Part II : Managing Input/Output Files

UNIT - I

1 JAVA Fundamentals

Syllabus

Review of Object oriented concepts, Evolution of Java, Comparison of Java with other programming languages, Java features, Java and World Wide Web, Java Run Time Environment. JVM architecture. Overview of Java Language, Simple Java Program, Java Program Structure. Installing and Configuring Java.

Java Tokens, Java Statements, Constants, variables, data types. Declaration of variables, Giving values to variables, Scope of variables, arrays, Symbolic constants, Typecasting, Getting values of variables, Standard default values, Operators, Expressions, Type conversion in expressions, Operator precedence and associatively, Mathematical functions, Control statements- Decision making & looping.

Contents

1.1 Review of Object Oriented Concepts

Object Oriented Programming (OOP) is a programming language model organized around "objects" rather than "actions".

1.1.1 Fundamentals of Object-Oriented Programming

Various characteristics of object oriented programming are -

1.1.1.1 Object

- Object is an instance of a class.
- Objects are basic run-time entities in object oriented programming.
- In C++ the class variables are called objects. Using objects we can access the member variable and member function of a class.
- Object represent a person, place or any item that a program handles.
- For example - If the class is country then the objects can be India, China, Japan, U.S.A and so on.
- A single class can create any number of objects.
- **Declaring objects -**

  The syntax for declaring object is –

  ```
  Class_Name Object_Name;
  ```

- **Example**

  ```
  Fruit f1;
  ```

For the class **Fruit** the object **f1** can be created.

1.1.1.2 Classes

- A class can be defined as an entity in which data and functions are put together.
- The concept of class is similar to the concept of **structure** in C.
- **Syntax of class** is as given below

```
class name_of_class
{
    private :
        variables declarations;
        function declarations;
    public :
        variable declarations;
        function declarations;
} ; ← do not forget semicolon
```

- **Example**

```
class rectangle
{
        private :
                int len, br;
        public :
                void get_data ( ) ;
                void area( );
                void print_data ( );
};
```

- **Explanation**
  - The class declared in above example is **rectangle**.
  - The class name must be preceded by the keyword **class**.
  - Inside the body of the class there are two keywords used **private** and **public**. These are called **access specifiers**.
- In Java the class can be written as

```
class <class_name>{
        Data Members;
        method;
}
```

1.1.1.3 Data Members

- The data members are the **variables** that are declared within the class.
- These members are declared along with the data types.
- The access specifier to these members can be **public**, **private** or **protected**.
- These data members can be accessible by the **main()** function using the object of a class.

1.1.1.4 Methods and Messages

- Object is an instance of a class. Every object consists of both data attributes and methods. The data attributes of every object are manipulated by the methods. These objects in the program communicate with each other by sending messages.
- A message for an object is nothing but the request for the execution of the procedure. Thus whenever the object wants to communicate it invokes a method. The procedure or method receives the information passed by an object and it generates the result. Refer Fig. 1.1.1.

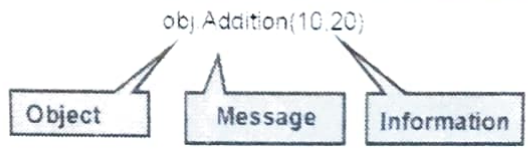


Fig. 1.1.1 : Message

1.1.1.5 Data Encapsulation

- Encapsulation is for the detailed implementation of a component which can be hidden from rest of the system.
- **Definition** : Encapsulation means binding of data and method together in a **single entity called class**.
- The data inside that class is accessible by the function in the same class. It is normally not accessible from the outside of the component.

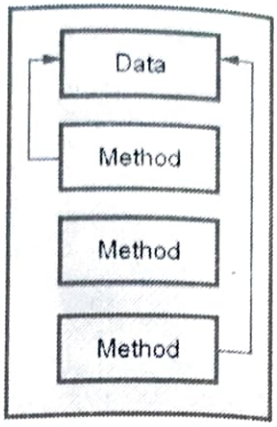


Fig. 1.1.2 : Concept of encapsulation

1.1.1.6 Data Abstraction and Information Hiding

- Data Abstraction is the process of eliminating unimportant details of a class. In this process only important properties are highlighted. Thus information hiding can be possible.
- Information Hiding is hiding the data which is being affected by that implementation. Generally by declaring the data member as **private**, we can protect it by unauthorized access.
- Data abstraction is independent upon object data type.
- It is used in software design phase.
- Data abstraction is represented by using abstract classes.

1.1.1.7 Inheritance

- **Definition :** Inheritance is a property by which the new classes are created using the old classes. In other words the new classes can be developed using some of the properties of old classes.
- Inheritance support hierarchical structure.
- The old classes are referred as **base classes** and the new classes are referred as **derived classes.** That means the derived classes inherit the properties (data and functions) of base class.
- **Example :**

Here the **Shape** is a base class from which the **Circle, Line** and **Rectangle** are the derived classes. These classes inherit the functionality draw() and resize(). Similarly the Rectangle is a base class for the derived class Square. Along with the derived properties the derived class can have its own properties. For example the class Circle may have the function like backgrcolor() for defining the back ground color.

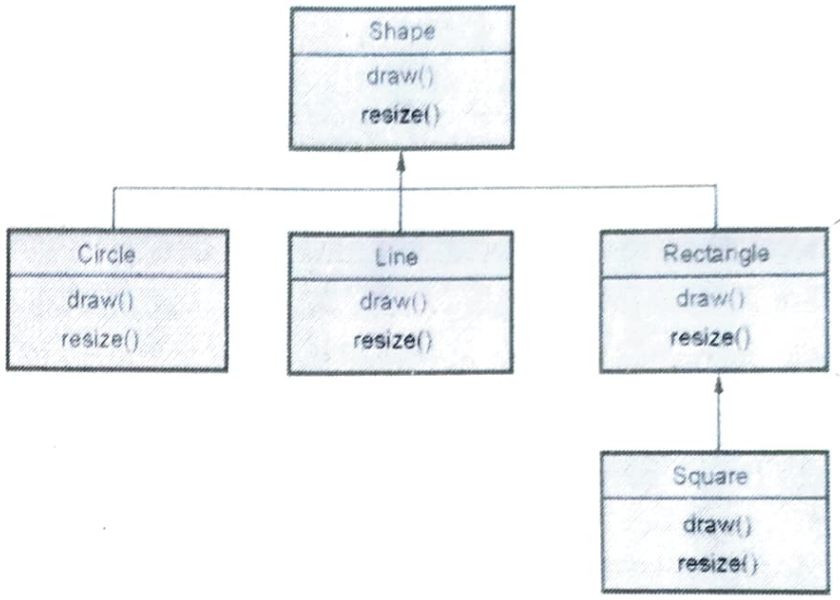


Fig. 1.1.3 : Hierarchical structure of inheritance

1.1.1.8 Polymorphism

- Polymorphism means many structures.
- **Definition :** Polymorphism is the ability to take **more than one form** and refers to an operation exhibiting different behavior in different instances (situations).
- The behavior depends on the type of data used in the operation. It plays an important role in allowing objects with different internal structures to share the same external interface.
- Without polymorphism, one has to create separate module names for each method.
- For example the method clean is used to **clean** a **dish** object, one that cleans a **car** object, and one that cleans a **vegetable object**.
- With polymorphism, you create a single "clean" method and apply it for different objects.

1.1.1.9 Static and Dynamic Binding

- Connecting a method call to method body is called **binding**.
- There are two types of bindings - (1) Static binding and (2) Dynamic binding.
- The binding which can be resolved at compile time by compiler is known as **static or early binding**. In Java the methods written using keywords **static**, **private** and **final** methods are resolved at compile-time.
- In Dynamic binding compiler doesn't decide the method to be called. **Method overriding** is an example of dynamic binding.

1.1.1.10 Message Passing

- **Definition** : Message passing is a mechanism by which the objects interact with each other. The process of interaction of objects is as given below -
  1. Define the class with data members and member functions.
  2. Create the objects belonging to the class.
  3. Establish the communication between these objects using the methods.
- Object send information to each other using the methods. This process is called as **message passing**.

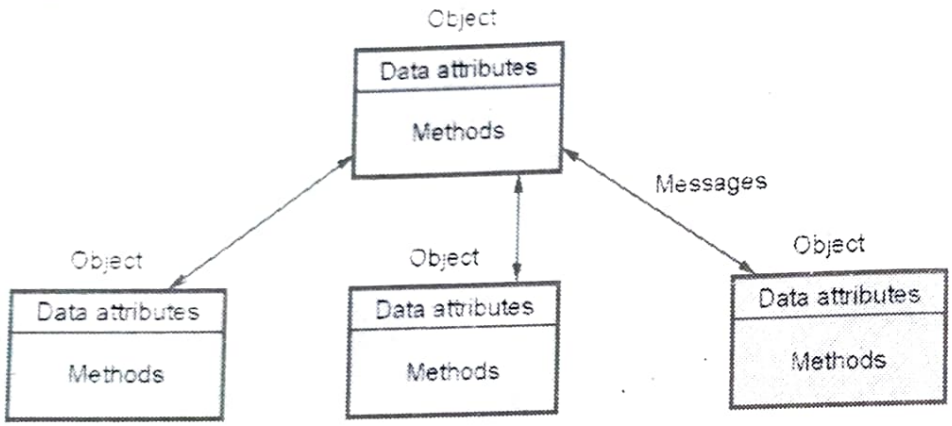


Fig. 1.1.4 : Methods and messages

Review Question

1. *State and explain various characteristics of object oriented programming.*

1.2 Evolution of Java

- Java was invented by James Gosling, Patrick Naughton, Chris Warth, Ed Frank and Mike Sheridan at Sun Microsystems, Inc. in 1991. It took 18 months to complete the first working version of Java.
- This language was initially called as **Oak** but was renamed as Java in 1995.
- Following table shows the evolution of Java language.

Period	Events took place
June -1991	The project for Java language was initiated by James Gosling, Patrick Naughton, Chris Warth, Ed Frank and Mike Sheridan at Sun Microsystems, Inc.
1992	This language was officially known as Oak.
1995	Sun Microsystems released the first public implementation as Java 1.0

1998 - 1999	The Java 2.0 was released. In this version there was a support for various platforms such as J2EE, J2ME.
2006	For marketing purposes, Sun renamed to J2 versions as Java EE, Java ME and Java SE.
2007	Sun released Java as open source software.
2010	Oracle acquired Sun Microsystems.

1.3 Comparison of Java with other Programming Languages

Sr. No.	C	C++	Java
1.	The C language needs to be **compiled**.	The C++ is a language that needs to be **compiled.**	The Java is a language that gets **interpreted** and **compiled**. JVM
2.	The C is platform **dependant**.	C++ is platform **dependant**.	Java is a **platform independent**.
3.	There is no concept of **threading** in C.	C++ does not support multi - threading programming	Java supports **multi-threading**
4.	Using C, the GUI applications cannot be created.	C++ does not have facility to create and implement the graphical user interface.	Using Java, one can design very interactive and **user friendly graphical user interface**.
5.	C does not support database oriented applicaiton.	Database handling using C++ is very complex. The C++ does not allow the database connectivity.	**Java servlets** can be created which can interact with the **databases** like MS-ACCESS, Oracle, My-SQL and so on.
6.	C cannot be embedded in scripting language.	C++ code can not be embedded in any scripting language.	Java code can be **embedded within a scripting language** by means of Applet programming.
7.	There is no concept of **inheritance**.	C++ supports **multiple inheritance**.	Java does **not support multiple inheritance** however it makes use of interface.
8.	C uses **pointers**.	In C++ we can use the **pointers**	In Java there is **no concept of pointers**.

9.	C does not support tempelate.	C++ supports **templates**	Java does not support the concept of templates.
10.	There is no **class** in C.	In C++ we can write a program without a class	In Java, there must be **at least one class** present.
11.	C is the most simple.	C++ is simple to use and implement.	Java is **safe** and more **reliable**.
12.	C can be compiled on variety of compilers.	C++ can be compiled with variety of compilers.	Java can be compiled using an unique compiler.

1.4 Java Features

Following are the features of Java which makes it as a revolutionary programming language -

1. Java can be compiled and interpreted

- Normally programming languages can be either compiled or interpreted but Java is a language which can be compiled as well as interpreted.
- First, Java compiler translates the Java source program into a special code called **bytecode**.
- Then Java interpreter interprets this bytecode to obtain the equivalent machine code. This machine code is then directly executed to obtain the output.

2. Java is a platform independent and portable programming language

- Platform independence is the most exciting feature of Java program. That means programs in Java can be executed on variety of platforms.
- This feature is based on the goal – *write once, run anywhere and at anytime forever.*
- Java supports portability in 2 ways - Java compiler generates the byte code which can be further used to obtain the corresponding machine code. Secondly the primitive data types used in Java are machine independent.

3. Java is known as an object oriented programming language

- Java is a true object oriented language as everything in java is an object.
- In Java, all the code and data lies within the classes and object.

4. Java is robust and secure

- Java ensures the reliable code.
- The data types are strictly checked at the compile time as well as run time in Java.

- The memory management technique is supported by garbage collection technique. This technique eliminates various memory management problems.
- The exception handling feature of Java helps the programmer to handle the serious errors delicately without crashing the overall system.

5. Java is a designed for distributed systems

- This feature is very much useful in networking environment.
- In Java, two different objects on different computers can communicate with each other.
- This can be achieved by Remote Method Invocation(RMI). This feature is very much useful in Client-Server communication.

6. Java is simple and small programming language

- Java is very simple programming language. Even though we have no programming background, we can learn this language very easily.
- The programmers who have worked on C++ can learn this language very efficiently.

7. Java is a multi-threaded and interactive language

- Java supports multi-threaded programming which allows a programmer to write such a program that can perform many tasks simultaneously.
- This allows the programmer to develop the interactive systems.

8. Java is known for its high performance, scalability, monitoring and manageability

- Due to the use of **bytecode** the Java has high performance.
- The use of multi - threading also helps to improve the performance of the Java.
- The J2SE helps to increase the scalability in Java. For monitoring and management Java has large number of Application Programming Interfaces(API).
- There are tools available for monitoring and tracking the information at the application level.

9. Java is a dynamic and extensible language

- This language is capable of dynamically linking new class libraries, methods and objects.
- Java also supports the functions written in C and C++. These functions are called native methods.

10. Java can be developed with ease

There are various features of Java such as **Generics, static import, annotations** and so on which help the Java programmer to create a error free **reusable code**.

1.5 Java and World Wide Web

- World wide web is an information retrieval system, designed for internetworking environment. The information in this system is accessed through the web pages. User can navigate from one web page to another in search of the information the web.
- There are various scripting languages used for designing the web pages. The most commonly used scripting language is HTML i.e. HyperText Markup Language.
- Java can be easily used on the Web systems. With the use of Java programs, the Web is becoming more interactive and dynamic. Similarly with the use of Web we can run the particular Java program on any computer system.
- Java **communicates with Web** using Java applets. The communication between Java applet and Web occurs as follow -
  1. The user sends the request for the desired web page using web browser (For example Internet Explorer, Mozilla Firefox, and so on) to the web server. The web server accepts this request, process it and sends the required document to the user.
  2. The web page in the form of HTML tags is returned to the user in his browser. This HTML page contains the **<applet>** tag which represents that the corresponding web page is an applet.
  3. While processing the applet, the server converts the applet source code and HTML document in the form of byte code.
  4. The server also transfers the applet byte code to the user's computer.
  5. The Java enabled web browser interprets the bytecode and provides the output.
  6. Now user can have interaction with the applet.

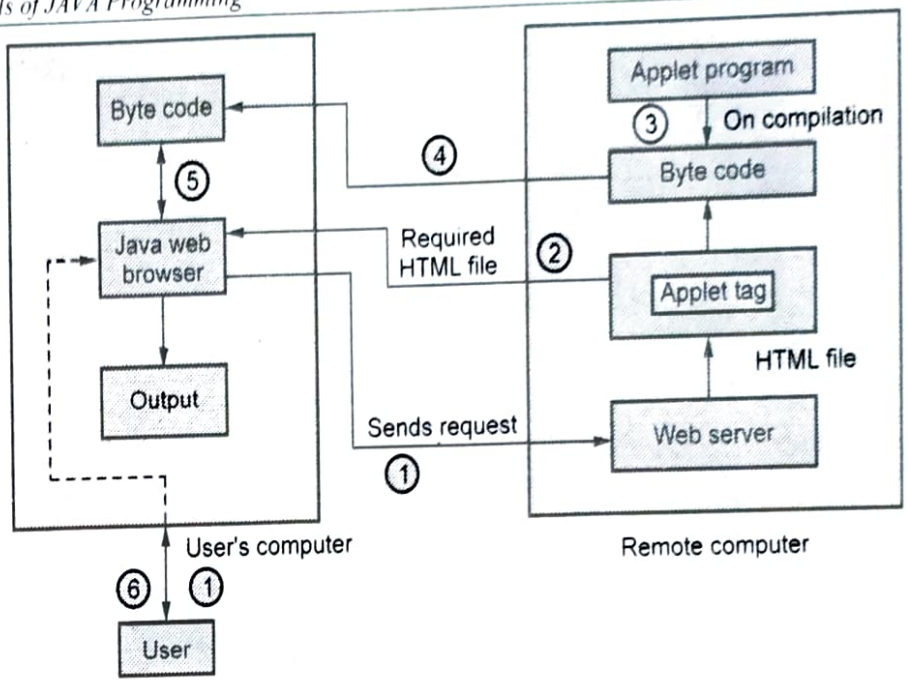


Fig. 1.5.1 Communication between Java and Web

1.6 Java Run Time Environment

- The Java environment is made up of three things - development tools, classes and methods and Java Runtime Environment (JRE).
- The java development tools constitute **Java Development Kit(JDK)**.
- The classes and methods are called **Application Programming Interface(API)** which forms a library called **Java Standard Library(JSL)**
- The java runtime environment is supported by **Java Virtual Machine (JVM)**.

1.6.1 Java Development Kit

- The Java Development Kit is nothing but a collection of tools that are used for development and runtime programs.
- It consists of -
  1. **javac** - The Java compiler which translates the source code to the bytecode form and stores it in a separate class file.
  2. **java** - The Java interpreter, which interprets the bytecode stored in the class file and executes the program to generate output.
  3. **javadoc** - For creating the HTML document for documentation from source code file.
  4. **javah** - It produces header files for the use of native methods.
  5. **jdb** - The Java debugger which helps to find the errors in the program.

6. **appletviewer** - For executing the Java applet.
7. **javap** - Java disassembler, which helps in converting byte code files into program description.

- Following are the steps that illustrate **execution process of the application program** -

1. The user creates the Java source code in the text editor.
2. The source code is compiled using the **javac** command.
3. The **javadoc** tool can be used to create the HTML files that document the source program.
4. On compiling the source program a class file gets generated which consists of the byte code.
5. The developer may use **javah** tool for generating the required header files.
6. The class file produced by **javac** can be interpreted using **java** in order to produce an executable.

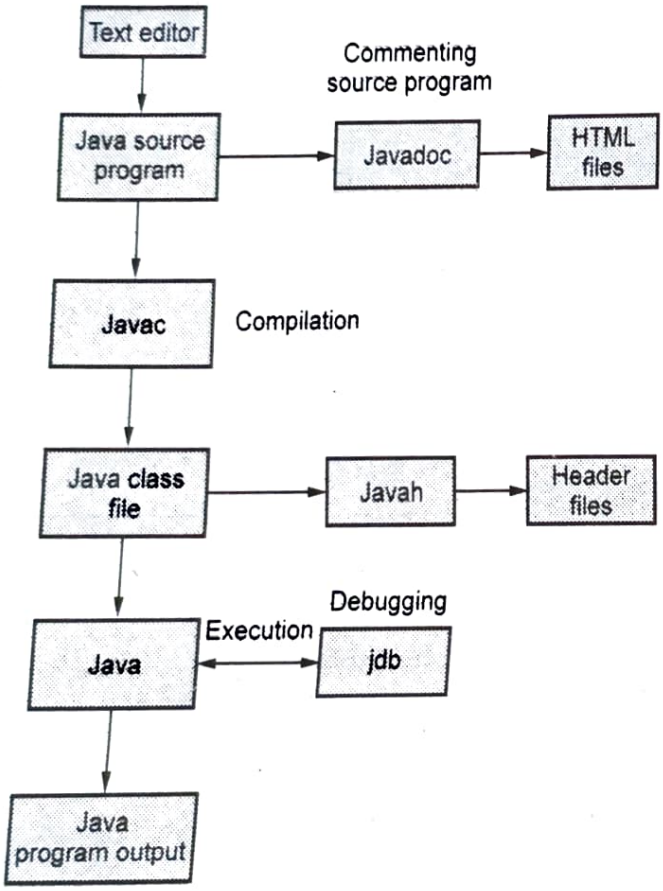


Fig. 1.6.1 Execution process of application program

1.6.2 Application Programming Interface

The Java API consists of large number of classes and methods. These classes and methods are grouped into package. Examples of some commonly used java packages are -

- **Input output package :** This package known as **java.io**. The collection of classes in this package helps in input and output operations.
- **Language support package :** This package is known as **java.lang**. The collection of classes and methods required for implementing basic features of Java.
- **Networking package :** This package is known as **java.net**. The classes and methods required for communicating with other computers through internet is supported by this package.
- **Applet package :** This package includes set of classes and methods that are required for creating applets.

1.6.3 Java Runtime Environment

The Java Runtime Environment(JRE) supports for execution of Java programs. It consists of following components -

- **Java Virtual Machine(JVM) :** The JVM takes the byte code as an input, reads it, interprets it and then executes it. The JVM can generate output corresponding to the underlying platform(operating system). Due to JVM the Java programs become protable.
- **Run Time Class libraries :** These libraries consist of core class libraries that are required for the execution of Java program.
- **Graphical User Interface toolkit :** In Java there are two types of GUI toolkits - AWT and swing.
- **Deployment technologies** : The **Java-plugins** are available that help in the execution of Java applet on the web browser.

1.7 JVM Architecture

Java Virtual Machine is a set of software and program components. As the name suggests, it is a virtual computer that resides in the real computer. Java can achieve its platform independence feature due to Java Virtual Machine.

Before understanding the concept of Java Virtual Machine, let us understand the creation and execution of Java program. (Refer Fig. 1.7.1)

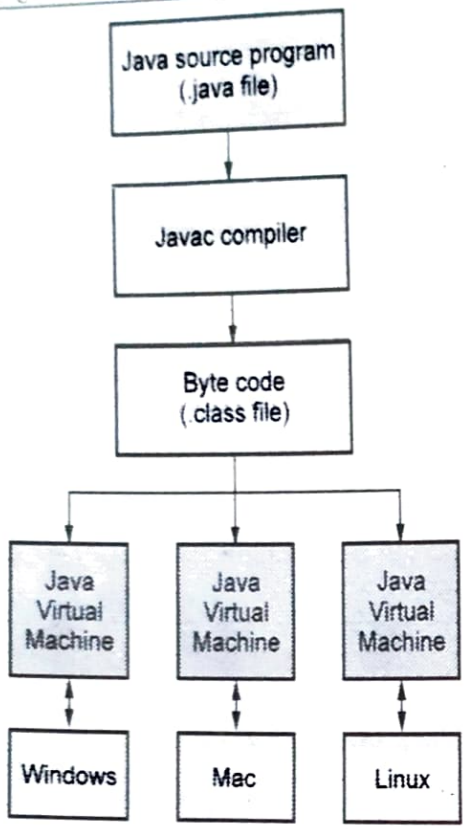


Fig. 1.7.1 Role of JVM

- When we want to write any Java program, we write the source code and store it in a file with extension .java.
- This .java file is compiled using **javac** and a class file gets created. This class file is actually a **byte code**. The name **byte code** is given because of the instruction set of Java program.
- The Java Virtual machine takes **byte code** as an input, reads it, interprets it and then executes it.
- The Java virtual machine can generate an output corresponding to the underlying platform(operating system)

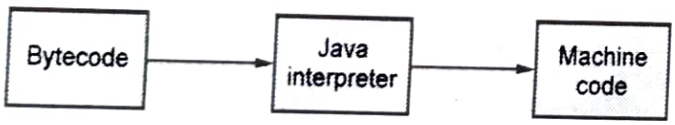


Fig. 1.7.2 Conversion of bytecode to machine code

The Java API works as an intermediary between the user programs and virtual machine.

The virtual machine acts as an intermediary between the Java API and operating system. Thus user can interact with the underlying platform

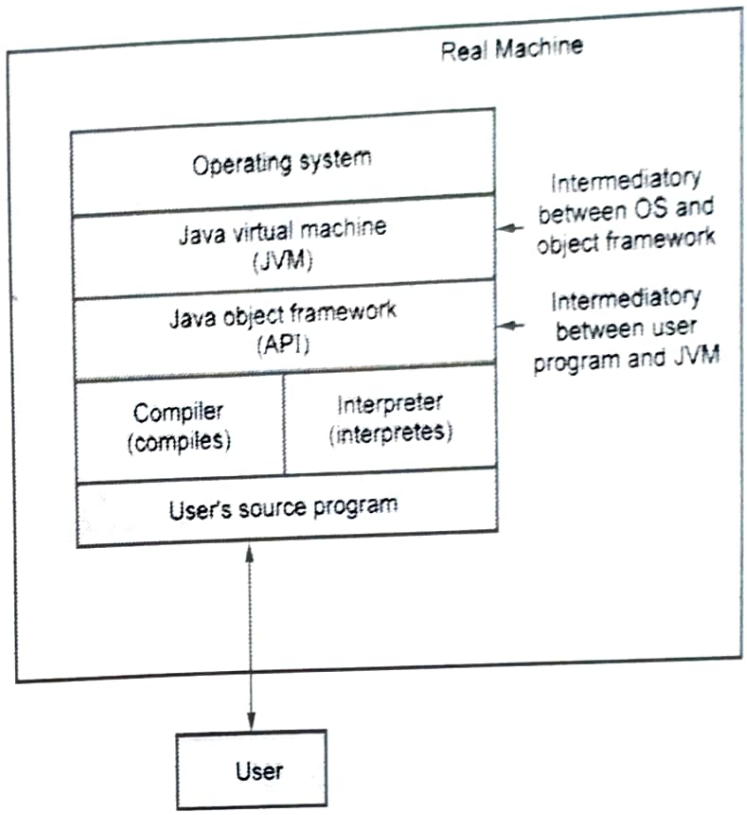
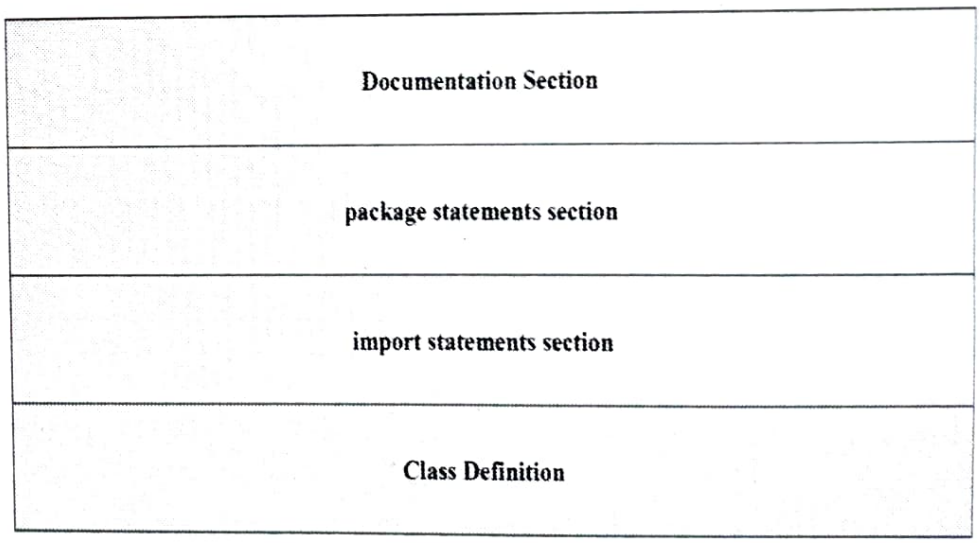


Fig. 1.7.3 Interactions for java program

1.8 Overview of Java Language

1.8.1 Java Program Structure

The program structure for the Java program is as given in the following figure -



```
Main Method class
{
        public static void main(String[] args)
        {
        //main method definition
        }
}
```

Documentation section : The documentation section provides the information about the source program. This section contains the information which is not compiled by the Java. Everything written in this section is written as comment.

Package section : It consists of the name of the package by using the keyword **package**. When we use the classes from this package in out program then it is necessary to write the package statement in the beginning of the program.

Import statement section : All the required java API can be imported by the import statement. There are some core packages present in the java. These packages include the classes and method required for java programming. These packages can be imported in the program in order to use the classes and methods of the program.

Class definition section : The class definition section contains the definition of the class. This class normally contains the data and the methods manipulating the data.

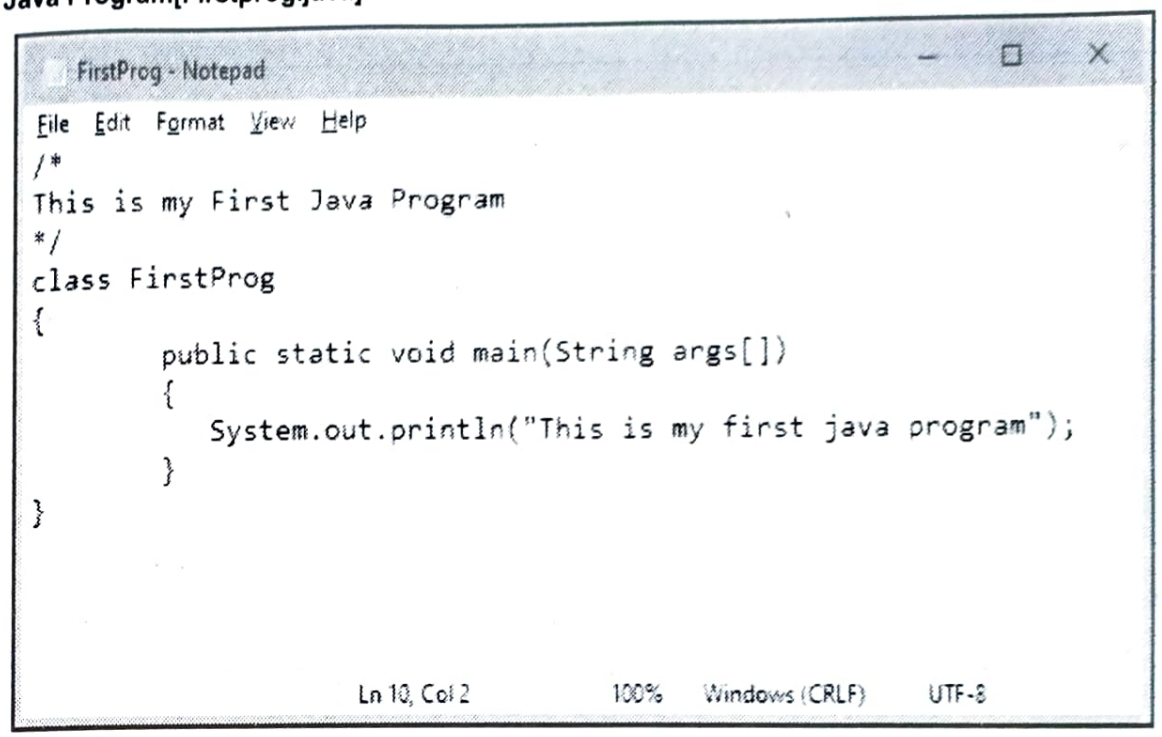
Main method class : This is called the main method class because it contains the main() function. This class can access the methods defined in other classes.

1.8.2 Simple Java Program

Any Java program can be written using simple **text editors** like Notepad or Wordpad. The file name should be same as the name of the class used in the corresponding Java program. Note that the name of the program is *Firstprog* and class name is *Firstprog*.

The extension to the file name should be **.java**. Therefore we have saved our first program by the name *Firstprog.java*

Java Program[Firstprog.java]

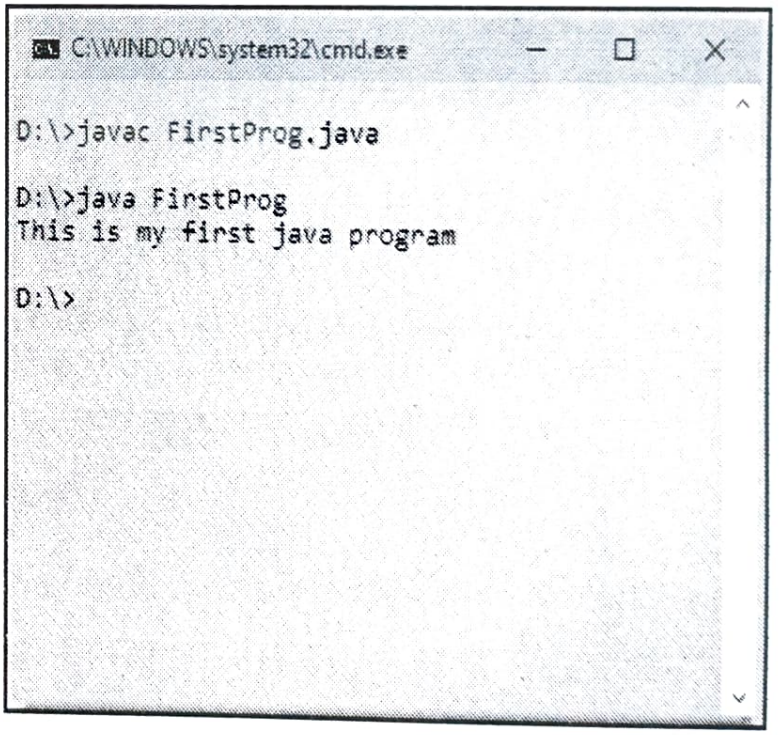


The output of this program can be generated on the command prompt using the commands

javac filename.java

java filename

The sample output is as shown below for the program Firstprog.java



Program explanation

In our First Java program, on the first line we have written a comment statement as

```
/*
This is my First Java Program
*/
```

This is a multi-line comment. Even a single line comment like C++ is also allowed in Java. Hence if the statement would be

```
//This is my First Java Program
```

is perfectly allowed in Java. Then actual program starts with

```
class Firstprog
```

Here class is a keyword and Firstprog is a variable name of class. Note that the name of the class and name of your Java program **should be the same**. The class definition should be within the curly brackets. Then comes

```
public static void main(String args[])
```

This line is for a function **void main()**. The main is of type public static *void* .

The public is a access mode of the *main()* by which the class visibility can be defined. Typically main must be declared as public.

The parameter which is passed to main is String args[]. Here String is a class name and *args[]* is an array which receives the command line arguments when the program runs.

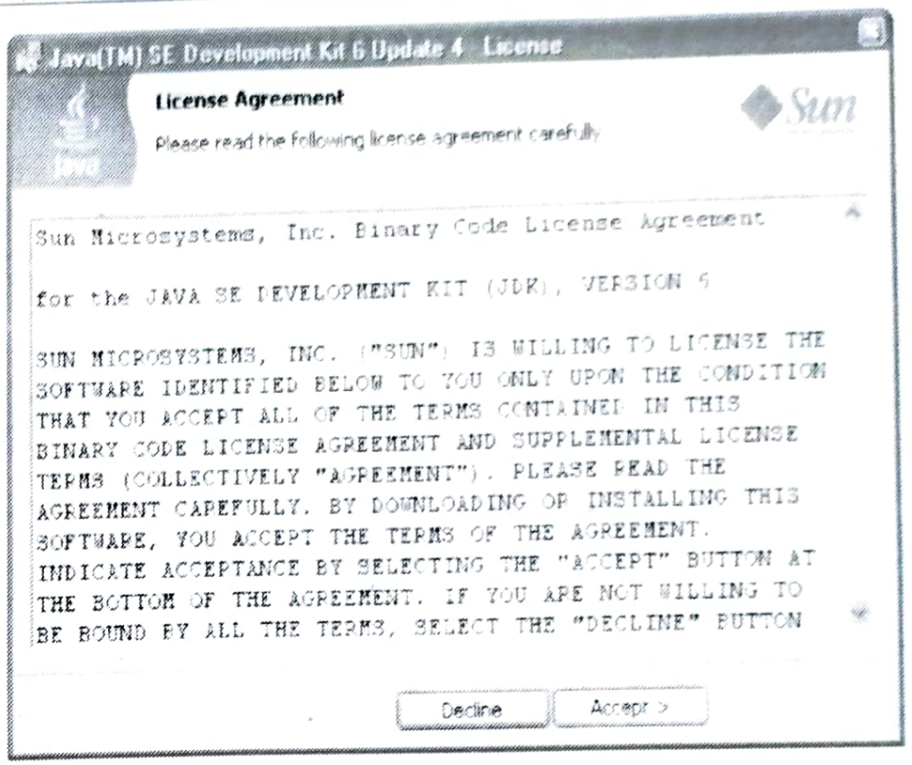
```
System.out.println("This is my first java program");
```

To print any message on the console *println* is a method in which the string "This is my first java program" is written. After println method, the newline is invoked. That means next output if any is on the new line. This *println* is invoked along with *System.out*. The *System* is a predefined class and *out* is an output stream which is related to console. The output as shown in above figure itself is a self explanatory. Also remember one fact while writing the Java programs that it is a case sensitive programming language like C or C++.

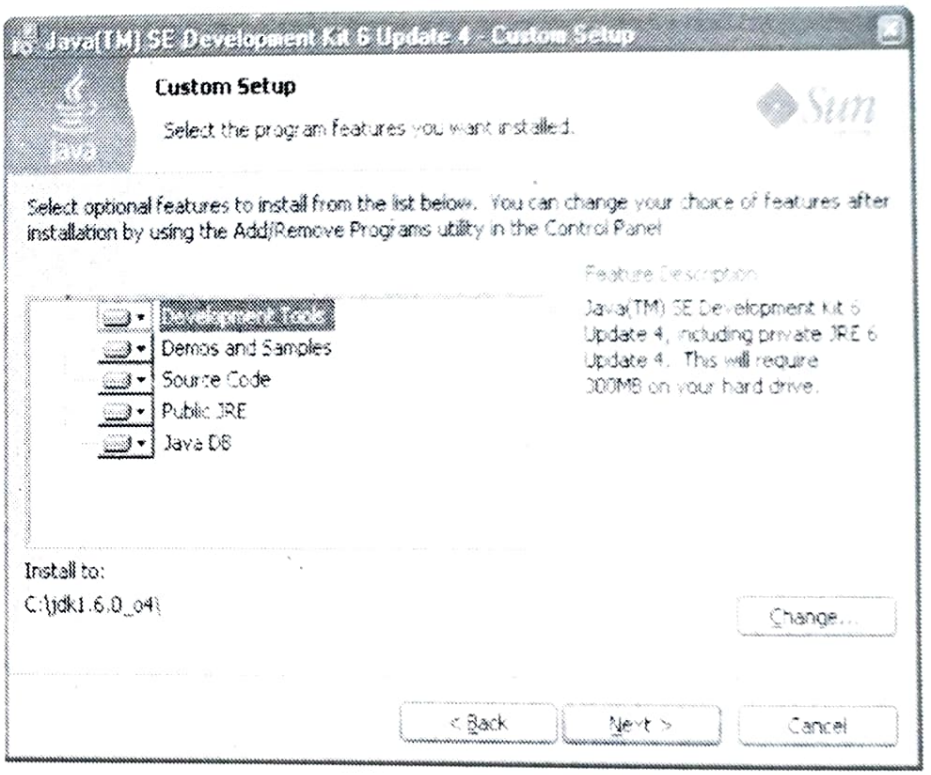
1.9 Installing and Configuring Java

For running any Java program what we require is JDK i.e. Java Development Kit. It is provided by Sun Inc. And it can be downloaded from the website https://www.java.com/en/download/ and you can download the JDK on your machine. After downloading J2SE file on your machine you can install jdk using following steps -

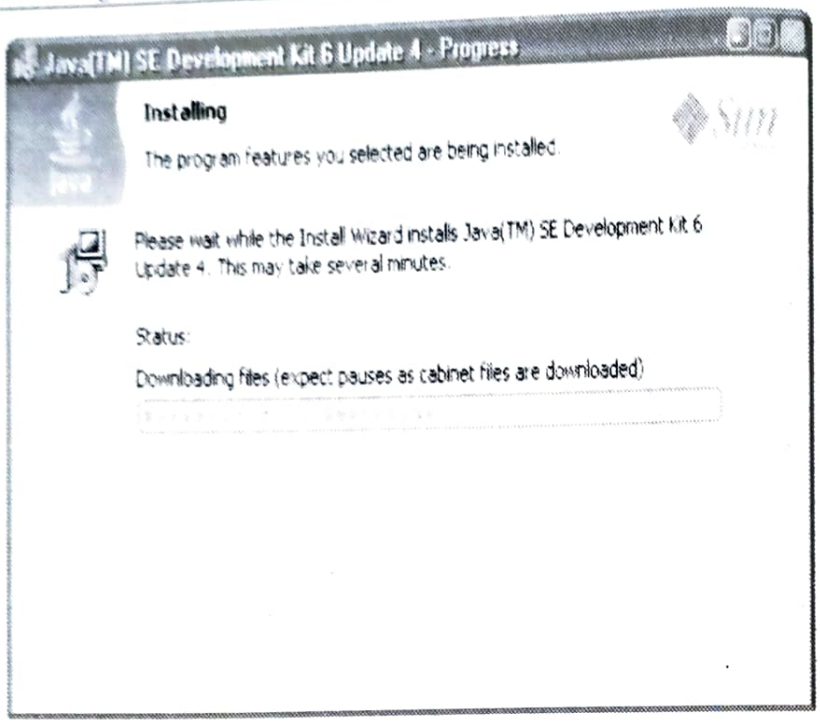
Step 1 : Double click on file download option. And you will get the license agreement window as shown below. Click on "I accept ..." or "Accept" button and then click next.



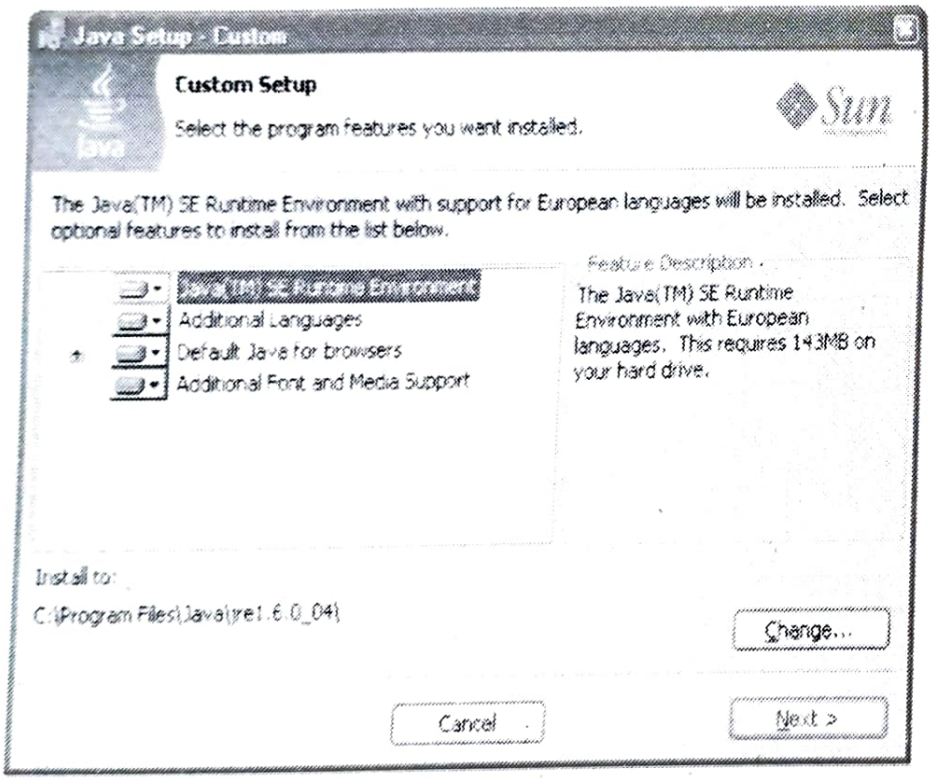
Step 2 : Then the setup screen will appear as follows -



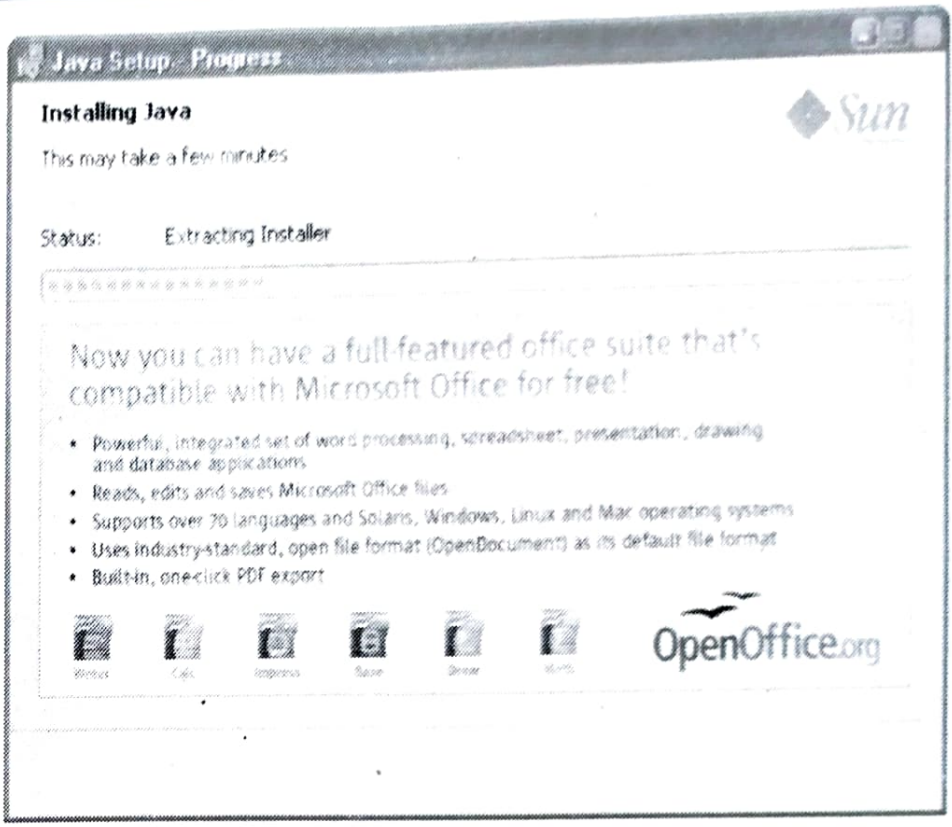
here we can change the default downloading directory to **C:\jdk1.6.0_04**. (I have used this folder) Note that this is an important directory in which the JDK can be downloaded further. Then click next button to proceed. Hence the following screen will appear.



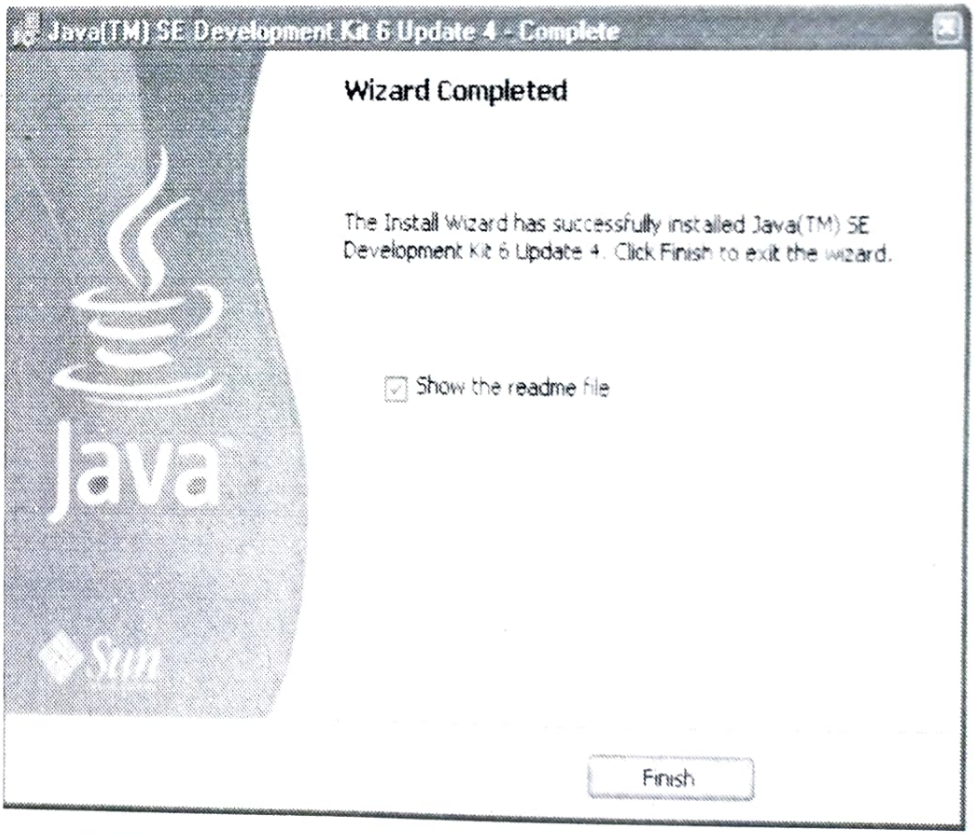
Step 3 : After installing the JDK further the Java Runtime Environment (JRE) will get downloaded. Hence in the next step of installation you will get following screen.



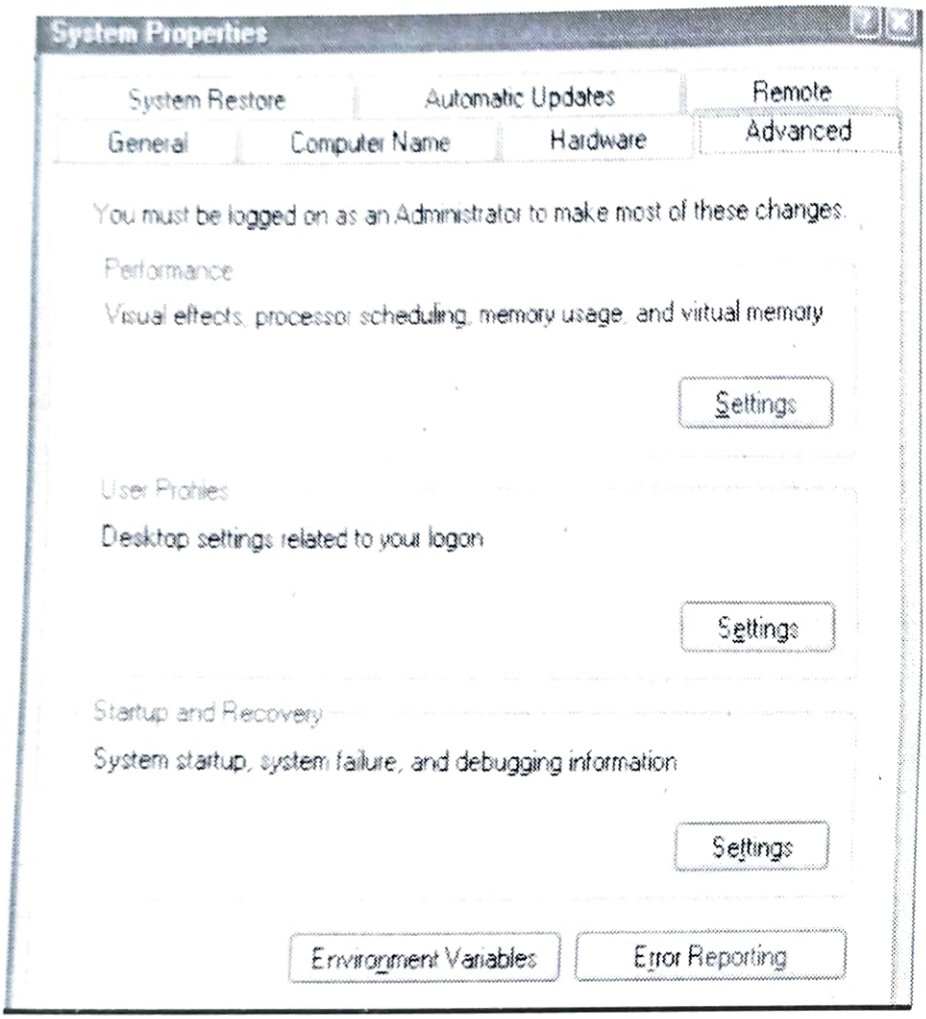
Just click **Next** to get the runtime environment downloaded on your computer. The Java Runtime Environment will help us to run the java program from anywhere from our local hard drive. Then you will get



Step 4 : When the installation will be completed following screen will appear showing the installation success.



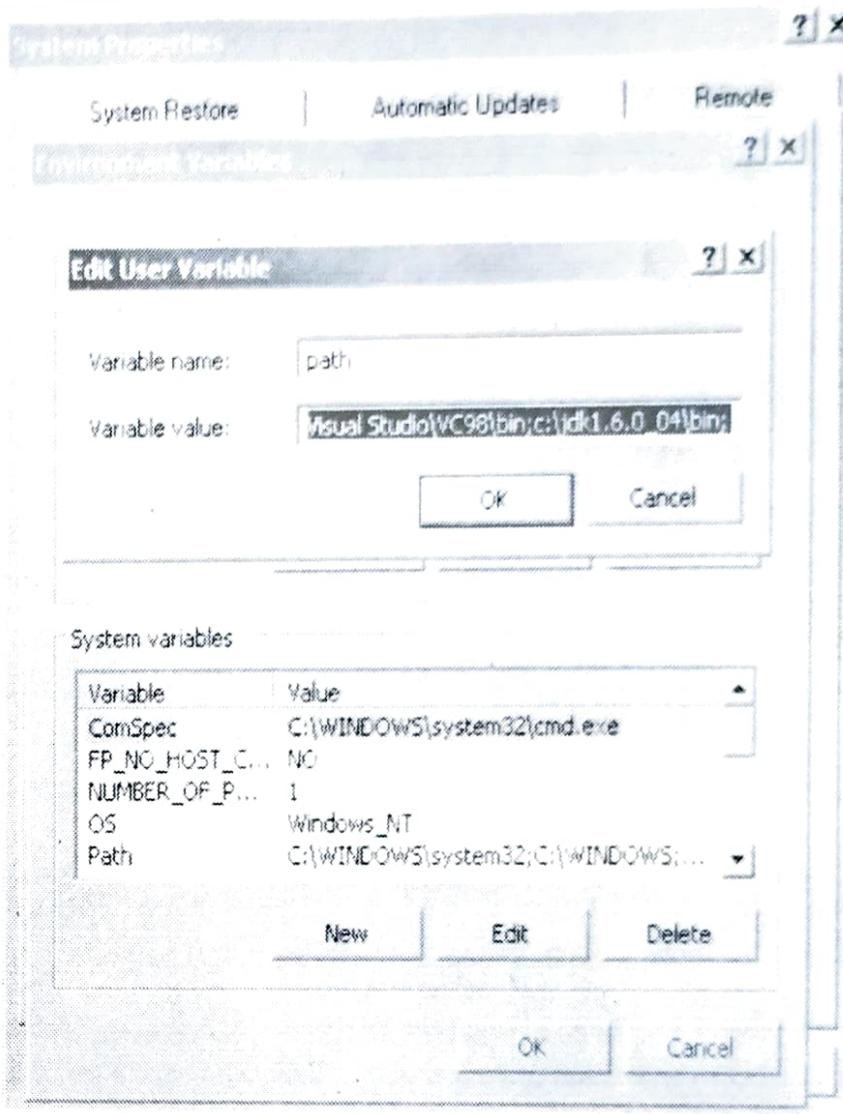
Step 5 : Thus now we have Java installed on our PC. The last important task which we have to do is : Setting up of **environmental variables**. This will help us to run our java programs. For setting the environmental variables just go to the **control panel**. You find the **System** folder over there. Click over there to set the **system properties**. Following screen may appear by this



Click on **Environment Variables** tab in order to set the environment variables. We can set the **path** variable by mentioning the directory C:\jdk1.06_04\bin. Note that in the subdirectory **bin** of our **jdk directory** the executable file of java is downloaded. Hence the **bin** directory has to be mentioned in the path. Click **Ok** to save these settings.

Step 6 : Finally go to the command-prompt and type **javac** over there.

Thus we have got the fully functioning java software ready to run our java programs. Let us now write the first java program and execute it.



1.10 Java Tokens

The **smallest individual and logical unit** of the java statements are called tokens. In Java there are five types of tokens used. These tokens are -

1. Reserved keywords
2. Identifiers
3. Literals
4. Operators
5. Separators

Reserved keywords

Keywords are the reserved words which are enlisted as below -

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

Identifiers

Identifiers are the kind of tokens defined by the programmer. They are used for naming the classes, methods, objects, variables, interfaces and packages in the program.

Following are the rules to be followed for the identifiers -

1. The identifiers can be written using alphabets, digits, underscore and dollar sign.
2. They should not contain any other special character within them.
3. There should not be a space within them.
4. The identifier must not start with a digit, it should always start with alphabet.
5. The identifiers are case - sensitive. For example -

```
int a;
int A;
```

   In above code **a** and **A** are treated as two different identifiers.

6. The identifiers can be of any length.

Literals

Literals are the kind of variables that store the sequence of characters for representing the constant values. Five types of literals are -

1. Integer literal
2. Floating point literal

3. Boolean literal
4. Character literal
5. String literal

As the name suggests the corresponding data type constants can be stored within these literals.

Operators

Operators are the symbols used in the expression for evaluating them.

Separators

For dividing the group of code and arranging them systematically certain symbols are used, which are known as separators. Following table describes various separators.

Name of the Separator	Description
( )	Parenthesis are used to - enclose the arguments passed to it to define precedence in the expression. For surrounding cast type
{ }	Curly brackets are used for - initialising array for defining the block
[ ]	Square brackets are used for - declaring array types dereferencing array values
;	Used to terminate the statement
,	Commas are used to separate the contents.
.	Period is used to separate the package name from subpackages.

1.11 Java Statements

Statements are the executable sentences in Java that are terminated by semicolon. Various types of statements used in Java are -

- **Empty statement**

These statements do nothing but used simply as a placeholder.

- **Labelled statement**

  Labelled statements are the kind of statements which begin with the labels. The labels must not be the reserved keywords or the identifiers that are already defined in the program. These statements are reachable by using jump statements.

- **Expression statement**

  Majority statements in Java are of expression type. Assignment statements, pre and post increment as well as pre and post decrement statements and memory allocation statements are few examples of expression statements.

- **Selection statement**

  The selection statements are the control structure statements which makes use of the programming constructs such as **if, if-else** and **switch**.

- **Iteration statement**

  For specifying the looping these type of statements are used. The programming constructs such as while, do while, and for are used for iteration statement.

- **Jump statement**

  Jump statements are used to transfer the control to any desired location in the program.

- **Synchronization statement**

  For handling the multi - threading in Java these statements are used.

- **Guarding statement**

  The guarding statements are used in order to handle errors, or to avoid the program from crashing. The **try - catch** block statements are called the guarding statements.

1.12 Constants

- Constants mean the fixed values that do not change during the execution of a program.
- In Java various types of constants are used and these are as shown in following Fig. 1.12.1.

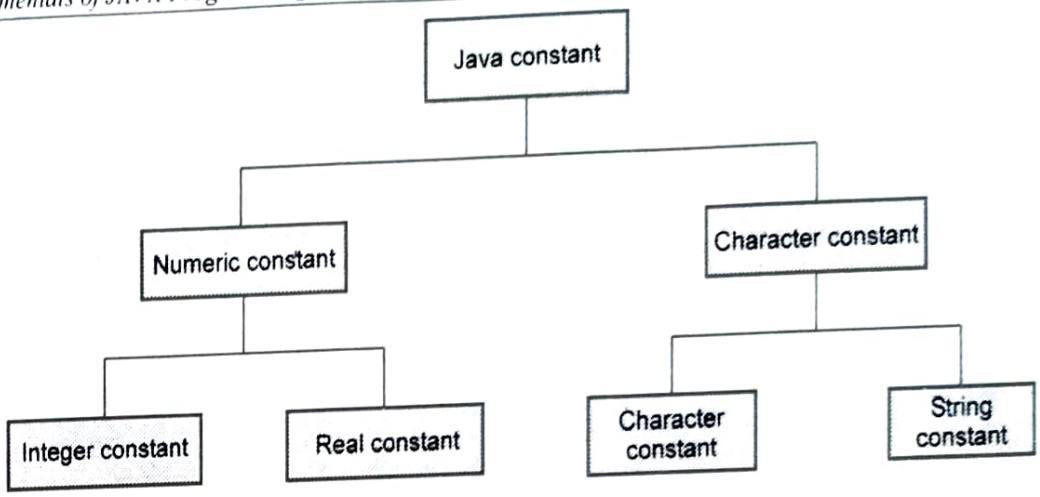


Fig. 1.12.1 Types of Constants

Numeric constants

Numeric constants are used for representing the numerical values. The numeric values can be integer or real type. Hence there exists the integer and real constants.

Integer constants

Integers mean sequence of digits. There are three types of integers -

Decimal integer : The decimal integers consist of a set of digits 0 through 9. These numbers can be unary numbers. That means these can be preceded by unary minus. For example

111, – 200, 455

Octal integer : The octal integer constants consists of set of 0 to 7 having a zero at the beginning. For example -

033, 0431, 0633

Hexadecimal integer : Along with the 0 to 9 digits, it also consists of the letters A to F.

Real constants : The real constants contain the floating point parts. For example -

0.045, 0.567, 0.3211

Character constants : A single character constant is enclosed within a pair of single quotes. For example-

'a', 'T', '1'

String constant : A string is a collection of characters enclosed between double quotes. The characters can be alphabets, numbers, white spaces or special characters.

For example-

"India" ,"Best10", "s?ss"

1.13 Variables and its Declarations

- **Definition**: A variable is an identifier that denotes the storage location.
- Variable is a fundamental unit of storage in Java. The variables are used in combination with identifiers, data types, operators and some value for initialization. The variables must be declared before its use.
- The **syntax** of variable declaration will be -

```
data_type name_of_variable [=initialization][,=initialization][,...];
```

- Following are **some rules** for variable declaration -
  - The variable name should not with digits.
  - No special character is allowed in identifier except underscore.
  - There should not be any blank space with the identifier name.
  - The identifier name should not be a keyword.
  - The identifier name should be meaningful.
  - For Example :

```
int a,b;
char m='a';
byte k=12,p,t=22;
```

1.14 Data Types

Various data types used in Java are byte, short, int, long, char, float, double and boolean.

byte

This is in fact smallest integer type of data type. Its width is of 8-bits with the range –128 to 127. The variable can be declared as byte type as -

```
byte i,j;
```

short

This data type is also used for defining the signed numerical variables with a width of 16 - bits and having a range from –32,768 to 32,767. The variable can be declared as short as

```
short a,b;
```

int

This is the most commonly used data type for defining the numerical data. The width of this data type is 32-bit having a range – 2,147,483,648 to 2,147,483,647. The declaration can be

```
int p,q;
```

long

Sometimes when int is not sufficient for declaring some data then *long* is used. The range of long is really very long and it is –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. The declaration can be

```
long x,y;
```

float

To represent the real number(i.e. number that may have decimal point) float data type can be used. The width is 32-bit and range of this data type is 3.4e – 038 to 3.4e+038.

double

To represent the real numbers of large range the double data type is used. Its width is 64 - bit having the range 1.7e–308 to 1.7e+308.

char

This data type is used to represent the character type of data. The width of this data type is 16-bit and its range is 0 to 65,536.

boolean

Boolean is a simple data type which denotes a value to be either true or false.

Java Program

Note that for displaying the value of variable, in the **println** statement the variable is appended to the message using + [In C we use comma in printf statement but in Java + is used for this purpose]

Java Program [DatatypeDemo.java]

```
/*
This program introduces use of various data type
in the program
*/
class DatatypeDemo
{
 public static void main(String args[])
 {
  byte a=10;
  short b=10*128;
  int c=10000* 128;
  long d=10000*1000*128;
  double e=99.9998;
```

```
    char f='a';
    boolean g=true,
    boolean h=false;        } dynamic initialization
    System.out.println("The value of a=: "+a);
    System.out.println("The value of b=: "+b);
    System.out.println("The value of c=: "+c);
    System.out.println("The value of d=: "+d);
    System.out.println("The value of e=: "+e);
    System.out.println("The value of f=: "+f);
    f++;
    System.out.println("The value of f after increment=: "+f);
    System.out.println("The value of g=: "+g);
    System.out.println("The value of h=: "+h);
  }
}
```

Output

```
The value of a=: 10
The value of b=: 1280
The value of c=: 1280000
The value of d=: 1280000000
The value of e=: 99.9998
The value of f=: a
The value of f after increment=: b
The value of g=: true
The value of h=: false
```

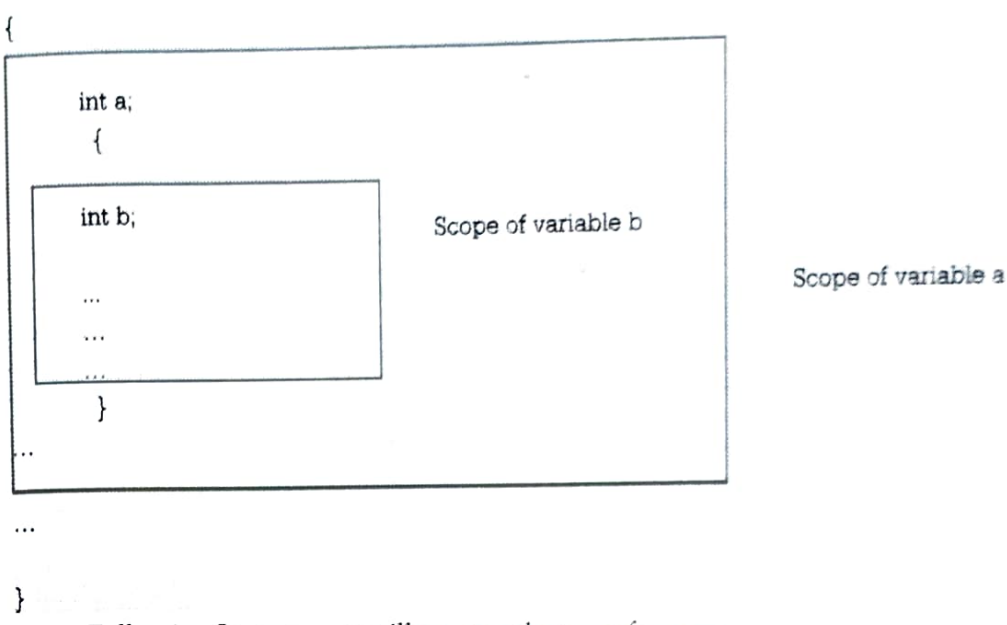
Dynamic initialization

Java is a flexible programming language which allows the dynamic initialization of variables. In other words, at the time of declaration one can initialize the variables(*note that in the above program we have done dynamic initialization*). In Java we can declare the variable at any place before it is used.

1.15 Scope of Variables

- Scope allows the visibility of object within the specific region.
- The scope is defined by a block which begins at the opening curly bracket and ends by a closing curly bracket.
- The scopes can be nested.
- The outer scope encloses the inner scope. In the nested scope the inner and outer objects are visible in the inner scope.
- Java allows to declare variables within any block. Thus any block defines the scope of that block.

For instance



- Following Java program illustrates the use of scope -

```
public class VarScopeProg
{
public static void main(String[] args)
 {
int A=10;              // scope of vairable A = 10 starts here
  {
int B=20;              // scope of B = 20 starts here
System.out.println("Block A: "+A);
System.out.println("Block B: "+B);
  }              //scope of B=20 ends here
  {
int B=30;      //scope of B=30 starts here
System.out.println("Block A: "+A);
System.out.println("Block B: "+B);
  }              //scope of B=30 ends here
int B=40;//scope of B=40 starts here
  {
System.out.println("Block A: "+A);
System.out.println("Block B: "+B);

  }
 }       //scope of all the variables end here
}
```

Output

```
D:\>javac VarScopeProg.java
D:\>java VarScopeProg
```

```
Block A: 10
Block B: 20
Block A: 10
Block B: 30
Block A: 10
Block B: 40
```

The comments indicate the scope of each corresponding variable. At the end of the program scope of all the variable ends.

1.16 Symbolic Constants

There are two problems that are associated with the constants and those are -

1. **Modifiability** : The constants may need to be modified during the program. For example : The value of pi may be 3.14 at one place in the program and it may be required as 3.14159 at some other place in the same program for the accuracy.
2. **Understandability** : The purpose of each constant need to be remembered. For example : The constant INDEX may be required to store the index, the constant COUNT may keep track of some count or there may be some constant for SIZE.
3. To overcome these problems there is a use of **symbolic constants**. The keyword **final** is used to declare the symbolic constants. The syntax for declaring the symbolic constants is -

```
final datatype variable_name=value
```

For example-

```
final int INDEX=1;
final int SIZE=10;
```

Following are the conventions that are associated with the symbolic names -

1. Symbolic names must be written in capital letters.
2. After declaring the symbolic name one must not modify it.
3. The symbolic constants are declared for types.
4. The symbolic constants can not be declared inside a method. They should be used only as class data members in the beginning of the class.

1.17 Arrays

- Array is a collection of similar type of elements. Thus grouping of similar kind of elements is possible using arrays.
- Typically arrays are written along with the size of them.
- The **syntax** of declaring array is -

```
data_typearray_name[ ];
```

and to allocate the memory -

```
array_name=new data_type[size];
```

where *array_name* represent name of the array, **new** is a keyword used to allocate the memory for arrays, *data_type* specifies the data type of array elements and *size* represent the size of an array. For example :

```
a=new int[10];
```

[Note that in C/C++ this declaration is as good as int a[10]]

- After this declaration the array **a** will be created as follows

Array a[10]

0	0	0	0	0	0	0	0	0	0

- That means after the above mentioned declaration of array, all the elements of array will get initialized to zero. Note that, always, while declaring the array in Java, we make use of the keyword **new**, and thus we actually make use of dynamic memory allocation.
- Therefore arrays are **allocated dynamically** in Java. Let us discuss on simple program based on arrays.

Java Program [SampleArray.Java]

```
/*
This is a Java program which makes use of arrays
*/
classSampleArray
{
public static void main(String args[])
{
int a[];
  a=new int[10];
System.out.println("\tStoring the numbers in array");
a[0]=1;
a[1]=2;
a[2]=3;
a[3]=4;
a[4]=5;
a[5]=6;
a[6]=7;
a[7]=8;
a[8]=9;
a[9]=10;
System.out.println("The element at a[5] is: "+a[5]);
 }
}
```

Output

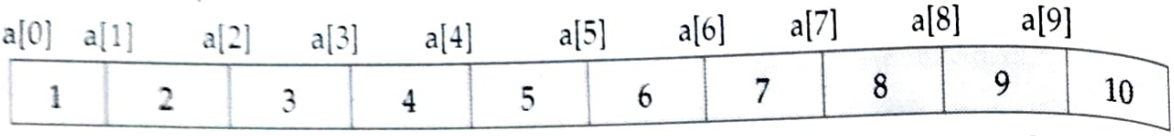
```
Storing the numbers in array
The element at a[5] is : 6
```

Program explanation

In above program

(1) We have created an array of 10 elements and stored some numbers in that array using the array index.

(2) The array that we have created in above program virtually should look like this -



(3) The **System.out.println** statement is for printing the value present at a[5]. We can modify the above program a little bit and i.e. instead of writing

```
int a[];
  a=new int[10];
```

these two separate statements we can combine them together and write it as -

```
int a[]=new int[10];
```

That means the declaration and initialization is done simultaneously.

Another way of initialization of array is

```
int a[ ]= {1,2,3,4,5,6,7,8,9,10};
```

That means, as many number of elements that are present in the curly brackets, that will be the size of an array. In other words there are total 10 elements in the array and hence size of array will be 10.

1.18 Typecasting

It is common practice to assign a value of one data type element to another. Java allows the automatic conversion between these data types. But Java performs type conversion between the compatible data type elements. For instance : If we assign an integer value to a float type element then Java automatically makes this conversion. However, this automatic conversion can not be done between any data type, rather such a conversion is done with the compatible data type elements.

Conditions in which Java allows automatic conversion
When two data types are compatible to each other.
When the target data type element is larger then the source data type

For example:

```
int a=5;
long b;
b=a;
```

For incompatible data types we have to do explicit type conversion. This mechanism is called **casting**. Hence

Defintition :

Type casting means explicit conversion between incompatible data types.

For casting the values we use following syntax :

(*type*) variable

Example :

```
double a;
int b;
b=(int) a;
```

In the following Java program, we use the type casting

Java Program[TypeCastProg.java]

```
public class TypeCastProg
{
 public static void main(String[] args)
 {
   double x;
   int y;
   x=25.50;
   System.out.println("Value of [double] x: "+x);
   System.out.println("Conversion of double to integer");
   y=(int)x;
   System.out.println("Value of [integer] y: "+y);
   int m;
   double n;
   m=10;
   n=(double)m;
   System.out.println("Value of [integer] m: "+m);
   System.out.println("Conversion of integer to double");
   System.out.println("Value of [double] n: "+n);

 }
}
```

Output

```
Value of [double] x: 25.5
Conversion of double to integer
Value of [integer] y: 25
Value of [integer] m: 10
Conversion of integer to double
Value of [double] n: 10.0
```

Program Explanation

In above program, we have made the type casting of -

Integer to double

double to integer

When a double value is converted to an integer the fractional parts is lost. This is called **truncation**. But when an integer gets converted to double then fractional part is **padded**

Following table shows the possible type casting without loss of information -

From	To
byte	short, char, int, long, float, double
short	int, long, float, double
char	int, long, float, double
int	long, float, double
long	double , float
float	double

1.19 Standard Default Values

Java provides default values to the variables if we don't initialise them. The standard default values for the variables of different data types are enlisted below -

Data type of the variable	Default value
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d

char	null character
Boolean	false
reference	null

1.20 Operators and Expressions

Various types operators used in Java are -

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Increment and decrement operator

1.20.1 Arithmetic Operators

Type	Operator	Meaning	Example
Arithmetic	+	Addition or unary plus	c=a+b
	–	Subtraction or unary minus	d= –a
	*	Multiplication	c=a*b
	/	Division	c=a/b
	%	Mod	c=a%b

Program demonstrating arithmetic operators

```
class ArithOperDemo
{
 public static void main(String[] args)
 {
  System.out.println("\n Performing arithmetic operations ");
  int a=10,b=20,c;
  System.out.println("a= "+a);
  System.out.println("b= "+b);
  c=a+b;
  System.out.println("\n Addition of two numbers is "+c);
  c=a-b;
  System.out.println("\n Subtraction of two numbers is "+c);
  c=a*b;
  System.out.println("\n Multiplication of two numbers is "+c);
  c=a/b;
  System.out.println("\n Division of two numbers is "+c);
 }
}
```

Output

```
Performing arithmetic operations
a= 10
b= 20

Addition of two numbers is 30

Subtraction of two numbers is -10

Multiplication of two numbers is 200

Division of two numbers is 0
```

1.20.2 Relational Operators

Type	Operator	Meaning	Example
Relational	<	Less than	a<4
	>	Greater than	b>10
	<=	Less than equal to	b<=10
	>=	Greater than equal to	a>=5
	==	Equal to	x==100
	!=	Not equal to	m!=8

Program Demonstrating Relational Operators

```
class RelOperDemo
{
public static void main(String[] args)
{
 System.out.println("\n Performing Relational operations ");
 int a=10,b=20,c=10;
 System.out.println("a= "+a);
 System.out.println("b= "+b);
 System.out.println("\n The a<b is "+(a<b));
 System.out.println("\n The a>b is "+(a>b));
 System.out.println("\n The a==c is "+(a==c));
 System.out.println("\n The a<=b is "+(a<=b));
 System.out.println("\n The a>=c is "+(a>=c));
 System.out.println("\n The a!=b is "+(a!=b));

}
}
```

Output

```
Performing Relational operations
a= 10
b= 20

The a<b is true

The a>b is false

The a==c is true

The a<=b is true

The a>=c is true

The a!=b is true
```

1.20.3 Logical Operators

Type	Operator	Meaning	Example
Logical	&&	And operator	0&&1
	||	Or operator	0 || 1

Program Demonstrating Logical Operators

```
import java.io.*;
import java.lang.*;
public class LogicalOper
{
 public static void main(String args[])throws IOException
 {
        boolean oper1,oper2;
        oper1=true;
        oper2=false;
        boolean ans1,ans2;
        ans1=oper1&oper2;
        ans2=oper1|oper2;
        System.out.println("The oper1 is: "+oper1);
        System.out.println("The oper2 is: "+oper2);
        System.out.println("The oper1&oper2 is: "+ans1);
        System.out.println("The oper1|oper2 is: "+ans2);
 }

}
```

Output

```
The oper1 is: true
The oper2 is: false
The oper1&oper2 is: false
The oper1|oper2 is: true
```

1.20.4 Increment and Decrement

Type	Operator	Meaning	Example
Increment	++	Increment by one	++i or i++
Decrement	--	Decrement by one	--k or k--

Program Demonstrating Increment and Decrement Operator

```
class IncrDecrOperDemo
{
public static void main(String[] args)
{
 System.out.println("\n Performing increment and decrement operations ");
 int a=11,b=22;
 System.out.println("a= "+a);
 System.out.println("b= "+b);
 System.out.println("\n PreIncrementing a "+ ++a);
 System.out.println("\n PostIncrementing b "+ b++);
 System.out.println("\n After this statement the value of b is "+ b);
}
}
```

Output

```
Performing increment and decrement operations
a= 11
b= 22

PreIncrementing a 12

PostIncrementing b 22

After this statement the value of b is 23
```

1.20.5 Conditional Operators

The conditional operator is "?" The syntax of conditional operator is -

```
Condition?expression1:expression2
```

Where expression1 denotes the true condition and expression2 denotes false conditio

For example:

a>b?true:false

This means that if a is greater than b then the expression will return the value true otherwise it will return false.

Program Demonstrating Conditional Operator

```
class Test
{
    public static void main(String args[])
    {
        int a,b,c;
        a=10;
        b=20;
        System.out.println("a = "+a);
        System.out.println("b = "+b);
        c=a>b?a:b;
        System.out.println(c+" is greatest");
    }
}
```

Output

```
a = 10
b = 20
20 is greatest
```

1.20.6 Bitwise Operators

Assume A=35 i.e. in binary form 00100011 and B = 14 i.e. in binary form 00001110

Type	Operator	Meaning	Example
bitwise and	&	Binary AND Operator copies a bit to the result if it exists in both operands.	A&B=00100011 & 00001110 = 0000010

Truth Table

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

<table>
<tr><td>bitwise or
|</td><td>Binary OR Operator copies a bit if it exists in either operand.
Truth Table<table><tr><th>A</th><th>B</th><th>Output</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table></td><td>A|B
=
00100011 | 0001110
= 00101111 i.e. 47</td></tr>
<tr><td>bitwise XOR
^</td><td>Binary XOR Operator copies the bit if it is set in one operand but not both.
Truth Table<table><tr><th>A</th><th>B</th><th>Output</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table></td><td>A^B
=
00100011 ^ 0001110
= 00101101
i.e. 45</td></tr>
<tr><td>bitwise complement
~</td><td>Binary ones complement operator is unary and it has effect of flipping bits.</td><td>~A=11011100 = ~36</td></tr>
<tr><td>Left shift
<<</td><td>The left operands value is moved left by the number of bits specified by the right operand.</td><td>00100011<<2
=1000 1100</td></tr>
<tr><td>Right shift
>></td><td>The left operands value is moved right by the number of bits specified by the right operand.</td><td>00100011>>2
= 0000 1000</td></tr>
<tr><td>Zero fill right shift
>>></td><td>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.</td><td>00100011>>>2
== 0000 1000</td></tr>
</table>

Program Demonstrating Bitwise Operator

```
public class Test {

  public static void main(String args[]) {
    int a = 35;        /* 35 = 00100011 */
```

```
    int b = 14;          /* 14 = 00001110 */
    int c = 0;

    c = a & b;       /* 2 = 0000 0010 */
    System.out.println("a & b = " + c );

    c = a | b;       /* 47 = 00101111 */
    System.out.println("a | b = " + c );

    c = a ^ b;       /* 45 = 00101101 */
    System.out.println("a ^ b = " + c );

    c = ~a;          /*-36 */
    System.out.println("~a = " + c );

    c = a << 2;      /* 140 = 1000 1100 */
    System.out.println("a << 2 = " + c );

    c = a >> 2;      /* 8 = 0000 1000 */
    System.out.println("a >> 2  = " + c );

    c = a >>> 2;     /* 8 = 0000 1000 */
    System.out.println("a >>> 2 = " + c );
  }
}
```

1.20.7 Instance of Operators and Dot Operators

- There are two special operators used in Java-**instanceof** and **dot** operators.
- For determining whether the object belongs to particular class or not an **instanceof** operator is used. For example -

  Ganga instanceof River

  If the object Ganga is an object of class River then it returns true otherwise false.
- The **dot** operator is used to access the instance variable and methods of class objects.

For example -

Customer.name //for accessing the name of the customer

Customer.ID //for accessing the ID of the customer

1.20.8 Operator Precedence and Associativity

- **Precedence :** The precedence relation specifies which operation must be done first during the expression evaluation.

- **Associativity** : Associativity tells the direction of execution of operators that can be either left to right or right to left. For example, in expression a = b = c = 10 the assignment operator is executed from right to left that means c will be assigned by 10, then b will be assigned by c and finally a will be assigned by b.
- Following table denotes the precedence relation and associativity. The operators are arranged from Highest priority to lowest priority. The operators in the same row indicate the equal precedence.

Precedence	Operator	Description	Associativity
1	[]	Array index	Left -> Right
	()	Method call	
	.	Member access	
2	++	Pre or postfix increment	Right -> Left
	--	Pre or postfix decrement	
	+ -	Unary plus, minus	
	~	Bitwise NOT	
	!	Logical NOT	
3	(type cast)	type cast	Right -> Left
	new	object creation	
4	*	multiplication	Left -> Right
	/	division	
	%	modulus (remainder)	
5	+ -	addition, subtraction	Left -> Right
	+	String concatenation	
6	<<	Left shift	Left -> Right
	>>	Signed right shift	
	>>>	Unsigned or zero-fill Right shift	
7	<	Less than	Left -> Right
	<=	Less than or equal to	
	>	Greater than	
	>=	Greater than or equal to	
	instanceof	reference test	

8	== !=	Equal to Not equal to	Left -> Right
9	&	Bitwise AND	Left -> Right
10	^	Bitwise XOR	Left -> Right
11	|	Bitwise OR	Left -> Right
12	&&	Logical AND	Left -> Right
13	||	Logical OR	Left -> Right
14	? :	Conditional (ternary)	Right -> Left
15	= + = - = * = / = % = & = ^ = | = << = >> = >>> =	Assignment and short hand assignment operators	Right -> Left

1.20.9 Evaluation of Expression

- Expression is a combination of operators and operands in specific manner.
- For example

  c=a+b*c; // Expression with arithmetic operators

  (a>b)&&(b<c)// Expression with logical operators and relational operators
- The evaluation of expression is an order of evaluating the expression using precedence and associativity rules.
- For example -

  To evaluate 2+3*4 we will first perform 3*4=12, then add 2 to 12 and the result of this expression will be 14. This is because multiplication has higher precedence than addition. So we perform multiplication operation first and then addition.

1.20.10 Mathematical Functions

- Java provides a support for performing mathematical operations by means of mathematical functions.
- The basic mathematical functions are defined in the **Math** class defined in **java.lang** package. Various mathematical functions are -

sin(double angle)	Returns the sine of angle in radians
cos(double angle)	Returns the cosine of angle in radians
tan (double angle)	Returns the tan of angle in radians
asin(double val)	Returns the angle whose sine is val
acos(double val)	Returns the angle whose cosine is val
atan(double val)	Returns the angle whose tan is val
pow(double a, double y)	It computes a^b
exp(double a)	It computes e^a
sqrt(double a)	It computes square root of a
max(a,b)	It computes the maximum of a and b
min(a,b)	It computes the minimum of a and b
log(val)	It computes the logarithmic values of val
abs(val)	It computes the absolute value of val
ceil(val)	It returns the smallest whole number which is greater than or equal to val
floor(val)	It returns the largest whole number which is lesser than or equal to val.

Following is an example of Java program which makes use of mathematical function -

Example 1.20.1 *Write a simple Java program to illustrate use of mathematical function.*

Solution :

Java Program[MathDemo.java]

```
class MathDemo
{
 public static void main(String[] args)
 {
  double val;
```

```
val=25;
System.out.println("\n The square root of "+val+" is "+Math.sqrt(val));
int a=10,b=20;
System.out.println("\n The maximum of "+a+" and "+b+" is "+Math.max(a,b));
System.out.println("\n The minimum of "+a+" and "+b+" is "+Math.min(a,b));
System.out.println("\n The sin of "+val+" is "+Math.sin(val));
System.out.println("\n The cos of "+val+" is "+Math.cos(val));
}
}
```

Output

The square root of 25.0 is 5.0

The maximum of 10 and 20 is 20

The minimum of 10 and 20 is 10

The sin of 25.0 is -0.13235175009777303

The cos of 25.0 is 0.9912028118634736

Review Questions

1. *Describe any two relational and any two logical operators in Java with simple example.*
2. *Explain any two relational operators in Java with example.*
3. *'? :' What this operator is called ? Explain with suitable example.*
4. *Explain any two bit-wise operators with example.*
5. *Explain following bitwise operator with an example :*
   (1) Left shift operator (2) Right shift operator

1.21 Control Statements- Decision Making and Looping

1.21.1 If Statement

The if statement is of two types -

i) **Simple if statement :** The if statement in which only one statement is followed by that is statement.

Syntax

```
if(apply some condition)
  statement
```

For example

```
if(a>b)
        System.out.println("a is Biiiiig!");
```

ii) Compound if statement : If there are more than one statement that can be executed when if condition is true. Then it is called compound if statement. All these executable statements are placed in curly brackets.

Syntax

```
if(apply some condition)
{
 statement 1
 statement 2
.
.
.
statement n
}
```

1.21.2 if else Statement

The syntax for if...else statement will be -

```
if(condition)
 statement
else
 statement
```

For example

```
if(a>b)
 System.out.println( "a is big")
else
 System.out.println ("b :big brother")
```

The if...else statement can be of compound type even. For example,

```
if(raining==true)
{
 System.out.println(" I won't go out");
 System.out.println(" I will watch T.V. Serial");
 System.out.println(" Also will enjoy coffee");
}
else

{
 System.out.println("I will go out");
 System.out.println("And will meet my friend");
 System.out.println("we will go for a movie");
 System.out.println("Any how I will enjoy my life");
}
```

1.21.3 Nested if-else Statement

The syntax of if...else if statement is

```
if(is condition true?)
statement
else if(another condition)
statement
else if(another condition)
statement
else
statement
```

For example

```
if(age==1)
 System.out.println("You are an infant");
else if(age==10)
 System.out.println("You are a kid");
else if(age==20)
 System.out.println("You are grown up now");
else
System.out.println("You are an adult");
```

1.21.4 If-else-if Ladder

- The control flow statements are evaluated from top to bottom. Hence the if-else-if ladder will be evaluated from top to bottom.
- As soon as an if statement from the ladder evaluates to true, the statements associated with that if are executed and the remaining part of the ladder is bypassed.
- The last most else is executed only when no condition in the whole ladder returns true.
- **Example Program**

```
public class Test
{
  public static void main(String[] args)
  {
    char marks=69;

    if (marks<50)
      System.out.println("You are Fail");
    else if (marks >=50 && marks<60)
      System.out.println("Second class is ");
    else if (marks>60 && marks<70)
      System.out.println(First Class");
    else
      System.out.println(" Distinction!!!.");
  }
}
```

Output

```
First Class
```

1.21.5 Switch Statement

Using switch statement we can make a choice from a list of values. Each value in switch statement is considered as **case**. A variable is used in switch statement to match the desired case.

The Syntax of switch statement is

```
switch(expression)
{
        case value:
                statements
                break;
        case value:
                statements
                break;
        ...
        default:  //optional
                //statements
}
```

For example

```
switch(ch)
{
        case 1: System.out.println("Monday");
                break
        case 2: System.out.println("Tuesday");
                break;
        case 3: System.out.println("Wednesday");
                break
        case 4: System.out.println("Thursday");
                break;
        case 5: System.out.println("Friday");
                break
        case 6: System.out.println("Saturday");
                break;
        case 7: System.out.println("Sunday");
                break;
}
```

Example 1.21.1 *In what ways does a switch statement differ from an if statements ?*

Solution :

Sr. No.	if statement	switch statement
1	if-else statement test for equality as well as for logical expression.	switch statement tests only for equality.
2	The if statement evaluates integer, character pointer, floating point or boolean values.	The switch statement evaluates only character or integer value.
3	The if-else statement uses multiple statements for multiple choices.	switch statement uses single expression for multiple choices.
4	Either if statement will be executed or else statement will be executed.	Switch statements executes one case after other until it reads break statement.

1.21.6 Nested Switch Statement

Nested Switch statement is a switch statement inside another switch statement.

For example - Consider following Java program

```
public class Test
{
    public static void main(String[] args)
    {
        int a,b;
        a=1;b=2;
        switch(a)
        {
            case 1:
                switch(b)
                {
                    case 1:System.out.println("Java");
                        break;
                    case 2:System.out.println("PHP");
                        break;
                }
                break;
            case 2:
                switch(b)
                {
                    case 1:System.out.println("HTML");
                        break;
                    case 2:System.out.println("XML");
                        break;
                }
                break;
        }
```

```
    }
}
```

Output

```
PHP
```

Program Explanation : In above Program,

As value of a=1, the control will enter in the outer switch with case 1.

As value of b=2,the control will enter in the inner switch with case 2.

Hence the output is "PHP".

1.21.7 The while Statement

- This is one form of statement which is used to have iteration of the statement for the any number of times.
- The syntax is,

```
while(condition)
{
 statement 1;
 statement 2;
 statement 3;
 ...
 statement n;
}
```

For example

```
int count=1;
 while(count<=5)
 {
    System.out.println("I am on line number "+count);
   count++;
}
```

Let us see a simple Java program which makes the use of while construct.

Java Program [whiledemo.java]

```
/*
This is java program which illustrates
while statement
*/
class whiledemo
{
public static void main(String args[])
{
 int count=1,i=0;
 while(count<=5)
 {
```

```
    i=i+1;
    System.out.println("The value of i= "+i);
    count++;
   }
  }
}
```

Output

```
The value of i= 1
The value of i= 2
The value of i= 3
The value of i= 4
The value of i= 5
```

Example 1.21.2 *Define a class having one 3 digit number, num as data member, initialize and display reverse of that number.*

Solution :

```
public class NumReverse
{
    int num;
    public void initialize()
    {
        num=123;
        System.out.println("The number is: "+num);
    }
    public void display()
    {
        int rev=0;
        System.out.print("Reverse of number is: ");
        while(num > 0)
        {
            rev = rev * 10 + (num % 10);
            num=num/10;
        }
        System.out.println(rev);
    }
    public static void main(String[] args)
    {
        NumReverse obj=new NumReverse();
        obj.initialize();
        obj.display();
    }
}
```

Output

```
The number is: 123
Reverse of number is: 321
```

Example 1.21.3 *Write a program to find sum of digit of number entered by user.*

Solution :

```
public class DigitSum
{
        int num;
        public void initialize()
        {
                num=123;
                System.out.println("The number is: "+num);
        }
        public void display()
        {
                int sum=0;
                while(num > 0)
                {
                        sum = sum + (num % 10);
                        num = num / 10;
                }
                System.out.println("The sum of digits is: "+sum);
        }
        public static void main(String[] args)
        {
                DigitSum obj=new DigitSum();
                obj.initialize();
                obj.display();
        }
}
```

Output

```
The number is: 123
The sum of digits is: 6
```

Example 1.21.4 *Write a program to find the number of and sum of all integers greater than 100 and less than 200 that are divisible by 7.*

Solution :

```
public class DigitRange
{
        public static void main(String[] args)
```

```
    {
        int sum=0;
        System.out.println("\n Numbers between 100 to 200 which is divisible by 7");
        for(int i=101;i<200;i++)
        {
            if(i%7==0)
            {
                System.out.print("  "+i);
                sum= sum + i;
            }
        }
        System.out.println("\n sum = "+sum);
    }
}
```

Output

```
Numbers between 100 to 200 which is divisible by 7
  105   112   119   126   133   140   147   154   161   168   175
182  189  196
 sum = 2107
```

1.21.8 The do while Statement

- This is similar to while statement but the only difference between the two is that in case of do...while statement the statements inside the do...while must be executed at least once.
- This means that -
  - The statement inside the do...while body gets executed first and then the while condition is checked for next execution of the statement.
  - Whereas in the while statement first of all the condition given in the while is checked first and then the statements inside the while body get executed when the condition is true.

Syntax

```
do
{
 statement 1;
 statement 2;
  ...
 statement n;
}while(condition);
```

For example

Java Program [dowhiledemo.java]

```
/*
This is java program which illustrates
do...while statement
*/
class dowhiledemo
{
public static void main(String args[])
{
 int count=1,i=0;
 do
 {
 i=i+1;
 System.out.println("The value of i= "+i);
 count++;
 }while(count<=5);
}
}
```

Output

```
The value of i= 1
The value of i= 2
The value of i= 3
The value of i= 4
The value of i= 5
```

1.21.9 The for Statement

for is a keyword used to apply loops in the program. Like other control statements for loop can be categorized in simple for loop and compound for loop.

Simple for loop :

```
for (statement 1;statement 2;statement 3)
 execute this statement;
```

Compound for loop :

```
for(statement 1;statement 2; statement 3)
{
execute this statement;
execute this statement;
execute this statement;
that's all;
}
```

Here

Statement 1 is always for initialization of conditional variables,

Statement 2 is always for terminating condition of the for loop,

Statement 3 is for representing the stepping for the next condition.

For example :

```
for(int i=1;i<=5;i++)
{
 System.out.println("Java is an interesting language");
System.out.println("Java is a wonderful language");
System.out.println("And simplicity is its beauty");
}
```

Let us see a simple Java program which makes use of for loop.

Java Program [forloop.java]

```
/*
This program shows the use of for loop
*/
class forloop
{
public static void main(String args[])
{
 for(int i=0;i<=5;i++)
  System.out.println("The value of i: "+i);
}
}
```

Output

```
The value of i: 0
The value of i: 1
The value of i: 2
The value of i: 3
The value of i: 4
The value of i: 5
```

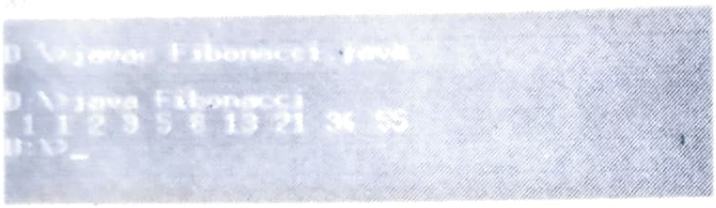
Example 1.21.5 *Write a program to generate Fibonacci series 1 1 2 3 5 8 13 21 34 55 89.*

Solution :

```
class Fibonacci
{
      public static void main(String args[])
      {
            int a,b, c, n;
```

```
        a=b=1;
        for(n=1;n<=10;n++)
        {
            System.out.print(" "+a);
            c=a+b;
            a=b;
            b=c;
        }
    }
}
```

Output



1.21.10 The Break Statement

When a **break** statement is encountered in a loop then the loop is terminated and the control resumes the next statement following the loop. For example

Java Program

```
public class breakDemo
{
  public static void main(String[] args)
  {
    for (int i = 1; i <= 10; i++)
    {
      if (i == 5)
      {
        break;    // terminate loop if i is 5
      }
      System.out.print(i + " ");
    }
    System.out.println("Loop is terminated using break!!");
  }
}
```

Output

```
1 2 3 4 Loop is terminated using break!!
```

Program Explanation : In above program,

1) The for loop is executed for i=1 to i=10.

2) But when i value reaches to 5, the break statement is encountered.
3) Due to this, the control terminates the for loop and the **System.out.println** statement outside the for loop will be executed. Hence is the output.

1.21.11 The continue Statement

When a **continue** statement is encountered inside the body of a loop, remaining statements are skipped and loop proceeds with the next iteration. For example -

Java Program

```
public class continueDemo
{
  public static void main(String[] args)
  {
    for (int i = 1; i <= 10; i++)
    {
      if (i == 5)
      {
        continue;
      }
      System.out.print(i + " ");
    }

  }
}
```

Output

```
1 2 3 4 6 7 8 9 10
```

Program Explanation : In above program,

1) When the value of i = 5 then it is skipped and loop proceeds with further remaining values of i. Hence is the output.
2) Note that : You cannot use break to transfer control to a block of code that does not enclose the break statement.

Example 1.21.6 *Write a program to check whether an entered number is prime or not.*

Solution :

```
class PrimeNumberDemo
{
        public static void main(String args[]) {
                int n = Integer.parseInt(args[0]);
                boolean flag = true;
                if(n < 2)
```

```
            flag = false;
        else
        {
            for(int i=2; i<=n/2; i++)
            {
                if(n % i == 0)
                {
                    flag = false;
                    break;
                }
                flag = true;
            }
        }
        if(flag == true)
            System.out.println(n + " is a prime number");
        else
            System.out.println(n + " is not a prime number");
    }
}
```

Output

```
D:\MSBTE_JAVA _Programs> javac PrimeNumberDemo.java
D:\MSBTE_JAVA _Programs> java PrimeNumberDemo 15
15 is not a prime number
```

1.21.12 The return Statement

- The **return** statement is used to return from a function. That means using **return** statement the control goes back to the caller function.
- At any time the return statement can be used to transfer back the control to the caller function.
- The return statement causes to terminate the function immediately.
- **Example Program**

```
class ReturnProg
{
    public static void main(String args[])
    {
        int a=10;
        System.out.println("This statement is before execution of return");
        if(a==10)
            return;
        System.out.println("This statement will not be excuted!!!");
    }
}
```

Output

```
This statement is before execution of return
```

1.21.13 The Nested Loops

When one loop is present inside the other loop then that structure is called nested loop.

The outer loop takes control of the number of complete repetitions of the inner loop.

Example Program

```
class NestedLoops
{
    public static void main(String args[])
    {
        int i,j;
        for(i=1;i<=3;i++)//outer loop
        {
            System.out.println(" i = "+i);
            for(j=1;j<=3;j++)//inner loop
            {
                System.out.print(" j = "+j);
            }
            System.out.println("\n----------");
        }
    }
}
```

Output

```
i = 1
j = 1 j = 2 j = 3
----------
i = 2
j = 1 j = 2 j = 3
----------
i = 3
j = 1 j = 2 j = 3
----------
```

Example 1.21.7 *Write a program to print the following output :*

1 1 1 1 1

2 2 2 2

3 3 3

4 4

5

Solution :

```
class Test
{
        public static void main(String args[])
        {
                int i,j;
                for(i=1;i<=5;i++)
                {
                        for(j=5;j>=i;j--)
                        {
                                System.out.print("  "+i);
                        }
                        System.out.println("");
                }
        }
}
```

Output

```
1  1  1  1  1
2  2  2  2
3  3  3
4  4
5
```

Example 1.21.8 *Write a java program to display all the odd numbers between 1 to 30 using for loop and if statement.*

Solution :

```
public class Test
{
        public static void main(String[] args)
        {
                System.out.println("The odd numbers are: ");
                for(int i=1;i<=30;i++)
                {
                        if(i%2!=0)
                                System.out.print("  "+i);
                }
        }
}
```

Output

```
The odd numbers are:
  1  3  5  7  9  11  13  15  17  19  21  23  25  27  29
```

Example 1.21.9 *Write a Java program to display following pattern.*

```
    *
   ***
  *****
 *******
*********
```

Solution :

```
class TriangleDisplay
{
    public static void main(String[] args)
    {
        int i,j,k;
        for(i=1; i<=5; i++)
        {
            for(j=4; j>=i; j--)
            {
                System.out.print(" ");
            }
            for(k=1; k<=(2*i-1); k++)
            {
                System.out.print("*");
            }
            System.out.println("");
        }
    }
}
```

Output

```
    *
   ***
  *****
 *******
*********
```

Example 1.21.10 *Write a Java program to display following number pattern*

```
1
1 2
```

```
1 2 3
1 2 3 4
1 2 3 4 5
```

Solution :

```
class NumberPatternDisplay
{
    public static void main(String[] args)
    {
        int i,j;
        for(i=1; i<=5; i++)
        {
            for(j=1; j<=i; j++)
            {
                System.out.print(j+" ");
            }
            System.out.println("");
        }
    }
}
```

Output

```
D:\>javac NumberPatternDisplay.java

D:\>java NumberPatternDisplay
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5

D:\>
```

1.21.14 The Labeled Loops

1) labelled break

- Sometimes the control needs to be transferred to a specific instruction. This can be done using the statements like **goto**.
- Java provides the support for jump statement using **labeled break**.
- The syntax for labelled break is

  break *label*;

***label* is the** name of the block which need to be skipped.

- Following program demonstrates the use of labeled break.

Example Program

```
class breakLblDemo
{
    public static void main(String args[])
    {
        int i=5;
        first:
        {
            second:
            {
                third:
                {
                    System.out.println("Before the break.");
                    if(i==5)
                        break second; // break out of second block
                        System.out.println("Inside third block.");
                }
                System.out.println("Inside second block.");
            }
            System.out.println("Inside first block.");
        }
    }
}
```

Output

```
Before the break.
Inside first block.
```

Program Explanation : In above program,

1) We have created three blocks, by labeling them as first, second and third.
2) Inside the third block the **labeled break** statement is written to break the second block.
3) Hence the control skips execution of second and third block and reaches to the first block. So is the output.

2) labelled Continue

- The labelled continue is basically a continue statement with label.
- The syntax is

  continue *label;*

- **Example Program :** Following java program, illustrates the labeled continue statement.

```
class continueLblDemo
{
    public static void main(String args[])
    {
        Label1: for(int i=0;i<10;i++)
            {
                if(i%2==0)
                {
                    System.out.println("Even value: = "+i);
                    continue Label1;
                }
                System.out.println("Odd value: = "+i);
            }
    }
}
```

Output

```
Even value: = 0
Odd value: = 1
Even value: = 2
Odd value: = 3
Even value: = 4
Odd value: = 5
Even value: = 6
Odd value: = 7
Even value: = 8
Odd value: = 9
```

1.21.15 The for-Each Version of the for Loop

- The **for-each** element is used to access each element of an array.
- There are two advantages of for-each statement -
  i) It eliminates the possibility of bug.
  ii) It makes the code more readable and understandable.
- **Syntax**

```
for(data_type variable_name:arry_name)
{
}
```

- **Example Program**

```
class ForEachProg
{
    public static void main(String args[])
    {
        int array[]={10,20,30,40};
        System.out.println("The elements in array are ...");
```

```
        for(int i:array)
            System.out.println(i);
    }
}
```

Output

```
The elements in array are ...
10
20
30
40
```

Review Questions

1. *Write general syntax of any two decision making statements and also give its examples.*
2. *Explain break and continue statements with example.*
3. *State syntax and describe working of 'for each' version of for loop with one example.*
4. *Illustrate with example the use of switch case statement.*
5. *Describe break and continue statement with example.*

1.22 Multiple Choice Questions

Q.1 Java is associated with ___ programming language

[a] Basic [b] C

[c] Oak [d] Perl

Q.2 Java programming was designed by _____.

[a] Amazon [b] Microsoft

[c] Intel [d] Sun Microsystem

Q.3 Who is said to be the father of Java programming language ?

[a] Dennis Ritchie [b] Bajame Stroustrup

[c] James Gosling [d] None of these

Q.4 Earlier name of Java was _____.

[a] Oak [b] J

[c] JVM [d] J++

Q.5 Java was publicly released in ____

[a] 1991 [b] 1992

[c] 1994 [d] 1995

Q.6 The first public implementation of Java was ___

[a] Java 1.0 [b] Java 1.1

[c] Java Premium 1.1 [d] None of these

Q.7 Write once and run anywhere at anytime forever is a feature of ___ programming

[a] C [b] C++

[c] Java [d] Smalltalk

Q.8 Following is a object oriented programming language _____.

[a] Java [b] C++

[c] Python [d] All of these

Q.9 Java is made useful for distributed systems. It is possible due to _____ feature of JAVA

[a] Inheritance [b] RMI

[c] API [d] None of these

Q.10 Java is said to be _____.

[a] Compiled [b] Interpreted

[c] Both compiled and interpreted [d] None of these

Q.11 ___ is the one of the features of object oriented programming that allows creation o hierarchical classification

[a] Dynamic Binding [b] Polymorphism

[c] Inheritance [d] None of these

Q.12 Following is one notable feature of Java :

[a] Robust and secure [b] Platform independence

[c] Useful in distributed system [d] All of these

Q.13 Java code is said to be ___ so that it can easily run on any system

[a] Portable [b] Discrete

[c] Simple [d] None of these

Java Environment

Q.14 Java Virtual machine takes ___ as input and produces output

[a] source code [b] byte code

[c] intermediate code [d] machine code

Q.15 For compiling Java code ___ tool is used

[a] javac [b] java

[c] Javadoc [d] jar

Q.16 The Java code is executed using ____ tool

[a] javac [b] java

[c] Javadoc [d] jar

Q.17 Compiler converts the Java program into an intermediate language representation which is called ___

[a] Byte [b] Bit

[c] Bytecode [d] Byteclass

Q.18 ___ was the first web browser developed in Java

[a] Internet Explorer [b] Opera

[c] Netscape Navigator [d] HotJava

Q.19 Java is designed for the distributed environment of internet because it handles ___ protocol

[a] TCP/IP [b] UDP

[c] FTP [d] None of these

Q.20 Java platform consists of JVM and a package of readymade software components is called ___

[a] JVM API [b] API

[c] JAVA API [d] None of these

Q.21 JVM stands for _____.

[a] Java Variable Machine [b] Java Verifying Machine

[c] Java Visual Machine [d] Java Virtual Machine

Q.22 JVM is basically a _____.

[a] compiler [b] Interpreter

[c] Debugger [d] None of these

Q.23 JDK stands for _____.

[a] Java Designing Kit [b] Java Development Kit

[c] Java Distributed Kit [d] None of these

Q.24 The platform independent code file created from the source program is understandable by ____.

[a] JRE [b] JVM

[c] Compiler [d] None of these

Q.25 Java tokens are classified into _____.

[a] keyword, identifiers [b] constants, operators

[c] punctuators [d] All of these

Q.26 Constants are also known as _____.

[a] literals [b] Numeric

[c] Character [d] None of these

Q.27 ___ constant refers to single character enclosed in single quote

[a] Numeric [b] character

[c] String [d] None of these

Q.28 In ___ the character is preceded by blackslash character

[a] variable [b] constant

[c] Escape Sequence [d] None of these

Q.29 Four types of integers in Java are _____.

[a] bit, byte, short and long [b] nibble, byte,int and long

[c] byte, short, int and long [d] None of these

Q.30 Primitive data types are also called as _____.

[a] Built-in Data type [b] User defined data type

[c] Primary Data types [d] None of these

Q.31 Which of the following is not Java keyword ?

[a] this [b] public

[c] synchronize [d] none of these

Q.32 What is the range of byte data type in Java ?

[a] –1024 to 1024 [b] –128 to 127

[c] – 32768 to 32768 [d] None of these

Q.33 Which of the following can be contained in float data type ?

[a] 1.7e-308 [b] 3.4e-038

[c] 1.7e+388 [d] 3.4e-068

Q.34 What is the range of char data type in Java ?

[a] –128 to 127 [b] 0 to 256

[c] 0 to 65535 [d] None of these

Q.35 Choose the correct statement

[a] boolean a='false' [b] boolean a='true'

[c] boolean a="true" [d] boolean a=true

Q.36 What is the output of following code ?

```
class Test
{
    public static void main(String args[])
    {
        char i = 'X';
        i++;
        System.out.print((int)i);
    }
}
```

[a] 89 [b] 88

[c] 87 [d] None of these

Q.37 What is the output of the following code ?

```
class Test
{
    public static void main(String args[])
    {
        boolean a = true;
        boolean b = false;
        if (!a)
            System.out.println(a);
        else
            System.out.println(b);
    }
}
```

[a] true [b] false

[c] 0 [d] 1

Q.38 How many primitive data types are there in Java ?

[a] 4 [b] 7

[c] 8 [d] 10

Q.39 In Java byte, short, int and long all these are _____.

[a] signed [b] unsigned

[c] both of these [d] none of these

Q.40 Size of int in Java is _____.

[a] 16 bit [b] 32 bit

[c] 64 bit [d] None of these

Q.41 The smallest integer type is ___ and its size is ___ bits

[a] short . 16 [b] short 8

[c] byte 8 [d] byte 32

Q.42 Size of float and double is _____.

[a] 32 bit and 64 bit [b] 16 bit and 32 bit

[c] 8 bit and 16 bit [d] none of these

Q.43 The expression containing byte, int and literal number is promoted to _____.

[a] byte [b] char

[c] int [d] long

Q.44 Which of the following can not be used for variable name in Java ?

[a] identifier [b] Keyword

[c] both a and b [d] None of these

Q.45 ___ are the Java statements that are used in order to handle error and avoid program from crashing

[a] Expression Statement [b] Selection Statement

[c] Guard Statement [d] Jump Statement

Q.46 Synchronization Statements are used for ____ in Java

[a] Handling multiple threads

[b] Handling error-causing statements

[c] Handling expressions

[d] None of these

Q.47 ___ and ___ variables are declared inside class

[a] Class and local [b] Class and instance

[c] Global and local [d] none of these

Q.48 The Keyword ___ is used to declare symbolic constant

[a] const [b] static

[c] final [d] double

Q.49 What is the output of the following code ?

```
class Test
{
    public static void main(String args[])
    {
        int a = 10;
        int b = 20;
```

```
    int c;
    int d;
    c = ++b;
    d = a++;
    c++;
    b++;
    ++a;
    System.out.println(a + " " + b + " " + c);
}
```

[a] 11 21 21 [b] 12 21 22

[c] 12 22 22 [d] None of these

Q.50 What will be the output of the following code ?

```
lass Test

    public static void main(String args[])
    {
        int x = 11;
    double y = 11.11;
    boolean b = (x = y);
    System.out.println(b);
}
```

[a] true [b] false

[c] Syntax error [d] exception

Q.51 && and || are called short circuit operator because _____.

[a] the outcome of the expression is determined by left operand

[b] the outcome of the expression is determined by right operand

[c] the outcome is always true

[d] the outcome is always false

Q.52 The necessary condition for automatic type conversion in Java is _____.

[a] The destination type should be smaller than the source type

[b] The destination type should be larger than the source type

[c] There is no automatic conversion allowed in Java

[d] None of these

Q.53 When double value is converted to an integer the fractional part is lost. This is called _____.

[a] truncation [b] loss

[c] padding [d] None

Q.54 ___ function returns the smallest whole number which is greater than or equal to val

[a] max()
[b] floor(val)
[c] ceil(val)
[d] None of these

Q.55 What is acceptable value of x for following code fragment ?

```
switch(x)
{
      default: System.out.println("Hi");
}
```

[a] float or short
[b] byte or char
[c] long or short
[d] None of these

Q.56 The syntax error will be on line _____.

```
public class While
{
  public void loop()
  {
    int x= 0;
    while ( 1 ) /* Line A */
    {
      System.out.print("x plus one is " + (x + 1)); /* Line B */
    }
  }
}
```

[a] Line A and Line B
[b] Line A
[c] Line B
[d] No error

Q.57 What is the output of the following code ?

```
public class Test
{
      public static void main(String args[])
      {
            int i;
            for(i = 1; i < 10; i++)
            {
            if(i > 5)
                        continue;
       }
       System.out.println(i);
   }
}
```

[a] 5
[b] 10
[c] 4
[d] None of these

Q.58 What is the output of the following code ?

```
public class Test
{
        public static void main(String args[])
        {
                int i = 1, j = -1;
                switch (i)
                {
                        case 0, 1: j = 1;
                                break;
                        case 2: j = 2;
                                break;
                        default: j = 0;
                }
                System.out.println("j = " + j);
    }
}
```

[a] j = 0 [b] j = 1

[c] Syntax error [d] j = 2

Q.59 What is the output of the following code ?

```
class Test
{
        public static void main(String args[])
        {
                int i = 0;
                for ( ; i < 10; ++i)
                {
                        if (i % 2 == 0)
                                continue;
                        else if (i == 8)
                                break;
                        else
                                System.out.print(i + " ");
            }
        }
}
```

[a] 1 3 5 7 9 [b] 8

[c] 1 3 5 7 [d] 1 2 3 4 5 6 7 8 9

Q.60 What will be the output of following code ?

```
public class Test
{
    public static void main(String[] args)
        {
```

```
        int x = 10, y = 20;
        switch(x + 1){
            case 11: y = 0;
            case 12: y = 1;
            default: y += 1;
        }
    }
}
```

[a] Syntax error
[b] 2
[c] 20
[d] 0

Answer Keys for Multiple choice Questions :

Q.1	c	Q.2	d	Q.3	c
Q.4	a	Q.5	d	Q.6	a
Q.7	c	Q.8	d	Q.9	b
Q.10	c	Q.11	c	Q.12	d
Q.13	a	Q.14	b	Q.15	a
Q.16	b	Q.17	c	Q.18	d
Q.19	a	Q.20	c	Q.21	d
Q.22	b	Q.23	b	Q.24	c
Q.25	d	Q.26	a	Q.27	b
Q.28	c	Q.29	c	Q.30	a
Q.31	c	Q.32	b	Q.33	b
Q.34	c	Q.35	d	Q.36	a
Q.37	b	Q.38	c	Q.39	a
Q.40	b	Q.41	c	Q.42	a
Q.43	c	Q.44	b	Q.45	c
Q.46	a	Q.47	b	Q.48	c
Q.49	c	Q.50	c	Q.51	a
Q.52	b	Q.53	a	Q.54	c
Q.55	b	Q.56	b	Q.57	b
Q.58	c	Q.59	a	Q.60	b

Explanations of Multiple Choice Questions :

Q.9 **Explanation :** The RMI stands for Remote method Invocation. Two different objects on two remote machine can communicate with each other due to Remote Method Invocation feature.

Q.10 **Explanation :** First, Java compiler translates the Java source program into a special code called bytecode. Then Java interpreter interprets this bytecode to obtain the equivalent machine code. This machine code is then directly executed to obtain the output.

Q.11 **Explanation :** Inheritance is one of the feature of object oriented programming in which the derived class is derived from base class. There can be single level, or multiple levels of inheritance possible.

Q.33 **Explanation :** The range of float data type.

Q.36 **Explanation :** The ASCII value of X is 88, it is incremented by one hence output is 89.

Q.43 **Explanation :** The expression containing byte, int, and literal is promoted to integers before any computations.

Q.44 **Explanation :** The keywords are the reserved words which are not allowed to be the names of the variables.

Q.50 **Explanation :** The syntax error occurs because instead of (x=y) there should be (x==y). The single = is assignment operator and = = is equality operator which returns true or false.

Q.55 **Explanation :** The byte and char is converted to integer.

Q.56 **Explanation :** Using the integer 1 in the while statement, or any other looping or conditional construct for that matter, will result in a compiler error. This is old C Program syntax, not valid Java.

Q.58 **Explanation :** The statement case 0,1 causes error due to comma

Q.59 **Explanation :** Whenever i is divisible by 2 remainder body of loop is skipped by continue statement, therefore if condition i == 8 is never true as when i is 8, remainder body of loop is skipped by continue statements of first if. Control comes to print statement only in cases when i is odd.

Q.60 **Explanation :** Initially x = 10, the x + 1 is evaluated at the case 11 gets matched due to which y becomes equal to 0 but as there is no break statement after case 11, y is initialized to 1 and then the default case will be executed and the value of y = y + 1 i.e. 1 + 1 = 2.

❑❑❑

UNIT - II

2 Classes and Objects

Syllabus

Class Fundamentals, Creating Objects, Accessing Class members, Assigning Object reference variables, Methods, Constructors, using objects as parameters, Argument passing, returning objects, Method Overloading, static members, Nesting of Methods, this keyword, Garbage collection, finalize methods, final variables and methods, final class.

Contents

2.1 Class Fundamentals

Object oriented programs are often easy to understand, correct and modify. Java is a true object oriented programming language. In any object oriented programming language, anything is encapsulated in a class. The objects use the methods of a class so that two objects can communicate with each other.

In this chapter we will focus on three important issues i.e. classes, object and methods.

2.1.1 Defining a Class

- A class can be defined as an entity in which data and functions are put together.
- The concept of class is similar to the concept of structure in C.
- A class is declared by using the keyword **class.** The general form of class is

```
class classname
{
        type variable1;
        type variable2;
        //...
        type method1(parameter-list)
        {
                ...
                ...
        }
        type method2(parameter-list)
        {
                ...
                ...
        }
        ...
        type methodn(parameter-list)
        {
                ...
                ...
        }
}
```

- The class template is as shown in the following figure

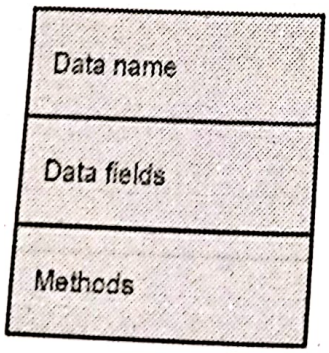


Fig. 2.1.1 Class template

2.1.2 Fields Declaration

- The data lies within the class and the data fields are accessed by the methods of that class.
- The data fields are also called as **instance variables** or **member variables** because they are created when the objects get instantiated.
- For example –

```
class Test
    {
        int a;
        int b;
    }
```

2.1.3 Method Declaration

- In object oriented programming any two objects communicate with each other using methods.
- All the methods have the same general form as the method **main()**. Most of the methods are specified as either **static** or **public.**
- Java classes do not need the method **main,** but if you want particular class as a starting point for your program then the **main** method is included in that class.
- The general form of method is -

```
type method(parameter-list)
{
        ...
        ...          } body of method
}
```

- The **type** specifies the type of data returned from the method. If the method doe not return anything then its data type must be **void.**
- For returning the data from the method the keyword **return** is used.

2.2 Creating Objects

- Objects are nothing but the instances of the class. Hence a block of memory ge allocated for these instance variables.

For creating objects in Java the operator new is used.

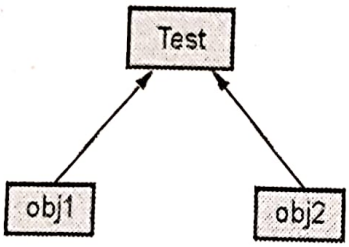
```
Test obj ; // Declaration of object obj.
obj = new Test () ; // obj gets instantiated for class Test
```

We instantiate one object obj; it can be represented graphically as -

Now if we create two objects then it can be

```
Test obj1, obj2;
obj1 = new Test ();
obj2 = new Test ();
```

It can be graphically as shown below -



Difference between Class and Object

Sr. No.	Class	Object
1.	For a single class there can be **any number of objects**. For example - If we define the class as River then Ganga, Yamuna, Narmada can be the objects of the class River.	There are many objects that can be **created from one class**. These objects make use of the methods and attributes defined by the belonging class.
2.	The **scope** of the class is **persistent** throughout the program.	The objects can be **created** and **destroyed** as per the requirements.
3.	The class **cannot be initialized** with some property values.	We can **assign** some property values to the objects.
4.	A class has **unique name.**	Various objects having **different names** can be created for the same class.

Review Questions

1. *Explain the concept of object with suitable example.*
2. *Give the difference between class and object.*

2.3 Accessing Class members

- There are two types of class members - The **data members** and the **method.**
- These members can be accessed using **dot** operator.
- For accessing the data members the **syntax** is -

```
name_of_object.variable_name=value;
```

- For accessing the method of the class the syntax is -

```
name_of_object.methodname(parameter list);
```

- For example –

```
obj.name="XYZ"
obj.display();
```

- Suppose we wish pass the parameters to the method then first we will create the object for the class as follows -

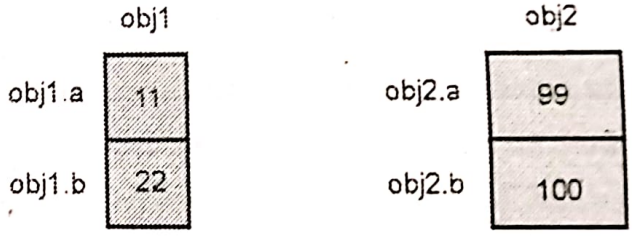
```
Test obj1=new Test();
Test obj2=new Test();
```

Now two objects are created namely **obj1** and **obj2.**

```
obj1.get_val(11,22);
obj2.get_val(99,100);
```

Suppose by this method we assign values to two variables a and b then -

It can be graphically represented as -



Let us see the simple Java program which makes use of classes and objects -

Java Program [Rectangle.java]

```
/*This is a Java program which shows the
use of class in the program
*/
class Rectangle
{
int height;
int width;
void area()
{

int result=height*width;
System.out.println("The area is "+result);
```

Class with attributes and methods defined within it.

```
}
}
class classDemo
{
public static void main(String args[])
{
Rectangle obj=new Rectangle();


obj.height=10;//setting the attribute values
obj.width=20;//from outside of class
obj.area();//using object method of class is called
}
}
```

Another class in which main () function is written.

Output

```
The area is 200
```

Program Explanation : In the above program,

(1) We have used two classes one class is **classDemo** which is our usual one in which the main function is defined and the other class is **Rectangle.**

(2) In this class, we have used **height** and **width** as attributes and one method **area()** for calculating area of rectangle.

(3) In the main function we have declared an object of a class as

```
Rectangle obj=new Rectangle();
```

And now using **obj** we have assigned the values to the attributes of a class. The operator **new** is used to create an object. The objects access the data fields and methods of the class using the **dot operator.** This operator is also known as the **object member access operator.** Thus data field *height* and *width* are called as **instance variables.** And the method, *area* is referred as **instance method.** The object on which the instance method is invoked is known as **calling object.**

Concept of Data Hinding :

The use of class allows to hide the important data from outsider of the class. If the data being hidden declared as public, then only these members will be accessible from outside class.Thus class helps in unauthorized access to its data members.

Review Question

1. *What is class ? How it accomplishes data hiding ?*

2.4 Assigning Object via Reference Variables

- Reference variables are those variables that contain references to the objects.
- Using reference variables we can access the data members and member functions of the object.
- The reference variable can be created using following syntax

```
Class_name Object_Reference_variable = new Class_name();
```

- For example - Consider that we have to create an object variable of object **Student** then

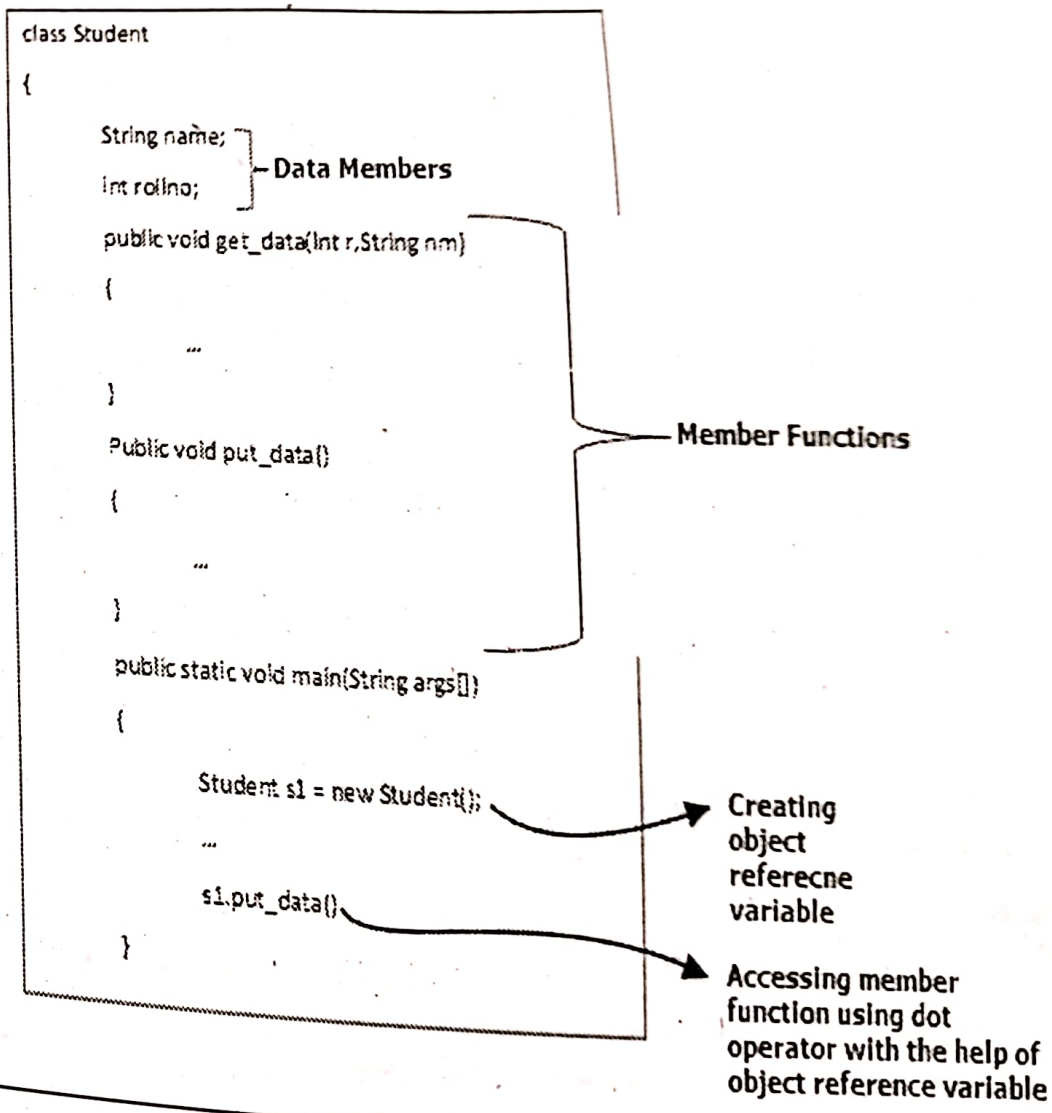
```
Student S = new Student();
```

Note that, here **S** is an object variable that can access the data member and member function of Student object. The S holds the reference to a **Student** object.

- We can access the data members of object using **dot operator** with the help of **object reference variable.** Hence The dot operator is also called as **object member access operator.**
- Syntax of accessing object's member

```
Object_variable_name.object_member;
```

- **For example**

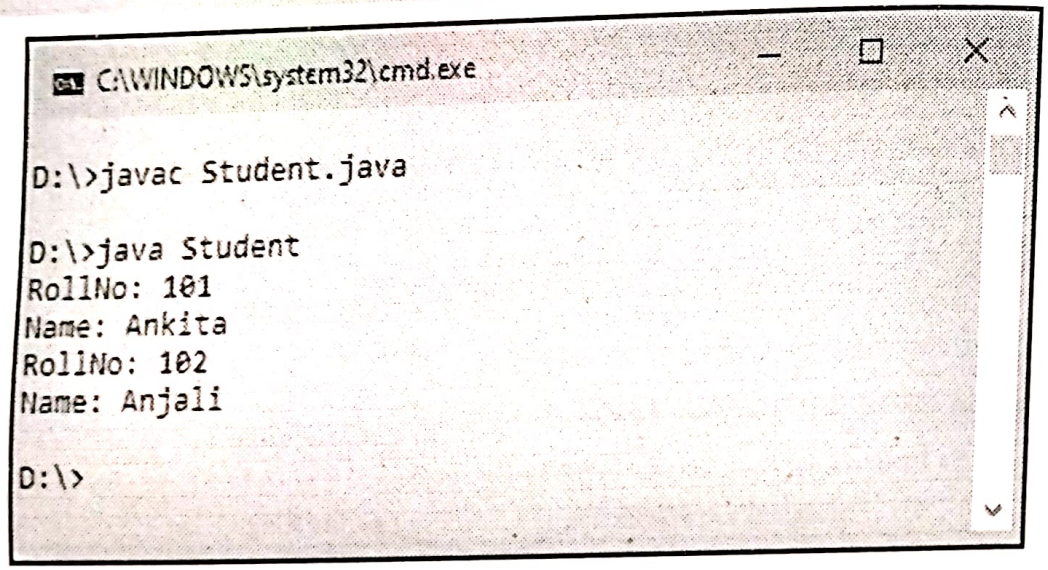


Following program illustrates above concept.

Java Program

```
public class Student
{
    String name;
    int rollno;
    public void get_data(int r, String nm)
    {
        rollno = r;
        name = nm;
    }
    public void put_data()
    {
        System.out.println("RollNo: "+rollno);
        System.out.println("Name: "+name);
    }
    public static void main(String args[])
    {
        Student s1 = new Student();
        Student s2 = new Student();
        s1.get_data(101,"Ankita");
        s2.get_data(102,"Anjali");
        s1.put_data();
        s2.put_data();
    }
}
```

Output



Review Question

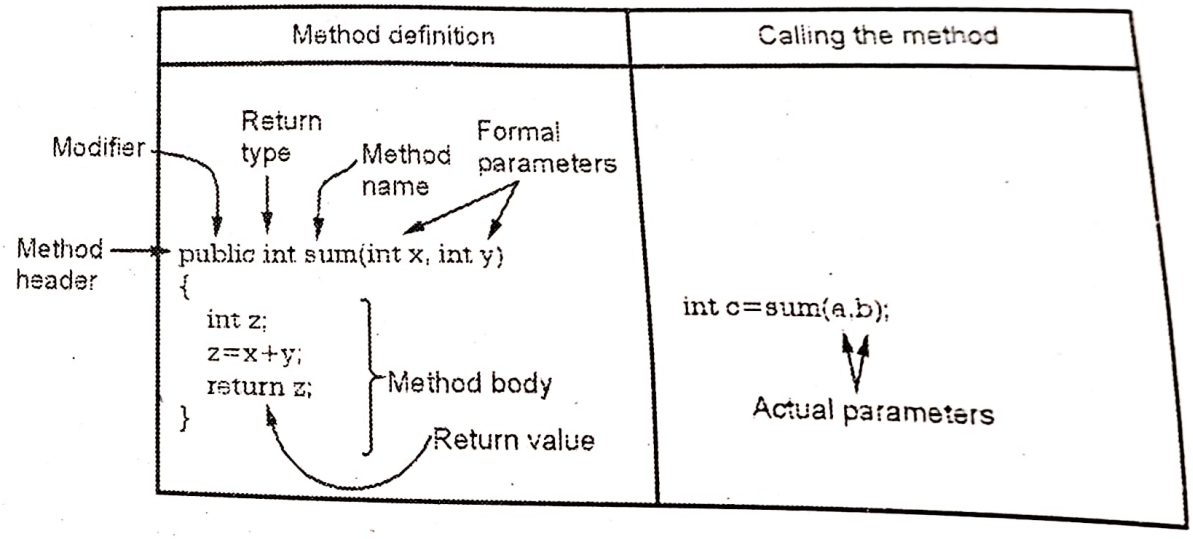
1. *What is reference variable ? How to access the contents of class using reference variable.*

2.5 Methods

- Methods are created to define the block of statements that are used repeatedly in the program.
- **Method definition** consists of method name, parameters, return value type, and body.
- **Syntax of Method Definition**

```
Modifier return_typeMethod_name(list of parameters)
{
    //Method Body
}
```

- For example - Here is an illustration which represents the method definition and calling of a method



Example 2.5.1 *Write a Java program for finding minimum of two numbers using function.*

Solution :

```
importjava.util.*;
classMinValueDemo
{
    public static int MinValue(int x,int y)
    {
            if(x < y)
                return x;
            else
    }               return y;
    public static void main(String args[])
    {
            Scanner input = new Scanner(System.in);
```

Method definition

```
        System.out.println("Enter value of first number: ");
        int a = input.nextInt();
        System.out.println("Enter value of second number: ");
        int b = input.nextInt();
        int c = MinValue(a,b);
        System.out.println("The minimum value is: "+c);
    }
}
```

Method call

Output

```
Enter value of first number:
10
Enter value of second number:
20
The minimum value is: 10
```

2.6 Constructors

- **Definition** : The constructor is a specialized method for **initializing objects.**
- Name of the constructor is same as that of its class name. In other words, the name of the constructor and class name is same.

```
Class Test
{
        Test()
        {
                //body of constructor
        }
}
```

Class Name
Constructor

- Whenever an object of its associated class is created, the constructor is invoked automatically.
- The constructor is called constructor because it creates values for data fields of class.
- Example -

Java Program[Rectangle1.java]

```
public class Rectangle1 {
    int height;
    int width;
    Rectangle1()
    {
        System.out.println(" Simple constructor: values are initialised...");
        height=10;
        width=20;
    }
    Rectangle1(int h,int w)
```

Data fields

Simple constructor

```
    {
        System.out.println(" Parameterised constructor: values are initialised...");
        height=h;
        width=w;
    }
    void area()
    {
        System.out.println("Now, The function is called...");
        int result=height*width;
        System.out.println("The area is "+result);
    }
}
class Constr_Demo {
public static void main(String args[])
{
    Rectangle1 obj=new Rectangle1();
    obj.area();//call the to method
    Rectangle1 obj1=new Rectangle1(11,20);
    obj1.area();//call the to method

}
}
```

Method defined

Object created and simple constructor is invoked

Object created and parameterised constructor is invoked

Output

Note the method of running the program

```
F:\test>javac Rectangle1.java
F:\test>java Constr_Demo
 Simple constructor: values are initialised...
Now, The function is called...
The area is 200.
 Parameterised constructor: values are initialised...
Now, The function is called...
The area is 220

F:\test>
```

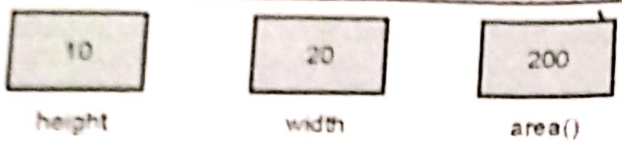
Program Explanation

In above program,

(1) There are two classes - **Rectangle1** and **Constr_Demo.**

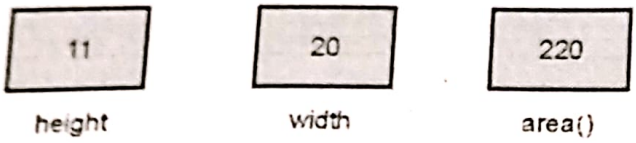
(2) In the **Rectangle1** class, data fields, method and constructor is defined.

(3) In the **Constr_Demo** class the objects are created. When an object obj gets crea then the simple constructor **Rectangle1()** gets invoked and the data fields he and width gets initialized.



Hence the **area** function will compute the area=10*20=200.

(4) When an object **obj1** gets created, then the parameterised constructor **Rectangle1(11, 20)** gets invoked and the data fields height and width gets initialised.



Hence the **area** function will compute the area = 11*20 = 220.

(5) The class **Constr_Demo** is called **main class** because it consists of **main** function.

Example 2.6.1 *Differentiate between constructor and method.*

Solution :

Sr. No.	Constructor	Method
1.	The name of the constructor must be same as the class name.	The name of the method should not be the class name. It can be any with alphanumeric or alphabetic characters with restricted character length. The name must not start with digit or special symbols. Only underscore is allowed.
2.	It does not return anything hence no return type.	It can return and hence it has a return type. If method does not return anything then the return type is void.
3.	The purpose of constructor is to initialize the data members of the class.	The method is defined to execute the core logic of the class.
4.	The constructor is invoked implicitly at the time of object creation.	The method must be called explicitly using the object name and dot operator.

1 Properties of Constructor

Name of constructor must be the same as the name of the class for which it is being used.

The constructor must be declared in the **public** mode.

3. The constructor gets invoked automatically when an object gets created.
4. The constructor should not have any return type. Even a **void** data type should not be written for the constructor.
5. The constructor cannot be used as a member of union or structure.
6. The constructors can have **default arguments.**
7. The constructor **cannot be inherited.** But the derived class can invoke the constructor of base class.
8. Constructor can make use of **new** or **delete** operators for allocating or releasing memory respectively.
9. Constructor can not be **virtual.**
10. Multiple constructors can be used by the same class.
11. When we declare the constructor explicitly then we must declare the object of that class.

2.6.2 Types of Constructor

Various types of constructors are -

1. Default constructor 2. Parameterized constructor

Let us discuss them in detail -

1) Default Constructor

- The default constructor is a simple constructor in which no parameter is passed the constructor function.
- This constructor is automatically called when the object of the class is created.
- **Example Program**

```
class Test
{
    Test() <-- This is default constructor
    {
        System.out.print("This message is from constructor");
    }
    public static void main(String args[])
    {
        Test t=new Test();
    }
}
```

Output

```
This message is from constructor
```

2) Parameterized Constructor

Parameterized constructor is a constructor in which the parameters are passed to the constructor function.

Example Program

```
class Person
{
        int id;
        String city;
        Person(int i,String c)
        {
                id=i;
                city=c;
        }
        void display()
        {
                System.out.print("ID: "+id);
                System.out.println(" City: "+city);
        }
        public static void main(String args[])
        {
                Person p1=new Person(10,"Ankita");
                Person p2=new Person(20,"Prajka");
                p1.display();
                p2.display();
        }
}
```

Output

```
ID: 10 City: Ankita
ID: 20 City: Prajka
```

Program Explanation : In above program,

1) We have created parameterized constructor by passing the parameters id and Name. Note that these constructors will be called at the time of creation of objects p1 and p2.
2) Then using these objects we can invoke the method **display().**

Example 2.6.2 *There is no destructor in Java. Justify.*

Solution : There are two purposes of destructor - 1. Freeing up the memory allocated for the objects. 2. Cleaning up resources like closing the open file stream. The garbage collection mechanism of Java takes care of these two tasks automatically. Hence there is no need for having destructor in Java.

Example 2.6.3 *Define the Rectangle class that contains : Two double fields x and y that specify the center of the rectangle, the data field width and height. A no-arg constructor that creates the default rectangle with (0,0) for (x,y) and 1 for both width and height.*

A parameterized constructor creates a rectangle with the specified x, y, height and width.

A method getArea() that returns the area of the rectangle.

A method getPerimeter() that returns the perimeter of the rectangle. A method contains(double x, double y) that returns true if the specified point (x, y) is inside this rectangle.

Write a test program that creates two rectangle objects. One with default values and other with user specified values. Test all the methods of the class for both the objects.

Solution :

```
class Rectangle
{
  double Center_X,Center_Y,width,height;
  double X_Left,X_Right,Y_UP,Y_Down;
  Rectangle() // No Argument constructor
  {
    Center_X=Center_Y=0;
    width=height=1;
  }
  Rectangle(double x,double y,double h,double w)      // Parameterized constructor
  {
    Center_X=x;
    Center_Y=y;

    height=h;
    width=w;
  }
  double getArea()
  {
    return (height*width);
  }
  double getPerimeter()
  {

    return ( 2*(height+width) );
  }
  boolean contains(double x,double y)
  {
    X_Left=( Center_X - (width/2) );
    X_Right=( Center_X + (width/2) );
    Y_UP=(Center_Y + (height/2));
```

```
        Y_Down=(Center_Y - (height/2));
        if( (x > X_Left && x < X_Right) && (y < Y_UP && y > Y_Down))
          return true.    //point is inside rect.
        else
          return false.   //point is outside rect.
      }
    }
    class testclass
    {
      public static void main(String args[])
      {
        Rectangle obj1 = new Rectangle();
        Rectangle obj2 = new Rectangle(300,300,100,200);
        System.out.println();
        System.out.println("\t\t Rectangle 1" );
        System.out.println(" Area "+obj1.getArea());
        System.out.println(" Perimeter "+obj1.getPerimeter());
        if(obj1.contains(10,20))
          System.out.println(" Contains (10,20) is inside the Rectangle ");
        else
          System.out.println(" Contains (10,20) is outside the Rectangle ");
        System.out.println("----------");
        System.out.println();
        System.out.println("\t\t Rectangle 2" );
        System.out.println(" Area "+obj2.getArea());
        System.out.println(" Perimeter "+obj2.getPerimeter());
        if(obj2.contains(250,310))
          System.out.println(" Contains (250,310) is inside the Rectangle ");
        else
          System.out.println(" Contains (250,310) is outside the Rectangle ");

      }
    }
```

Example 2.6.4 *State whether any error exists in the following code. If so, correct the error and give output.*

```
class Test {
public static void main(String args[]) {
A a = new A();
a.print();
}
}
class A {
```

```
String s;
A(String s) {
this.s=s;
}
public void print() {
System.out.println(s);
}
}
```

Solution : The error at the line where the instance a for the class A is created. Some string must be passed as an argument to the constructor. The corrected code is as given below -

```
class Test {
public static void main(String args[]) {
A a = new A("JAVA");
a.print();
}
}
class A {
String s;
A(String s) {
this.s=s;
}
public void print() {
System.out.println(s);
}
}
```

Example 2.6.5 *Implement a class Student. A Student has a name and total quiz score. Supply an appropriate constructor and methods getName(), addQuiz(int score), getTotalScore() and getAverageScore(). To Compute the latter, you also need to store number of quizzes that the student took.*

Solution :

```
class Student
{
        private String name;
        private int total_quizScore;
        private int quizCount;
        Student(String n)
        {
                name = n;
                total_quizScore = 0;
```

```
        quizCount = 0;
    }
    public String getName()
    {
        return name;
    }
    public void addQuiz(int score)
    {
        total_quizScore = total_quizScore + score;
        quizCount = quizCount + 1;
    }
    public int getTotalScore()
    {
        return total_quizScore;
    }
    public double getAverageScore()
    {
        return total_quizScore / quizCount;
    }
}
class StudentDemo
{
    public static void main(String args[])
    {
        Student obj = new Student("AAA");
        int no_of_quizzes = 5;
        int Score[] = { 50,70,60,55,68 };
        for (int i = 0; i<no_of_quizzes; i++)
            obj.addQuiz(Score[i]);
        System.out.print("\n Student's Name: " + obj.getName());
        System.out.print("\n Total number of Quizzes: " + no_of_quizzes);
        System.out.print("\n Student's Total Score: " + obj.getTotalScore());
        System.out.print("\n Student's Average Score: " + obj.getAverageScore());
    }
}
```

Output

```
Student's Name: AAA
Total number of Quizzes: 5
Student's Total Score: 303
Student's Average Score: 60.0
```

2.7 Using objects as parameters

- The object can be passed to a method as an argument. Using dot operator the object's value can be accessed.
- Such a method can be represented syntactically as -

```
Data_Type name_of_method(object_name)
{
        //body of method
}
```

- Following is a sample Java program in which the method **area** is defined.The parameter passed to this method is an **object.**

Java Program[ObjDemo.java]

```
public class ObjDemo {
        int height;
        int width;
        ObjDemo(int h,int w)
        {
                height=h;
                width=w;
        }
        void area(ObjDemo o)
        {
                int result=(height+o.height)*(width+o.width);
                System.out.println("The area is "+result);
        }

}
class Demo {
public static void main(String args[])
{
        ObjDemo obj1=new ObjDemo(2,3);
        ObjDemo obj2=new ObjDemo(10,20);
        obj1.area(obj2);

}
}
```

height=2 and o.height=10
width=3 and o.width=20
Hence,
result=(2+10)*(3+20)
=12*23
=276

Method with object as parameter

Output

```
F:\test>javac ObjDemo.java
F:\test>java Demo
The area is 276
```

- When an object needs to be passed as a parameter to the function then constructor is built. Hence we have to define **constructor** in which the values of the object are used for initialization.

2.8 Returning objects

- We can return an object from a method. The data type such method is a class type.

Java Program[ObjRetDemo.java]

```
public class ObjRetDemo {
        int a;
        ObjRetDemo(int val)
        {
                a=val;
        }
        ObjRetDemo fun()
        {
                ObjRetDemo temp=new ObjRetDemo(a+5); //created a new object
                return temp; //returning the object from this method
        }
}
class ObjRet {
 public static void main(String args[])
{
        ObjRetDemo obj2=new ObjRetDemo(20);
        ObjRetDemo obj1;
        obj1=obj2.fun();//obj1 gets the value from object temp
       System.out.println("The returned value is = "+obj1.a);

}
}
```

Data type of this method is name of the class

Output

```
F:\test>javac ObjRetDemo.java

F:\test>java ObjRet
The returned value is = 25
```

Example 2.8.1 *Write Java program for computing Fibonacci series.*

Solution :

```
/**********************************************************************
Program for computing the number in the fibonacci series at certain location.
For example the eighth number in Fibonacci series will be 21.
The fibonacci series is 1 1 2 3 5 8 13 21 34...
**********************************************************************/
```

```
import java.io.*;
import java.util.*;
class Fibonacci
{
public int fib(int n)
{
int x,y;
if(n<=1)
return n;
x=fib(n-1);
y=fib(n-2);
return (x+y);
}
}//end of class
class FibonacciDemo
{
public static void main(String[] args)throws IOException
{
        Fibonacci f=new Fibonacci();
        int n=8;
        System.out.println("\nThe number at "+n+" is "+f.fib(n));
 }
}
```

Output

```
D:\>javac FibonacciDemo.java
D:\>java FibonacciDemo
The number at 8 is 21
```

Review Question

1. *Explain how to pass object to a method ?*

2.9 Argument Passing

- Argument or Parameter can be passed to the function by two ways -

  1. Call by value 2. Call by reference

- In the **call by value** method the value of the actual argument is assigned to formal parameter. If any change is made in the formal parameter in the subrout definition then that change does not reflect the actual parameters.
- Following Java program shows the use of parameter passing by value -
- **Example : Java Program**

```
public class ParameterByVal
{
 void Fun(int a,int b)
```

```
{
a=a+5;
b=b+5;
}
public static void main(String args[])
{
ParameterByVal obj1=new ParameterByVal();
int a,b;
a=10;b=20;
System.out.println("The values of a and b before function call");
System.out.println("a= "+a);
System.out.println("b= "+b);
obj1.Fun(a,b);
System.out.println("The values of a and b after function call");
System.out.println("a= "+a);
System.out.println("b= "+b);

}
}
```

Output

```
The values of a and b before function call
a= 10
b= 20
The values of a and b after function call
a= 10
b= 20
```

Program Explanation

- The above Java program makes use of the parameter passing by value. By this approach of parameter passing we are passing the actual values of variables a and b.
- In the function **Fun** we are changing the values of **a** and **b** by adding 5 to them. But these incremented values will remain in the function definition body only. After returning from this function these values will not get preserved. Hence we get the same values of a and b before and after the function call.
- On the contrary the parameter passing by reference allows to change the values after the function call. But use of variables as a parameter does not allow to pass the parameter by reference.
- For passing the **parameter by reference** we **pass the object** of the class as a parameter.
- Creating a variable of class type means creating reference to the object. If any changes are made in the object made inside the method then those changes get preserved and we can get the changed values after the function call.

2.10 Method Overloading

Overloading is a mechanism in which we can use many methods having the same function name but can pass different number of parameters or different types of parameter.

For example :

```
int sum(inta,int b);
double sum(double a,double b);
int sum(inta,intb,int c);
```

That means, by overloading mechanism, we can handle **different number of parameters** or **different types** of parameter by having the same method name. Following Java program explain the concept overloading –

Java Program[OverloadingDemo.java]

```
public class OverloadingDemo {
public static void main(String args[]) {
System.out.println("Sum of two integers     ");
Sum(10,20);<---------- line A
System.out.println("Sum of two double numbers     ");
Sum(10.5,20.4); <---------- line B
System.out.println("Sum of three integers    ");
Sum(10,20,30); <---------- line C
 }
public static void Sum(int num1,int num2)
{
  int ans;
  ans=num1+num2;
System.out.println(ans);
 }
public static void Sum(double num1,double num2)
{
  double ans;
  ans=num1+num2;
System.out.println(ans);
 }
public static void Sum(int num1,int num2,int num3)
{
  int ans;
  ans=num1+num2+num3;
System.out.println(ans);
 }
}
```

Output

```
F:\test>javac OverloadingDemo.java

F:\test>java OverloadingDemo
Sum of two integers
30
Sum of two double numbers
30.9
Sum of three integers
60
```

Program Explanation

In above program, we have used three different methods possessing the same name. Note that,

on line A

We have invoked a method to which two integer parameters are passed. Then compiler automatically selects the definition **public static void Sum(double num1,double num2)** to fulfil the call. Hence we get the output as 30 which is actually the addition of 10 and 20.

on line B

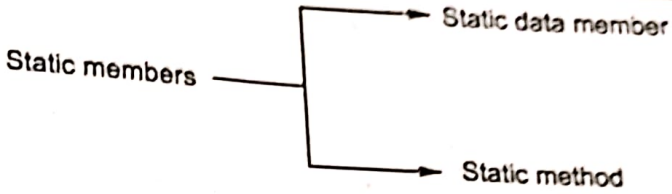
We have invoked a method to which two double parameters are passed. Then compiler automatically selects the definition **public static void Sum(double num1,double num2)** to fulfil the call. Hence we get the output as 30.9 which is actually the addition of 10.5 and 20.4.

on line C

We have invoked a method to which the three integer parameters are passed. Then compiler automatically selects the definition **public static void Sum(int num1,int num2,int num3)** to fulfil the call. Hence we get the output as 60 which is actually the addition of 10, 20 and 30.

2.11 Static Members

- The static members can be **static data member** or **static method.**
- The static members are those members which can be accessed **without using object.**
- Following program illustrates the use of static members.



Example Program

```
class StaticProg
{
 static int a=10;
 static void fun(int b)
 {

   System.out.println("b= "+b);
   System.out.println("a= "+a);


 }
}
class AnotherClass
{
 public static void main(String[] args)
 {
   System.out.println("a= "+StaticProg.a);
   StaticProg.fun(20);

 }
}
```

Output

```
a= 10
b= 20
a= 10
```

Program Explanation

In above program, we have declared one static variable **a** and a static member functic **fun()**. These static members are declared in one class **StaticProg**. Then we have writt one more class in which the **main** method is defined. This class is named **AnotherClass**. From this class the static members of class **StaticProg** are accessed *witho using any object.* Note that we have simple used a syntax

```
ClassName.staticMember
```

Hence by using **StaticProg.a** and **StaticProg.fun** we can access static members

Restrictions on Static

- The static methods must can access only static data.

```
thisDemo obj1=new thisDemo();
thisDemo obj2=new thisDemo();
obj1.Sum(10,20);
obj1.display();
obj2.Sum(40,50);
obj2.display();
}
}
```

Output

```
a= 10
b= 20
Sum= 30

a= 40
b= 50
Sum= 90
```

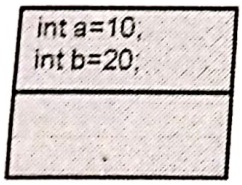
Program Explanation

In above program,

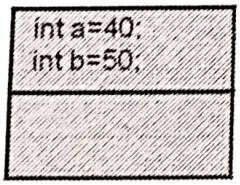
(1) We have created two objects for the class - **obj1** and **obj2.**

(2) The value of a is assigned to be 10 and value of b as 20 for the object **obj1** using the keyword **this.** Similarly the obj2 posses the values of a and b as 40 and 50 respectively.

(3) Note that the keyword **this** is useful to refer the current object.



this.a=a;
this.b=b;
this statement makes the current value of a(which belongs to **obj1**) as 10. Similarly the value of b is initialised with 20



this.a=a;
this.b=b;
this statement makes the current value of a(which belongs to **obj2**) as 40. Similarly the value of b is initialised with 50

2.14 Garbage Collection and Finalize Methods

- Garbage collection is a method of automatic memory management.
- It works as follows -
  1. When an application needs some free space to allocate the nodes and if there is no free space available to allocate the memory for these objects then a system routine called **garbage collector** is called.

2. This routine then searches the system for the free space. The free space is then made available for reuse.

- Java used **finalize()** method for garbage collection.
- To add finalizer to a class simply define the **finalize** method. The syntax to finalize() the code is -

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

- Even though we allocate the memory and then forget to deallocate it then the objects that are no longer is used get freed. Inside the finalize() method you will specify those actions that must be performed before an object is destroyed. The garbage collector runs periodically checking for objects that are no longer referenced by any running state or indirectly though other referenced objects.
- Note that finalize() method is called just before the garbage collection.

Review Question

1. *How Garbage collection is done in Java. Also explain finalize method.*

2.15 The final Variables and Methods

- A variable can be declared as final. If a particular variable is declared as final ther cannot be modified further.
- The final variable is always a constant. For example -

```
final int a=10;
```

- The final keyword can also be applied to the method. When **final** keywor applied to the method, the method overriding is avoided. That means the met those are declared with the keyword final cannot be overridden.
- Consider the following Java program which makes use of the keyword **fin** declaring the method -
- **Example Program**

```
class Test
{
    final void fun()
    {
        System.out.println("\n Hello, this function declared using final");
}
    }
    class Test1 extends Test
```

```
{
    final void fun()
    {
      System.out.println("\n Hello, this another function");
    }
}
```

Output

```
Test.java:10: fun() in Test1 cannot override fun() in Test; overridden method is final
final void fun()
           ^
1 error
```

Program Explanation : The above program,

1) on execution shows the error. Because the method fun is declared with the keyword final and it cannot be overridden in derived class.

Three uses of Final Keyword

1) The final variables can not be modified.
2) The final method cannot be overridden.
3) The final class can not be inherited.

Review Questions

1. *Describe final method and final variable with respect to inheritance.*
2. *State three uses of final keyword.*

2.16 Multiple Choice Questions

Q.1 The keyword used for declaring the class is _____.

[a] class [b] object
[c] myclass [d] Javaclass

Q.2 The data field in class is also known as

[a] private members [b] variables
[c] instance variables [d] none of these

Q.3 The variables declared in a class for the use of all methods of the class are called____.

[a] reference variables [b] objects
[c] instance variables [d] none of these

Q.4 Classes are useful because they _____.

[a] permit data to be hidden from other classes

[b] can closely model objects in the real world

[c] brings together all aspects of an entity in one place

[d] all of these

Q.5 Choose the incorrect statement.

[a] There can be only one main method in the program..

[b] Every class must contain the main() method.

[c] main() method must be public in class.

[d] Applets do not contain main method.

Q.6 What is the output of the following code ?

```
class Test
{
    public static void main(String args[])
    {
        int a = 100;
        if (a == 100)
        {
            int a = 200;
            System.out.println(a);
        }
    }
}
```

[a] 100 [b] 200

[c] 0 [d] Syntax error

Q.7 Which of the following is the correct statement to create an object of Test class ?

[a] Test obj=new object(); [b] Test obj=new Test();

[c] Test obj()=new Test(); [d] Test obj()=new Test();

Q.8 The new keyword is used to ____.

[a] call a method of a class [b] allocate memory to an object

[c] release memory of an object [d] none of these

Q.9 What is the output of the following code ?

```
class Circle
{
    int radius;
}
class Test
```

```
{
    public static void main(String args[])
    {
        Circle obj = new Circle();
        obj.radius = 10;
        double area = 3.14*obj.radius*obj.radius;
        System.out.print(area);
    }
}
```

[a] 10 [b] 314

[c] 31.4 [d] Syntax error

Q.10 Which of the following is used to allocate the memory for object ?

[a] malloc [b] alloc

[c] new [d] calloc

Q.11 What is the output of the following code?

```
class Circle
{
    int radius;
}
class Test
{
    public static void main(String args[])
    {
        Circle obj1 = new Circle();
        Circle obj2 = new Circle();
        obj1.radius = 10;
        obj2=obj1;
        System.out.print(obj2.radius);
    }
}
```

[a] 10

[c] Garbage Value [b] 0

[d] None of these

Q.12 What is the output of the following code ?

```
class Circle
{
    int radius;
}
class Test
{
    public static void main(String args[])
    {
        Circle obj = new Circle();
        System.out.print(obj);
    }
}
```

[a] 0 [b] garbage value

[c] Syntax error [d] 1

The **obj** is nothing but the instance of a class. Using print statement we are actually printing the address of obj.

Q.13 The dot operator connects the ____.

[a] class object and class

[b] the class and member of that class

[c] the class object and members of that class

[d] none of these

Q.14 Constructor is a special type of ____.

[a] class [b] object

[c] method [d] none of these

Q.15 ___ is a method having same name as its class name.

[a] class [b] finalize

[c] new [d] constructor

Q.16 What will be the output of the following code ?

```
class Interest
{
        double p;
        double n;
        double r;
        double result;
        Interest()
        {
                p = 100000;
                n = 5;
                r = 7;
        }
        void result()
        {
                result = (p*n*r)/100;
        }
}
class Test
{
        public static void main(String args[])
        {
                Interest obj = new Interest();
                obj.result();
                System.out.println(obj.result);
        }
}
```

[a] 3500 [b] 35000

[c] 30000 [d] Syntax error

Q.17 Choose the correct statement.

I. Default constructor is called at the time of declaration of the object if constructor has not been defined.

II. We can pass parameters to the constructor

[a] Only I [b] Only II

[c] Both I and II [d] Neither I nor II

Q.18 The constructor must be declared in ____ mode

[a] public [b] private

[c] protected [d] none of these

Q.19 public class A { }

What is the prototype of the default constructor ?

[a] public A(void) [b] A()

[c] A(void) [d] public A()

Q.20 The implicit return type of a constructor is ____.

[a] void [b] a class object in which it is defined.

[c] there is no return type. [d] none of the above

Q.21 Constructor ____.

[a] can not be inherited

[b] can not be virtual

[c] should not have any return type

[d] all of these

Q.22 The default constructor has ____.

[a] has no argument

[b] has one argument

[c] has one argument and no return type

[d] none of these

Q.23 The return type of constructor is ____.

[a] int [b] void

[c] float [d] none of these

Q.24 ___ keyword is used to refer to the current object.

[a] super [b] this

[c] new [d] volatile

Q.25 The keyword used by a method to refer to the object that invoked it is ____.

[a] catch [b] abstract

[c] this [d] import

Q.26 ___ is a technique of automatic memory management in Java.

[a] Constructor [b] Destructor

[c] Garbage collection [d] None of these

Q.27 ___ function is used to perform when the object is destroyed.

[a] delete() [b] finalize()

[c] main() [d] none of these

Q.28 What allows the programmer to destroy the object obj ?

[a] obj.delete() [b] obj.finalize()

[c] obj.final

[d] Only the garbage collection system can destroy an object.

Q.29 During the collection phase the garbage collector makes the node free. This is called ____.

[a] marking [b] sweeping

[c] collection [d] none of these

Q.30 ___ must be avoided for better performance in Garbage collection.

[a] Marking [b] Sweeping

[c] Collection [d] Thrashing

Q.31 ___ method is called just before the Garbage collection.

[a] main() [b] constructor

[c] finalize() [d] None of these

Q.32 The technique in which two or more methods are declared with the same name bu different parameter declarations is called ____.

[a] method overloading [b] method overriding

[c] constructor [d] This facility is not available in Java

Q.33 ___ can be overloaded.

[a] function [b] constructor

[c] all of these [d] none of these

Q.34 What is the output of the following code ?

```
class A
{
    int x;
    int y;
    void display(int a)
    {
        x = a + 1;
    }
    void display(int a , int b)
    {
        x = a + 2;
    }
}
class Test
{
    public static void main(String args[])
    {
        A obj = new A();
        int a = 0;
        obj.display(10,20);
        System.out.println(obj.x);
    }
}
```

[a] 21 [b] 12

[c] 22 [d] Syntax error

Q.35 What is the output of the following code ?

```
class A
{
    int x;
    double y;
    void function(int a , int b)
    {
        x = a + b;
    }
    void function(double c , double d)
    {
        y = c + d;
    }
    A()
```

```
    {
        this.x = 0;
        this.y = 0;
    }
}
class Test
{
    public static void main(String args[])
    {
        A obj = new A();
        int a = 10;
        double b = 11.11;
        obj.function(a, a);
        obj.function(b, b);
        System.out.println(obj.x + " " + obj.y);
    }
}
```

[a] 21 21.11 [b] 0 0

[c] 20 22.22 [d] None of these

Q.36 What will happen if we try to overload the main method in a class ?

[a] compile time error [b] runtime error

[c] main method gets overloaded [d] none of these

Q.37 Which of the method must be made static ?

[a] delete() [b] finalize()

[c] main() [d] none of these

Q.38 Which modifier can not be used for constructors ?

[a] public [b] private

[c] protected [d] static

Q.39 The main method should be static for the reason _____.

[a] It can be accessed easily by the class loader.

[b] It can be accessed by every method or variable without any hindrance.

[c] It can be executed without creating any instance of the class.

[d] None of the above.

Q.40 Which of the following statements regarding static methods are correct ?

1. Static methods are difficult to maintain, because you can not change t implementation.
2. Static methods can be called using an object reference to an object of the clas which this method is defined.

[a] only 1 [b] only 2

[c] Both 1 and 2 [d] neither 1 nor 2

Q.41 Which of the following statements regarding static methods are correct ?

1. Static methods are always public, because they are defined at class-level.
2. Static methods do not have direct access to non-static methods which are defined inside the same class.

[a] only 1 [b] only 2

[c] Both 1 and 2 [d] neither 1 nor 2

Q.42 ___ keyword is used to prevent the contents of the variable being modied.

[a] static [b] final

[c] finally [d] constant

Q.43 ___ can not be declared static.

[a] method [b] class

[c] object [d] none of these

Answer Keys for Multiple Choice Questions :

Q.1	a	Q.2	c	Q.3	c	Q.4	d
Q.5	b	Q.6	d	Q.7	b	Q.8	b
Q.9	b	Q.10	c	Q.11	a	Q.12	b
Q.13	c	Q.14	c	Q.15	d	Q.16	b
Q.17	c	Q.18	a	Q.19	d	Q.20	b
Q.21	d	Q.22	a	Q.23	d	Q.24	b
Q.25	c	Q.26	c	Q.27	b	Q.28	d
Q.29	b	Q.30	d	Q.31	c	Q.32	a
Q.33	c	Q.34	b	Q.35	c	Q.36	a
Q.37	c	Q.38	d	Q.39	c	Q.40	b
Q.41	b	Q.42	b	Q.43	c		

Explanations of Multiple Choice Questions :

Q.2 Explanation : The data fields in the class are known as instance variables or member variables because they are created when the objects get instantiated.

Q.5 Explanation : It is not necessary that every class should contain the main method.

Q.6 Explanation : The two variables with same name can not be created in Java class.

Q.8 Explanation : For example - consider that the class name is Test, then the object **obj** can be created as follows -

```
Test obj;
Obj=new Test();
```

Q.9 Explanation : The area of circle is computed and its area is printed.

Q.11 Explanation : When the obj2 is assigned with the address of obj1 by the assignment statement obj2=obj1, then all the elements of obj1 are copied to obj2.

Q.15 Explanation : This is one of the property of constructor.

Q.16 Explanation : The simple interest is calculated using the formula (p*n*r)/100 = (100000*5*7)/100

Q.23 Explanation : The constructor does not have any return type.

Q.30 Explanation : Consider a scenario that the garbage collector is called for getting some free space and almost all the nodes are accessible by external pointers. Now Garbage collection routine executes and returns a small amount of space. Then after some time system demands for some free space. This happens repeatedly and garbage collection routine is executing all the time. This process is called thrashing.

Q.34 Explanation : The display function having two parameters will be executed and a = 10
Hence x = a + 2 = 12