 2.9, 3.4, 3.5};

                // Print all the array elements
                for (int i = 0; i < myList.length; i++) {
                System.out.println(myList[i] + " ");
        }
```

```
      // adding all elements
            double total = 0;
            for (int i = 0; i < myList.length; i++)
            {
                  total += myList[i];
            }
     System.out.println("Total is " + total);
     // Finding the largest element
     double max = myList[0];
     for (int i = 1; i < myList.length; i++)
    {
       if (myList[i] > max)
       max = myList[i];
     }
     System.out.println("Max is " + max);
  }
}
```

This would produce the following result:

```
1.9
2.9
3.4
3.5
Total is 11.7
Max is 3.5
```


The foreach Loop

JDK 1.5 introduced a new for loop known as for-each loop or enhanced for loop, which enables you to traverse the complete array sequentially without using an index variable.

Example:

The following code displays all the elements in the array myList:

```
public class TestArray {

  public static void main(String[] args) {
    double[] myList = {1.9, 2.9, 3.4, 3.5};

    // Print all the array elements
```

```
    for (double element: myList)
    {
       System.out.println(element);
    }
  }
}
```

This would produce the following result:

```
1.9
2.9
3.4
3.5
```

Type Casting: It is often necessary to store a value of one type into the variable of another type. In these situations the value that to be stored should be casted to destination type. Type casting can be done in two ways.

Type Casting

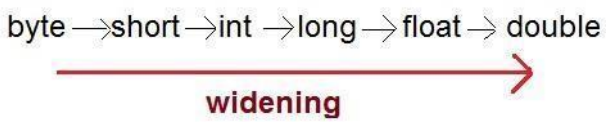
Assigning a value of one type to a variable of another type is known as **Type Casting**.

Example :

```
int x = 10;
byte y = (byte)x;
```

In Java, type casting is classified into two types,

- Widening Casting(Implicit)



- Narrowing Casting(Explicitly done)

Widening or Automatic type converion

Automatic Type casting take place when,

- the two types are compatible
- the target type is larger than the source type

Example :

```
public class Test
{
    public static void main(String[] args)
    {
      int i = 100;
      long l = i;       //no explicit type casting required
      float f = l;      //no explicit type casting required
      System.out.println("Int value "+i);
      System.out.println("Long value "+l);
      System.out.println("Float value "+f);
    }
}
```


Output :

```
Int value 100
Long value 100
Float value 100.0
```

Narrowing or Explicit type conversion

When you are assigning a larger type value to a variable of smaller type, then you need to perform explicit type casting.

Example :

```
public class Test
{
    public static void main(String[] args)
    {
      double d = 100.04;
```

```
        long l = (long)d;  //explicit type casting required
        int i = (int)l;  //explicit type casting required

        System.out.println("Double value "+d);
        System.out.println("Long value "+l);
        System.out.println("Int value "+i);

    }

}
```

Output :

```
Double value 100.04
Long value 100
Int value 100
```

Java operators:

Java operators can be categorized into following ways:

(1) Arithmetic operator
(2) Relational operator
(3) Logical operator
(4) Assignment operator
(5) Increment and decrement operator
(6) Conditional operator
(7) Bitwise operator
(8) Special operator.

Arithmetic operator: The Java arithmetic operators are:

+	: Addition
-	: Subtraction
•	: Multiplication
/	: Division
%	: Remainder

Relational Operator:

<	: Is less then
<=	: Is less then or equals to
➢	: Is greater then
>=	: Is grater then or equals to
==	: Is equals to
!=	: Is not equal to

Logical Operators:

&&	: Logical AND
||	: Logical OR

! : Logical NOT

Assignment Operator:

+= : Plus and assign to
-= : Minus and assign to
*= : Multiply and assign to.
/= : Divide and assign to.
%= : Mod and assign to.
= : Simple assign to.

Increment and decrement operator:

++ : Increment by One {Pre/Post)
-- : Decrement by one (pre/post)

Conditional Operator: Conditional operator is also known as ternary operator. The conditional operator is :

Exp1 ? exp2 : exp3

Bitwise Operator: Bit wise operator manipulates the data at Bit level. These operators are used for tasting the bits. The bit wise operators are:

& : Bitwise AND
! : Bitwise OR
^ : Bitwise exclusive OR
~ : One"s Complement.
<< : Shift left.
>> : Shift Right.
>>> : Shift right with zero fill

Example:

Bitwise operator works on bits and performs bit-by-bit operation. **Assume if a = 60; and b = 13;** now in binary format they will be as follows:

a = 0011 1100
b = 0000 1101
---
a&b = 0000 1100
a|b = 0011 1101
a^b = 0011 0001
~a = 1100 0011

<<	**Binary Left Shift Operator.** The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	**Binary Right Shift Operator**. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>>	**Shift right zero fill Operator**. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

Special Operator:

Instanceof operator: The instanceof operator is a object refrence operator that returns true if the object on the right hand side is an instance of the class given in the left hand side. This operator allows us to determine whether the object belongs to the particular class or not.

Person instanceof student

The expression is true if the person is a instance of class student.

Dot operator: The dot(.) operator is used to access the instance variable or method of class object.

Example Programs

```
class arithmeticop
{
	public static void main(String args[])
	{
		float a=20.5f;
		float b=6.4f;
		System.out.println("a = " + a);
		System.out.println("b = " + b );
		System.out.println("a + b = " + (a+b));
	}
}
```



```
class Bitlogic
{
	public static void main(String args[])
	{
String binary[] = {"0000","0001","0010","0011","0100","0101","0110","0111","1000","1001","1010",
                   "1011","1100","1101","1110","1111"};
		int a = 3;
		int b = 6;
		int c = a | b;
		int d = a & b;
		int e = a ^ b;
		int f = (~a&b)|(a & ~b);
		System.out.println("a or b :"+binary[c]);
		System.out.println("a and b : "+binary[d]);
		System.out.println("a xor b : "+binary[e]);
		System.out.println("(~a&b)|(a & ~b) :  "+binary[f]);
	}
}
```

Control Statements

Decision making statements:

1. **Simple If statement:**

The general form of single if statement is :

```
If ( test expression)
{
        statement-Block;
}
statement-Blocks;
```

2. **If- Else statement:**

The general form of if-else statement is

```
If ( test expression)
{
        statement-block1;
}
else
{
        statement-block2
}
```

3. **Else-if statement:**

The general form of else-if statement is:

```
If ( test condition)
{
        statement-block1;
}
else if(test expression2)
{
        statement-block2;
}
else
{
        statement block3;
}
```

4. **Nested if – else statement:**

The general form of nested if-else statement is:

```
If ( test condition)
{
        if ( test condition)
        {
                statement block1;
        }
        else
        {
```

```
                    statement block2;
                }
            }
            else
            {
                statement block 3
            }
```

5. **The switch statements:**

The general form of switch statement is:

```
Switch ( expression)
{
    case value-1:
        block-1;
        break;
    case value-2:
        block-2;
        break;
    . . . . . .
    default:
        default block;
        break;
}
```

Loops In Java: In looping a sequence of statements are executed until a number of time or until some condition for the loop is being satisfied. Any looping process includes following four steps:

(1) Setting an initialization for the counter.
(2) Execution of the statement in the loop
(3) Test the specified condition for the loop.
(4) Incrementing the counter.

Java includes three loops. They are:

(1) While loop:

The general structure of a while loop is:

```
Initialization
While (test condition)
{
    body of the loop
}
```

(2) Do loop:

The general structure of a do loop is :

```
Initialization
do
{
    Body of the loop;
}
while ( test condition);
```

(3) For loop :

The general structure of for loop is:

```
For ( Initialization ; Test condition ; Increment)
{
        body of the loop;
}
```

More about Loops:

Break Statement: Using "break" statement we can jump out from a loop. The "break" statement will cause termination of the loop.

Continue statement: The "continue" statement will cause skipping some part of the loop.

Labeled loops: We can put a label for the loop. The label can be any Java recognized keyword. The process of giving label is

```
Label-name : while (condition)
                {
                        Body ;
                }
```

Example Program for label break statement


```
class BreakTest
{
    public static void main(String args[])
    {
        boolean t= true;
        first:
        {
            second:
            {
                third:
                {
                    System.out.println("Third stage");
                    if(t)
                        break second;
                        System.out.println("Third stage complete");
                }
                System.out.println("Second stage");
            }
            System.out.println("First stage");
        }
    }
```

```
}
```

Example Program for continue statement

```
class ContinueTest
{
	public static void main(String args[])
	{
		outer: for(int i=0;i<10;i++)
			{
				for(int j = 0; j<10;j++)
				{
					if(j>i)
					{
						System.out.println("\n");
						continue outer;
					}
					System.out.print("  "+(i*j));
				 }
			 }

	//System.out.println(" ");
	}
}
```


Example Program for return statement

```
class ReturnTest
{
	public static void main(String args[])
	{
		boolean t = true;
		System.out.println("Before the return");
		if(t) return;
		System.out.println("After return");
	}
}
```

Java Access Specifiers/Modifiers

Private Access Modifier - private:

Methods, Variables and Constructors that are declared private can only be accessed within the declared class itself.

Private access modifier is the most restrictive access level. Class and interfaces cannot be private.

Using the private modifier is the main way that an object encapsulates itself and hides data from the outside world.

Public Access Modifier - public:

A class, method, constructor, interface etc declared public can be accessed from any other class. Therefore fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.

However if the public class we are trying to access is in a different package, then the public class still need to be imported.

Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.

Protected Access Modifier - protected:

Variables, methods and constructors which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.

The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected.

Default Access Modifier - No keyword:

Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.A variable or method declared without any access control

modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public.

Advantages of JAVA:

- It is an open source, so users do not have to struggle with heavy license fees each year.
- Platform independent.
- Java API's can easily be accessed by developers.
- Java perform supports garbage collection, so memory management is automatic.
- Java always allocates objects on the stack.
- Java embraced the concept of exception specification.
- Multi-platform support language and support for web-services.
- Using JAVA we can develop dynamic web applications.
- It allows you to create modular programs and reusable codes.

Questions

1. List & explain the characteristics features of java language. **(10 Marks)**
2. Briefly discuss about the java development tool kit. **(07 Marks)**
3. Explain the process of building and running java application program **(05Marks)**.
4. Explain the following: a)JVM b)Type casting. **(05Marks)**
5. Class Example{

```
public static void main(String s[]) {
        int a;
        for(a=0;a<3;a++){
        int b=-1;
        System.out.println(" "+b);
         b=50;
        System.out.println(" "+b);
```

```
        }
    }}
```

What is the output of the above code? If you insert another „int b" outside the for loop what is the output. **(05Marks)**

6. With example explain the working of >> and >>>. **(06Marks)**
7. What is the default package & default class in java? **(02Marks)**
8. Write a program to calculate the average among the elements {4, 5, 7, 8}, using for each in java. How for each is different from for loop? **(07Marks)**
9. Briefly explain any six key consideration used for designing JAVA language.**(06Marks)**
10. Discuss three OOP principles. **(06Marks)**
11. How compile once and run anywhere is implemented in JAVA language? **(04Marks)**
12. List down various operators available in JAVA language. **(04Marks)**
13. What is polymorphism?explain with an example. **(04Marks)**
14. Explain the different access specifiers in java, with examples. **(06Marks)**
15. a)

```
int num,den;
  if(den!=0&&num|den>2)
{
}
```

b)

```
int num,den;
  if(den!=0&num|den==2)
{
}
```

Compare & explain the above two snippets. **(02Marks)**

16. Write a note on object instantiation. **(02Marks)**
17. Explain type casting in JAVA
18. With a program explain break, continue and return keyword in java