CS405PC: JAVA PROGRAMMING

B.TECH II Year II Sem. **L T P C**
3 1 0 4

Course Objectives:

- To introduce the object oriented programming concepts.
- To understand object oriented programming concepts, and apply them in solving problems.
- To introduce the principles of inheritance and polymorphism; and demonstrate how they relate to the design of abstract classes
- To introduce the implementation of packages and interfaces
- To introduce the concepts of exception handling and multithreading.
- To introduce the design of Graphical User Interface using applets and swing controls.

Course Outcomes:

- Able to solve real world problems using OOP techniques.
- Able to understand the use of abstract classes.
- Able to solve problems using java collection framework and I/o
- classes. Able to develop multithreaded applications with synchronization.
- Able to develop applets for web applications.
- Able to design GUI based applications

UNIT - I

Object-Oriented Thinking- A way of viewing world – Agents and Communities, messages and methods, Responsibilities, Classes and Instances, Class Hierarchies- Inheritance, Method binding, Overriding and Exceptions, Summary of Object-Oriented concepts. Java buzzwords, An Overview of Java, Data types, Variables and Arrays, operators, expressions, control statements, Introducing classes, Methods and Classes, String handling.

Inheritance– Inheritance concept, Inheritance basics, Member access, Constructors, Creating Multilevel hierarchy, super uses, using final with inheritance, Polymorphism-ad hoc polymorphism, pure polymorphism, method overriding, abstract classes, Object class, forms of inheritance- specialization, specification, construction, extension, limitation, combination, benefits of inheritance, costs of inheritance.

UNIT - II

Packages- Defining a Package, CLASSPATH, Access protection, importing packages.

Interfaces- defining an interface, implementing interfaces, Nested interfaces, applying interfaces, variables in interfaces and extending interfaces.

Stream based I/O (java.io) – The Stream classes-Byte streams and Character streams, Reading console Input and Writing Console Output, File class, Reading and writing Files, Random access file operations, The Console class, Serialization, Enumerations, auto boxing, generics.

UNIT - III

Exception handling - Fundamentals of exception handling, Exception types, Termination or resumptive models, Uncaught exceptions, using try and catch, multiple catch clauses, nested try statements, throw, throws and finally, built- in exceptions, creating own exception sub classes.

Multithreading- Differences between thread-based multitasking and process-based multitasking, Java thread model, creating threads, thread priorities, synchronizing threads, inter thread communication.

UNIT - IV

The Collections Framework (java.util)- Collections overview, Collection Interfaces, The Collection classes- Array List, Linked List, Hash Set, Tree Set, Priority Queue, Array Deque. Accessing a Collection via an Iterator, Using an Iterator, The For-Each alternative, Map Interfaces and Classes, Comparators, Collection algorithms, Arrays, The Legacy Classes and Interfaces- Dictionary, Hashtable ,Properties, Stack, Vector

More Utility classes, String Tokenizer, Bit Set, Date, Calendar, Random, Formatter, Scanner

UNIT - V

GUI Programming with Swing – Introduction, limitations of AWT, MVC architecture, components, containers. Understanding Layout Managers, Flow Layout, Border Layout, Grid Layout, Card Layout, Grid Bag Layout.

Event Handling- The Delegation event model- Events, Event sources, Event Listeners, Event classes, Handling mouse and keyboard events, Adapter classes, Inner classes, Anonymous Inner classes.

A Simple Swing Application, Applets – Applets and HTML, Security Issues, Applets and Applications, passing parameters to applets. Creating a Swing Applet, Painting in Swing, A Paint example, Exploring Swing Controls- JLabel and Image Icon, JText Field, **The Swing Buttons**- JButton, JToggle Button, JCheck Box, JRadio Button, JTabbed Pane, JScroll Pane, JList, JCombo Box, Swing Menus, Dialogs.

TEXT BOOKS:

1. Java The complete reference, 9th edition, Herbert Schildt, McGraw Hill Education (India) Pvt. Ltd.
2. Understanding Object-Oriented Programming with Java, updated edition, T. Budd, Pearson Education.

REFERENCE BOOKS:

1. An Introduction to programming and OO design using Java, J. Nino and F.A. Hosch, John Wiley & sons
2. Introduction to Java programming, Y. Daniel Liang, Pearson Education.
3. Object Oriented Programming through Java, P. Radha Krishna, University Press.
4. Programming in Java, S. Malhotra, S. Chudhary, 2nd edition, Oxford Univ. Press.
5. Java Programming and Object-oriented Application Development, R. A. Johnson, Cengage Learning.

UNIT I:

Object-Oriented Thinking- A way of viewing world – Agents and Communities, messages and methods, Responsibilities, Classes and Instances, Class Hierarchies- Inheritance, Method binding, Overriding and Exceptions, Summary of Object-Oriented concepts. Java buzzwords, An Overview of Java, Data types, Variables and Arrays, operators, expressions, control statements, Introducing classes, Methods and Classes, String handling.

Inheritance– Inheritance concept, Inheritance basics, Member access, Constructors, Creating Multilevel hierarchy, super uses, using final with inheritance, Polymorphism-ad hoc polymorphism, pure polymorphism, method overriding, abstract classes, Object class, forms of inheritance- specialization, specification, construction, extension, limitation, combination, benefits of inheritance, costs of inheritance.

Object-Oriented Thinking

A way of viewing world:

To illustrate some of the major ideas in object-oriented programming, let us consider rst how we might go about handling a real-world situation and then ask how we could make the computer more closely model the techniques employed. Suppose an individual named Chris wishes to send owers to a friend named Robin, who lives in another city. Because of the distance, Chris cannot simply pick the owers and take them to Robin in person. Nevertheless, it is a task that is easily solved. Chris simply walks to a nearby ower shop, run by a florist named Fred. Chris will tell Fred the kinds of owers to send to Robin, and the address to which they should be delivered. Chris can then be assured that the flowers will be delivered expediently and automatically.

Agents and Communities

At the risk of belaboring a point, let us emphasize that the mechanism that was used to solve this problem was to nd an appropriate agent (namely, Fred) and to pass to this agent a message containing a request. It is the responsibility of Fred to satisfy the request. There is some method - some algorithm or set of operations - used by Fred to do this. Chris does not need to know the particular method that Fred will use to satisfy the request; indeed, often the person making a request does not want to know the details. This information is usually hidden from inspection.

An investigation, however, might uncover the fact that Fred delivers a slightly different message to another orist in the city where Robin lives. That orist, in turn, perhaps has a subordinate who makes the ower arrangement. The orist then passes the owers, along with yet another message, to a delivery person, and so on. Earlier, the orist in Robin's city had obtained her owers from a flower wholesaler who, in turn, had interactions with the ower growers, each of whom had to manage a team of gardeners.

So, our rst observation of object-oriented problem solving is that the solution to this problem required the help of many other individuals (Figure 1.2). Without their help, the problem could not be easily solved. We phrase this in a general fashion as the following:

An object oriented program is structured as a community of interacting agents, called objects. Each object has a role to play. Each object provides a service, or performs an action, that is used by other members of the community.

Messages and Methods

The chain reaction that ultimately resulted in the solution to Chris's problem began with a request given to the orist Fred. This request lead to other requests, which lead to still more requests, until the owers ultimately reached Chris's friend Robin. We see, therefore, that members of this community interact with each other by making requests. So, our next principle of object-oriented problem solving is the vehicle used to indicate an action to be performed:

Action is initiated in object-oriented programming by the transmission of a message to an agent (an object) responsible for the action. The message encodes the request for an action and is accompanied by any additional information (arguments) needed to carry out the request. The receiver is the object to whom the message is sent. If the receiver accepts the message, it accepts the responsibility to carry out the indicated action. In response to a message, the receiver will perform some method to satisfy the request.

We have noted the important principle of information hiding in regard to message passing - that is, the client sending the request need not know the actual means by which the request will be honored. There is another principle, all too human, that we see is implicit in message passing. If there is a task to perform, the first thought of the client is to find somebody else he or she can ask to do the work. This second reaction often becomes atrophied in many programmers with extensive experience in conventional techniques. Frequently, a di cult hurdle to overcome is the idea in the programmer's mind that he or she must write everything and not use the services of others. An important part of object-oriented programming is the development of reusable components, and an important rst step in the use of reusable components is a willingness to trust software written by others.

Messages versus Procedure Calls

Information hiding is also an important aspect of programming in conventional languages. In what sense is a message di erent from, say, a procedure call? In both cases, there is a set of well-defined steps that will be initiated following the request. But, there are two important distinctions.

The rst is that in a message there is a designated receiver for that message; the receiver is some object to which the message is sent. In a procedure call, there is no designated receiver.

The second is that the interpretation of the message (that is, the method used to respond to the message) is determined by the receiver and can vary with di erent receivers. Chris could give a message to a friend named Elizabeth, for example, and she will understand it and a satisfactory outcome will be produced (that is, owers will be delivered to their mutual friend Robin). However, the method Elizabeth uses to satisfy the request (in all likelihood, simply passing the request on to Fred) will be di erent from that used by Fred in response to the same request.

If Chris were to ask Kenneth, a dentist, to send owers to Robin, Kenneth may not have a method for solving that problem. If he understands the request at all, he will probably issue an appropriate error diagnostic.

Let us move our discussion back to the level of computers and programs. There, the distinction between message passing and procedure calling is that, in message passing, there is a designated receiver, and the interpretation - the selection of a method to execute in response to the message - may vary with different receivers. Usually, the speci c receiver for any given message will not be known until run time, so the determination of which method to invoke cannot be made until then. Thus, we say there is late binding between the message (function or procedure name) and the code fragment (method) used to respond to the message. This situation is in contrast to the very early (compile-time or link-time) binding of name to code fragment in conventional procedure calls.

Responsibilities

A fundamental concept in object-oriented programming is to describe behavior in terms of responsibilities. Chris's request for action indicates only the desired out- come (flowers sent to Robin). Fred is free to pursue any technique that achieves the desired objective, and in doing so will not be hampered by interference from Chris.

By discussing a problem in terms of responsibilities we increase the level of abstraction. This permits greater independence between objects, a critical factor in solving complex problems. The entire collection of responsibilities associated with an object is often described by the term protocol.

A traditional program often operates by acting on data structures, for exam- ple changing elds in an array or record. In contrast, an object oriented program requests data structures (that is, objects) to perform a service. This di erence between viewing software in traditional, structured terms and viewing it from an object-oriented perspective can be summarized by a twist on a well-known quote:

> Ask not what you can do to your data structures, but rather ask what your data structures can do for you.

Classes and Instances

Although Chris has only dealt with Fred a few times, Chris has a rough idea of the transaction that will occur inside Fred's ower shop. Chris is able to make certain assumptions based on previous experience with other orists, and hence Chris can expect that Fred, being an instance of this category, will fit the general pattern. We can use the term Florist to represent the category (or class) of all orists. Let us incorporate these notions into our next principle of object-oriented programming:

> All objects are instances of a class. The method invoked by an object in response to a message is determined by the class of the receiver. All objects of a given class use the same method in response to similar messages.

Class Hierarchies - Inheritance

Chris has more information about Fred - not necessarily because Fred is a florist but because he is a shopkeeper. Chris knows, for example, that a transfer of money will be part of the transaction, and that in return for payment Fred will o er a receipt. These actions are true of grocers, stationers, and other shopkeepers. Since the category Florist is a more specialized form of the category Shopkeeper, any knowledge Chris has of Shopkeepers is also true of Florists and hence of Fred.

One way to think about how Chris has organized knowledge of Fred is in terms of a hierarchy of categories. Fred is a Florist, but Florist is a specialized form of Shopkeeper. Furthermore, a Shopkeeper is also a Human; so Chris knows, for example, that Fred is probably bipedal. A Human is a Mammal (therefore they nurse their young and have hair), and a Mammal is an Animal (therefore it breathes oxygen), and an Animal is a Material Object (therefore it has mass and weight). Thus, quite a lot of knowledge that Chris has that is applicable to Fred is not directly associated with him, or even with the category Florist.

The principle that knowledge of a more general category is also applicable to a more speci c category is called inheritance. We say that the class Florist will inherit attributes of the class (or category) Shopkeeper.

There is an alternative graphical technique often used to illustrate this rela- tionship, particularly when there are many individuals with di ering lineage's. This technique shows classes listed in a hierarchical tree-like structure, with more abstract classes (such as Material Object or Animal) listed near the top of the tree, and more speci c classes, and nally individuals, are listed near the bottom. This same hierarchy also includes Elizabeth, Chris's dog Fido, Phyl the platypus who lives at the zoo, and the owers the Chris is sending to Robin. Notice that the structure and interpretation of this type of diagram is similar to the biological hierarchy.

Information that Chris possess about Fred because Fred is an instance of class Human is also applicable to Elizabeth, for example. Information that Chris knows about Fred because he is a Mammal is applicable to Fido as well. Information about all members of Material Object is equally applicable to Fred and to his owers. We capture this in the idea of inheritance:

> Classes can be organized into a hierarchical inheritance structure. A child class (or subclass) will inherit attributes from a parent class higher in the tree. An abstract parent class is a class (such as Mam-mal) for which there are no direct instances; it is used only to create subclasses.

Method Binding and Overriding

Phyl the platypus presents a problem for our simple organizing structure. Chris knows that mammals give birth to live children, and Phyl is certainly a Mammal, yet Phyl (or rather his mate Phyllis) lays eggs. To accommodate this, we need to nd a technique to encode exceptions to a general rule.

We do this by decreeing that information contained in a subclass can override information inherited from a parent class. Most often, implementations of this approach takes the form of a method in a subclass having the same name as a method in the parent class, combined with a rule for how the search for a method to match a specific message is conducted:

> he search for a method to invoke in response to a given message be- gins with the class of the receiver. If no appropriate method is found, the search is conducted in the parent class of this class. The search continues up the parent class chain until either a method is found or the parent class chain is exhausted. In the former case the method is executed; in the latter case, an error message is issued. If methods with the same name can be found higher in the class hierarchy, the method executed is said to override the inherited behavior.

Even if a compiler cannot determine which method will be invoked at run time, in many object-oriented languages, such as Java, it can determine whether there will be an appropriate method and issue an error message as a compile-time error diagnostic rather than as a run-time message.

The fact that both Elizabeth and Fred will react to Chris's messages, but use different methods to respond, is one form of polymorphism. As explained, that Chris does not, and need not, know exactly what method Fred will use to honor the request is an example of information hiding.

Summary of Object-Oriented Concepts

Alan Kay, considered by some to be the father of object-oriented programming, identi ed the following characteristics as fundamental to OOP [Kay 1993]:

1. Everything is an object.
2. Computation is performed by objects communicating with each other, requesting that other objects perform actions. Objects communicate by sending and receiving messages. A message is a request for action bundled with whatever arguments may be necessary to complete the task.
3. Each object has its own memory, which consists of other objects.
4. Every object is an instance of a class. A class simply represents a grouping of similar objects, such as integers or lists.
5. The class is the repository for behavior associated with an object. That is, all objects that are instances of the same class can perform the same actions.
6. Classes are organized into a singly rooted tree structure, called the inheritance hierarchy. Memory and behavior associated with instances of a class are automatically available to any class associated with a descendant in this tree structure.

History of Java:

In 1990, Sun Micro Systems Inc (US) was conceived a project to develop software for consumer electronic devices that could be controlled by a remote This project was called Stealth Project but later its name was changed to Green Project

In January 1991, Project Manager James Gosling and his team members Patrick Naughton, Mike Sheridan, Chris Wrath, and Ed Frank met to discuss about this project

Gosling thought C and C++ would be used to develop the project But the problem he faced with them is that they were system dependent languages The trouble with C and C++ (and most other languages) is that they are designed to be compiled for a specific target and could not be used on various processors, which the electronic devices might use

James Gosling with his team started developing a new language, which was completely system independent This language was initially called **OAK** Since this name was registered by some other company, later it was changed to **Java**

James Gosling and his team members were consuming a lot of coffee while developing this language Good quality of coffee was supplied from a place called "Java Island‟ Hence they fixed the name of the language as Java The symbol for Java language is cup and saucer

Sun formally announced Java at Sun World conference in 1995 On January 23rd 1996, JDK10 version was released

Features of Java (Java buzz words):

Simple: Learning and practicing java is easy because of resemblance with c and C++
Object Oriented Programming Language: Unlike C++, Java is purely OOP
Distributed: Java is designed for use on network; it has an extensive library which works in agreement with TCP/IP

Secure: Java is designed for use on Internet Java enables the construction of virus-free, tamper free systems

Robust (Strong/ Powerful): Java programs will not crash because of its exception handling and its memory management features

Interpreted: Java programs are compiled to generate the byte code This byte code can be downloaded and interpreted by the interpreter class file will have byte code instructions and JVM which contains an interpreter will execute the byte code

Portable: Java does not have implementation dependent aspects and it yields or gives same result on any machine

Architectural Neutral Language: Java byte code is not machine dependent, it can run on any machine with any processor and with any OS

High Performance: Along with interpreter there will be JIT (Just In Time) compiler which enhances the speed of execution

Multithreaded: Executing different parts of program simultaneously is called multithreading This is an essential feature to design server side programs

Dynamic: We can develop programs in Java which dynamically change on Internet (eg: Applets)

Obtaining the Java Environment:

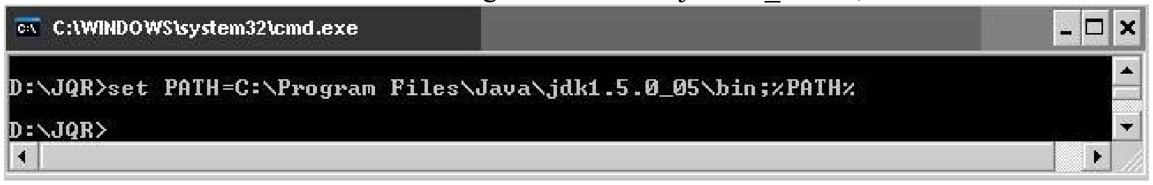
We can download the JDK (Java Development Kit) including the compiler and runtime engine from Sun at: http://javasuncom/javase

Install JDK after downloading, by default JDK will be installed in
C:\Program Files\Java\jdk150_05 (Here jdk150_05 is JDK‟s version)

Setting up Java Environment: After installing the JDK, we need to set at least one environment variable in order to able to compile and run Java programs A PATH environment variable enables the operating system to find the JDK executables when our working directory is not the JDK's binary directory

Setting environment variables from a command prompt: If we set the variables from a command prompt, they will only hold for that session To set the PATH from a command prompt:

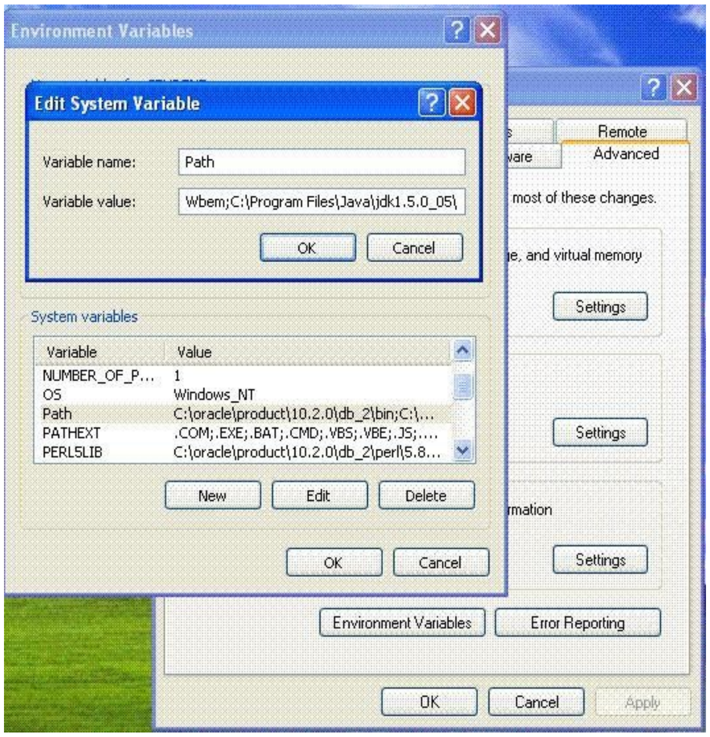
set PATH=C:\Program Files\Java\jdk150_05\bin;%PATH%



```
D:\JQR>set PATH=C:\Program Files\Java\jdk1.5.0_05\bin;%PATH%

D:\JQR>
```

Setting environment variables as system variables: If we set the variables as system variables they will hold continuously

- o Right-click on *My Computer*
- o Choose *Properties*
- o Select the *Advanced* tab
- o Click the *Environment Variables* button at the bottom
- o In system variables tab, select path (system variable) and click on edit button
- o A window with variable name-path and its value will be displayed
- o Don"t disturb the default path value that is appearing and just append (add) to that path at the end: ;C:\ProgramFiles\Java\ jdk150_05\bin;
  - o Finally press OK button



Variable name: Path
Variable value: Wbem;C:\Program Files\Java\jdk1.5.0_05\

Variable	Value
NUMBER_OF_P...	1
OS	Windows_NT
Path	C:\oracle\product\10.2.0\db_2\bin;C:\...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....
PERL5LIB	C:\oracle\product\10.2.0\db_2\perl\5.8...

Programming Structure

Comments: Comments are description about the aim and features of the program Comments increase readability of a program Three types of comments are there in Java:

Single line comments: These comments start with //
eg: // this is comment line
Multi line comments: These comments start with /* and end with */
eg: /* this is comment line*/
Java documentation comments: These comments start with /** and end with */
These comments are useful to create a HTML file called API (application programming Interface) document This file contains description of all the features of software

Structure of the Java Program:

As all other programming languages, Java also has a structure

The first line of the C/C++ program contains include statement For example, <stdioh> is the header file that contains functions, like printf (), scanf () etc So if we want to use any of these functions, we should include this header file in C/ C++ program

Similarly in Java first we need to import the required packages By default javalang* is imported Java has several such packages in its library A package is a kind of directory that contains a group of related classes and interfaces A class or interface contains methods Since Java is purely an Object Oriented Programming language, we cannot write a Java program without having at least one class or object So, it is mandatory to write a class in Java program We should use class keyword for this purpose and then write class name

In C/C++, program starts executing from main method similarly in Java, program starts executing from main method The return type of main method is void because program starts executing from main method and it returns nothing

Sample Program:

```
//A Simple Java Program
import javalangSystem; import
javalangString; class Sample
{
            public static void main(String args[])
                        {
                                    System.out.print ("Hello world");
                        }
}
```

Since Java is purely an Object Oriented Programming language, without creating an object to a class it is not possible to access methods and members of a class But main method is also a method inside a class, since program execution starts from main method we need to call main method without creating an object

Static methods are the methods, which can be called and executed without creating objects Since we want to call main () method without using an object, we should declare main () method as static JVM calls main () method using its Classnamemain () at the time of running the program.JVM is a program written by Java Soft people (Java development team) and main () is the method written by us Since, main () method should be available to the JVM, it should be declared as public If we don"t declare main () method as public, then it doesn"t make itself available to JVM and JVM cannot execute it

JVM always looks for main () method with String type array as parameter otherwise JVM cannot recognize the main () method, so we must provide String type array as parameter to main () method

A class code starts with a {and ends with a} A class or an object contains variables and methods (functions) We can create any number of variables and methods inside the class This is our first program, so we had written only one method called main ()

Our aim of writing this program is just to display a string "Hello world" In Java, print () method is used to display something on the monitor

A method should be called by using objectnamemethodname () So, to call print () method, create an object to PrintStream class then call objectnameprint () method

An alternative is given to create an object to PrintStream Class ie Systemout Here, System is the class name and out is a static variable in System class out is called a field in System class When we call this field a PrintStream class object will be created internally So, we can call print() method as: System.out.print ("Hello world"); println () is also a method belonging to PrintStream class It throws the cursor to the next line after displaying the result

In the above Sample program System and String are the classes present in javalang package

Escape Sequence: Java supports all escape sequence which is supported by C/ C++ A character preceded by a backslash (\) is an escape sequence and has special meaning to the compiler When an escape sequence is encountered in a print statement, the compiler interprets it accordingly

Escape Sequence	Description
\t	Insert a tab in the text at this point
\b	Insert a backspace in the text at this point
\n	Insert a newline in the text at this point
\r	Insert a carriage return in the text at this point
\f	Insert a form feed in the text at this point
\'	Insert a single quote character in the text at this point
\"	Insert a double quote character in the text at this point
\\	Insert a backslash character in the text at this point

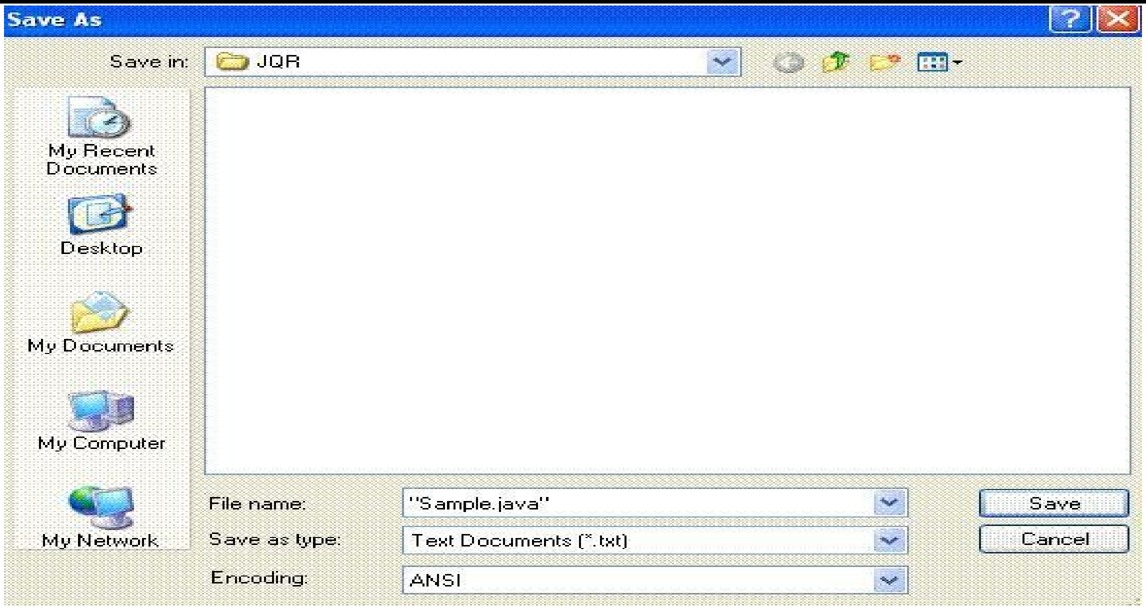
Creating a Source File:

Type the program in a text editor (ie Notepad, WordPad, Microsoft Word or Edit Plus) We can launch the Notepad editor from the **Start** menu by selecting **Programs > Accessories > Notepad** In a new document, type the above code (ie Sample Program)

Save the program with filename same as Class_name (ie Samplejava) in which main method is written To do this in Notepad, first choose the **File > Save** menu item Then, in the **Save** dialog box:

Using the **Save in** combo box, specify the folder (directory) where you'll save your file

In the **File name** text field, type "Samplejava", including the quotation marks Then the dialog box should look like this:



Save in: JQR

File name: "Sample.java"

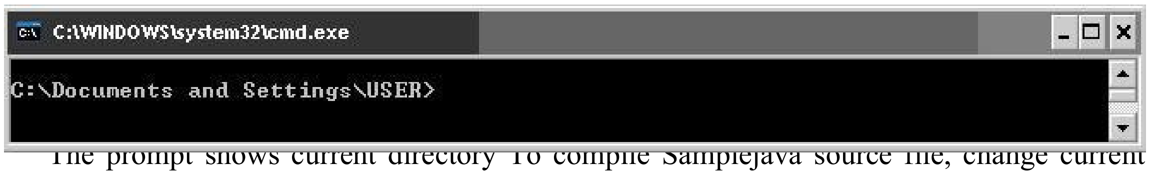
Save as type: Text Documents (*.txt)

Encoding: ANSI

Now click **Save**, and exit Notepad

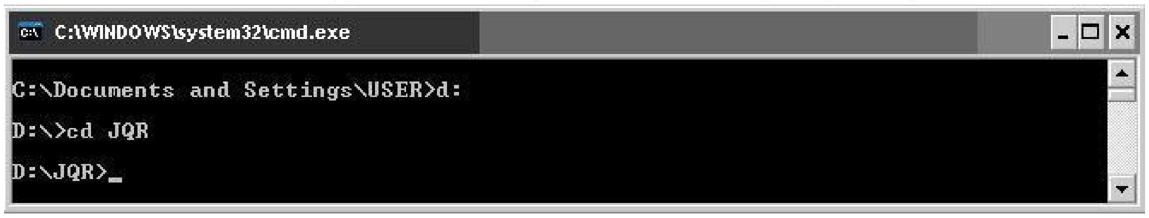
Compiling the Source File into a class File:

To Compile the Samplejava program go to DOS prompt We can do this from the **Start** menu by choosing **Run** and then entering cmd The window should look similar to the following figure



```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\USER>
```

The prompt shows current directory To compile Samplejava source file, change current directory to the directory where Samplejava file is located For example, if source directory is JQR on the D drive, type the following commands at the prompt and press Enter:

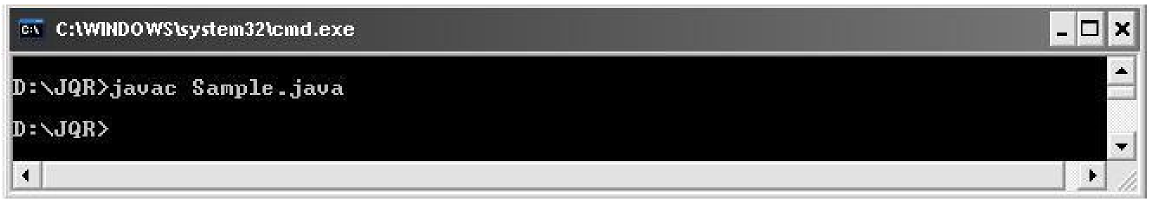


```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\USER>d:
D:\>cd JQR
D:\JQR>_
```

Now the prompt should change to D:\JQR>

At the prompt, type the following command and press Enter

javac Samplejava



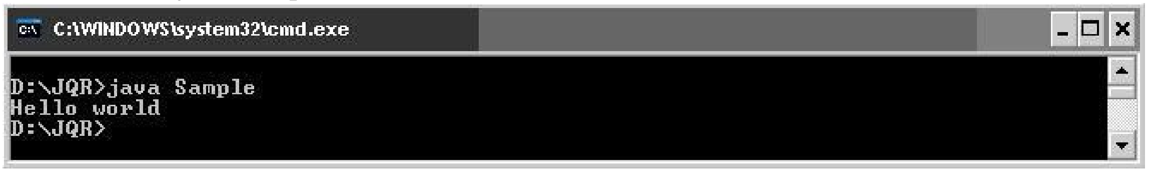
```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Sample.java

D:\JQR>
```

The compiler generates byte code and Sampleclass will be created

Executing the Program (Sampleclass):

To run the program, enter java followed by the class name created at the time of compilation at the command prompt in the same directory as:

java Sample



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>java Sample
Hello world
D:\JQR>
```

The program interpreted and the output is displayed

The Java Virtual Machine: Java Virtual Machine (JVM) is the heart of entire Java program execution process First of all, the java program is converted into a class file consisting of byte code instructions by the java compiler at the time of compilation Remember, this java compiler is outside the JVM This class file is given to the JVM Following figure shows the architecture of Java Virtual Machine

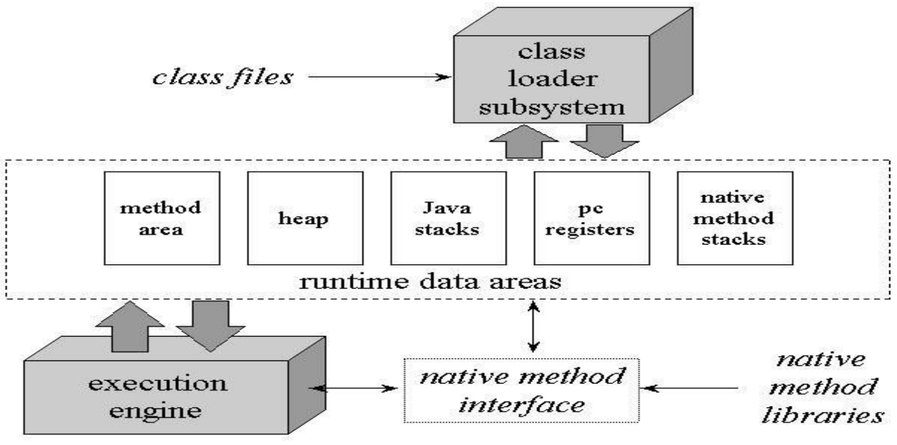


Figure: The internal architecture of the Java virtual machine

In JVM, there is a module (or program) called class loader sub system, which performs the following instructions:

First of all, it loads the class file into memory

Then it verifies whether all byte code instructions are proper or not If it finds any instruction suspicious, the execution is rejected immediately

.If the byte instructions are proper, then it allocates necessary memory to execute the program This memory is divided into 5 parts, called run time data areas, which contain the data and results while running the program These areas are as follows:

- o **Method area:** Method area is the memory block, which stores the class code, code of the variables and code of the methods in the Java program (Method means functions written in a class)
- o **Heap:** This is the area where objects are created Whenever JVM loads a class, method and heap areas are immediately created in it
- o **Java Stacks:** Method code is stored on Method area But while running a method, it needs some more memory to store the data and results This memory is allotted on Java Stacks So, Java Stacks are memory area where Java methods are executed While executing methods, a separate frame will be created in the Java Stack, where the method is executed JVM uses a separate thread (or process) to execute each method
- o **PC (Program Counter) registers:** These are the registers (memory areas), which contain memory address of the instructions of the methods If there are 3 methods, 3 PC registers will be used to track the instruction of the methods
- o **Native Method Stacks:** Java methods are executed on Java Stacks Similarly, native methods (for example C/C++ functions) are executed on Native method stacks To execute the native methods, generally native method libraries (for example C/C++ header files) are required These header files are located and connected to JVM by a program, called Native method interface

Execution Engine contains interpreter and JIT compiler which translates the byte code instructions into machine language which are executed by the microprocessor Hot spot (loops/iterations) is the area in class file ie executed by JIT compiler JVM will identify the Hot spots in the class files and it will give it to JIT compiler where the normal instructions and statements of Java program are executed by the Java interpreter

Naming Conventions, Data Types and Operators

Naming Conventions: Naming conventions specify the rules to be followed by a Java programmer while writing the names of packages, classes, methods etc

Package names are written in small letters **eg**: javaio, javalang, javaawt etc

Each word of class name and interface name starts with a capital **eg**: Sample, AddTwoNumbers

Method names start with small letters then each word start with a capital **eg**: sum (), sumTwoNumbers (), minValue ()

Variable names also follow the same above method rule **eg**: sum, count, totalCount

Constants should be written using all capital letters **eg**: PI, COUNT

Keywords are reserved words and are written in small letters **eg**: int, short, float, public, void

Data Types: The classification of data item is called data type Java defines eight simple types of data byte, short, int, long, char, float, double and boolean These can be put in four groups:

Integer Data Types: These data types store integer numbers

Data Type	Memory size	Range
Byte	1 byte	-128 to 127
Short	2 bytes	-32768 to 32767
Int	4 bytes	-2147483648 to 2147483647
Long	8 bytes	-9223372036854775808 to 9223372036854775807

eg: byte rno = 10;
long x = 150L; L means forcing JVM to allot 8 bytes

Float Data Types: These data types handle floating point numbers

Data Type	Memory size	Range
Float	4 bytes	-34e38 to 34e38
Double	8 bytes	-17e308 to 17e308

eg: float pi = 3142f;
double distance = 198e8;

Character Data Type: This data type represents a single character char data type in java uses two bytes of memory also called Unicode system Unicode is a specification to include alphabets of all international languages into the character set of java

Data Type	Memory size	Range
Char	2 bytes	0 to 65535

eg: char ch = 'x';

Boolean Data Type:can handle truth values either true or false
eg:- boolean response = true;

Operators: An operator is a symbol that performs an operation An operator acts on variables called operands

Arithmetic operators: These operators are used to perform fundamental operations like addition, subtraction, multiplication etc

Operator	Meaning	Example	Result
+	Addition	3 + 4	7
-	Subtraction	5 - 7	-2
*	Multiplication	5 * 5	25
/	Division (gives quotient)	14 / 7	2
%	Modulus (gives remainder)	20 % 7	6

· **Assignment operator:** This operator (=) is used to store some value into a variable

Simple Assignment	Compound Assignment
x = x + y	x += y
x = x – y	x -= y
x = x * y	x *= y
x = x y	x /= y

.
.

·

·

·

· **Unary operators:** As the name indicates unary operator‟s act only on one operand

Operator	Meaning	Example	Explanation
-	Unary minus	j = -k;	k value is negated and stored into j

++	Increment Operator	b++;
		++b;
			b value will be incremented by 1 (called as post incrementation)
			b value will be incremented by 1 (called as pre incrementation)

--	Decrement Operator	b--; --b;	b value will be decremented by 1 (called as post decrementation) b value will be decremented by 1 (called as pre decrementation)

· **Relational operators:** These operators are used for comparison purpose

Operator	Meaning	Example
==	Equal	x == 3
!=	Not equal	x != 3
<	Less than	x < 3
>	Greater than	x > 3
<=	Less than or equal to	x <= 3

· **Logical operators:** Logical operators are used to construct compound conditions A compound condition is a combination of several simple conditions

Operator	Meaning	Example	Explanation

&&	and operator	if(a>b && a>c)
		System.out.print(“yes”);
			If a value is greater than b and c then only yes is displayed

||	or operator	if(a==1 || b==1)
		System.out.print(“yes”);
			If either a value is 1 or b value is 1 then yes is displayed

!	not operator	if(!(a==0))
		System.out.print(“yes”);
			If a value is not equal to zero then only yes is displayed

.

Bitwise operators: These operators act on individual bits (0 and 1) of the operands They act only on integer data types, ie byte, short, long and int

Operator	Meaning	Explanatio n
&	Bitwise AND	Multiplies the individual bits of operands
|	Bitwise OR	Adds the individual bits of operands
^	Bitwise XOR	Performs Exclusive OR operation
<<	Left shift	Shifts the bits of the number towards left a specified number of positions
>>	Right shift	Shifts the bits of the number towards right a specified number of positions and also preserves the sign bit
>>>	Zero fill right shift	Shifts the bits of the number towards right a specified number of positions and it stores 0 changing 0"s as 1"s and vice versa

Ternary Operator or Conditional Operator (? :): This operator is called ternary because it acts on 3 variables The syntax for this operator is:

Variable = Expression1? Expression2: Expression3;

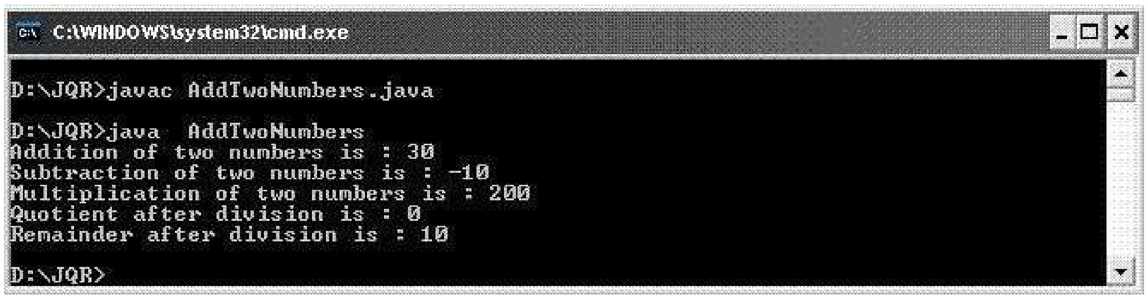
First Expression1 is evaluated If it is true, then Expression2 value is stored into variable otherwise Expression3 value is stored into the variable

eg: max = (a>b) ? a: b;

Program 1: Write a program to perform arithmetic operations

```
//Addition of two numbers
class AddTwoNumbers
{               public static void mian(String args[])
                {int i=10, j=20;
                        System.out.println("Addition of two numbers is : " + (i+j));
                        System.out.println("Subtraction of two numbers is : " + (i-j));
                        System.out.println("Multiplication of two numbers is : " + (i*j));
                        System.out.println("Quotient after division is : " + (i/j) );
                        System.out.println("Remainder after division is : " +(i%j) );
                }
}
```

Output:

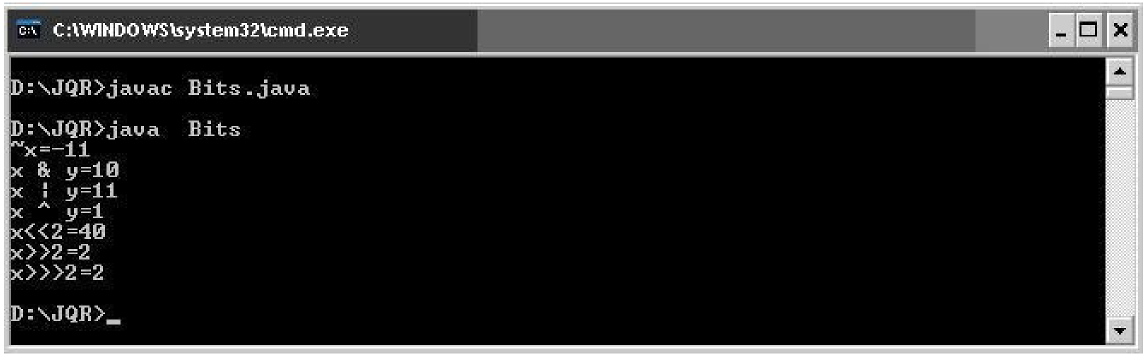


```
D:\JQR>javac AddTwoNumbers.java

D:\JQR>java AddTwoNumbers
Addition of two numbers is : 30
Subtraction of two numbers is : -10
Multiplication of two numbers is : 200
Quotient after division is : 0
Remainder after division is : 10

D:\JQR>
```

Program 2: Write a program to perform Bitwise operations

```
//Bitwise Operations
class Bits
{public static void main(String args[])
	{byte	x,y;
		x=10;
		y=11;
		System.out.println	("~x="+(~x));
		System.out.println ("x & y="+(x&y));
		System.out.println ("x | y="+(x|y));
		System.out.println ("x ^ y="+(x^y));
		System.out.println ("x<<2="+(x<<2));
		System.out.println ("x>>2="+(x>>2));
		System.out.println
		("x>>>2="+(x>>>2));
	}
}
```

Output:



C:\WINDOWS\system32\cmd.exe

```
D:\JQR>javac Bits.java

D:\JQR>java  Bits
~x=-11
x & y=10
x | y=11
x ^ y=1
x<<2=40
x>>2=2
x>>>2=2

D:\JQR>_
```

Control Statements

Control statements are the statements which alter the flow of execution and provide better control to the programmer on the flow of execution In Java control statements are categorized into selection control statements, iteration control statements and jump control statements

Java's Selection Statements: Java supports two selection statements: if and switch These statements allow us to control the flow of program execution based on condition

if Statement: if statement performs a task depending on whether a condition is true or false

Syntax: if (condition)

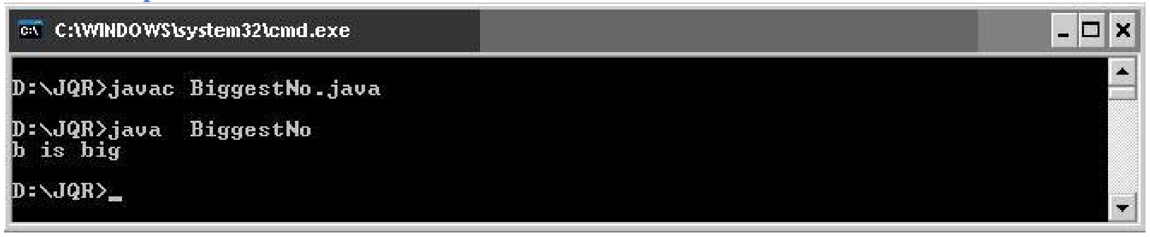
```
                statement1;
        else
                statement2;
```

Here, each statement may be a single statement or a compound statement enclosed in curly braces (that is, a block) The condition is any expression that returns a boolean value The else clause is optional

Program 1: Write a program to find biggest of three numbers

```
//Biggest of three numbers
class BiggestNo
{public static void main(String args[])
        {int a=5,b=7,c=6; if ( a
                > b && a>c)
                        System.out.println ("a is big");
                else if ( b > c)
                        System.out.println ("b is big");
else
                        System.out.println ("c is big");
        }
}
```

Output:



```
D:\JQR>javac BiggestNo.java

D:\JQR>java  BiggestNo
b is big

D:\JQR>_
```

Switch Statement: When there are several options and we have to choose only one option from the available ones, we can use switch statement

Syntax:switch (expression)

```
        {case  value1:  //statement  sequence
                break;
        case value2:   //statement sequence
```

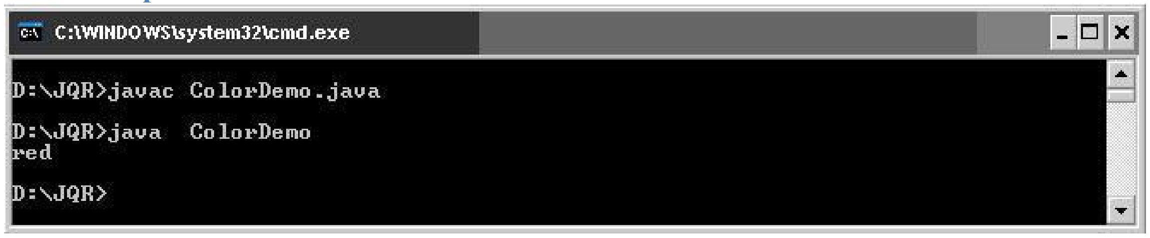
```
                break;
                ……………
                case valueN: //statement sequence
                             break;
                default:     //default statement sequence
        }
```

Here, depending on the value of the expression, a particular corresponding case will be executed

Program 2: Write a program for using the switch statement to execute a particular task depending on color value

```
//To display a color name depending on color
value class ColorDemo
{public static void main(String args[])
     {char color = „r‟;
          switch (color)
          {    case „r‟: System.out.println ("red");       break;
               case „g‟: System.out.println ("green");     break;
               case „b‟: System.out.println ("blue");      break;
               case „y‟: System.out.println ("yellow");    break;
               case „w‟: System.out.println ("white");     break;
               default:  System.out.println ("No Color
               Selected");
          }
     }
}
```

Output:



```
D:\JQR>javac ColorDemo.java

D:\JQR>java  ColorDemo
red

D:\JQR>
```

Java's Iteration Statements: Java‟s iteration statements are for, while and do-while These statements are used to repeat same set of instructions specified number of times called loops A loop repeatedly executes the same set of instructions until a termination condition is met

while Loop: while loop repeats a group of statements as long as condition is true Once the condition is false, the loop is terminated In while loop, the condition is tested first; if it is true, then only the statements are executed while loop is called as entry control loop

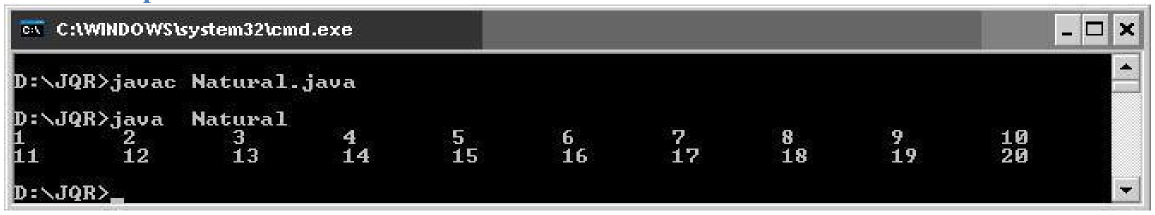
Syntax:while (condition)

```
          {
               statements;
          }
```

Program 3: Write a program to generate numbers from 1 to 20

```
//Program to generate numbers from 1 to 20
class Natural
{public static void main(String args[])
        {int i=1; while (i <=20)
                {System.out.print (i + "\t"); i++;
                }
        }
}
```

Output:



C:\WINDOWS\system32\cmd.exe

```
D:\JQR>javac Natural.java

D:\JQR>java  Natural
1       2       3       4       5       6       7       8       9       10
11      12      13      14      15      16      17      18      19      20

D:\JQR>_
```

do…while Loop: do…while loop repeats a group of statements as long as condition is true In dowhile loop, the statements are executed first and then the condition is tested do…while loop is also called as exit control loop

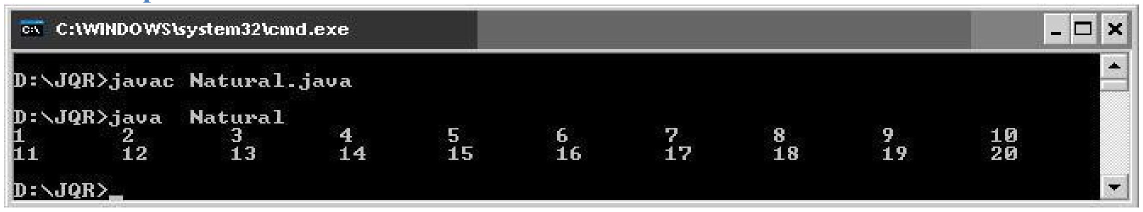
Syntax:do

```
                {
                        statements;
                } while (condition);
```

Program 4: Write a program to generate numbers from 1 to 20

```
//Program to generate numbers from 1 to 20
class Natural
{public static void main(String args[])
        {int i=1; do
                {System.out.print (i + "\t"); i++;
                } while (i <= 20);
        }
}
```

Output:



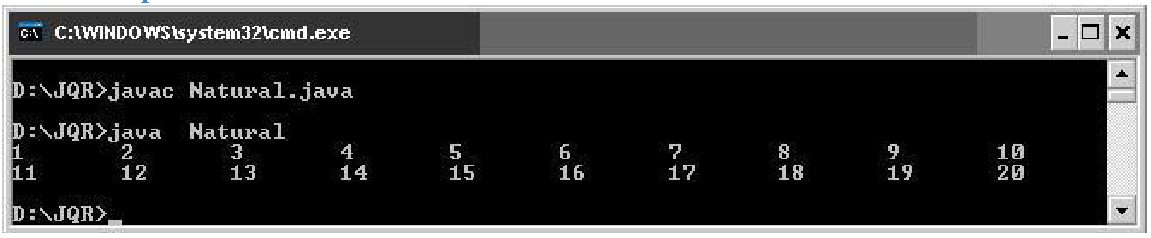
C:\WINDOWS\system32\cmd.exe

```
D:\JQR>javac Natural.java

D:\JQR>java  Natural
1       2       3       4       5       6       7       8       9       10
11      12      13      14      15      16      17      18      19      20

D:\JQR>_
```

. **for Loop:** The for loop is also same as do…while or while loop, but it is more compact syntactically The for loop executes a group of statements as long as a condition is true

Syntax:
```
for (expression1; expression2; expression3)
{               statements;
}
```
Here, expression1 is used to initialize the variables, expression2 is used for condition checking and expression3 is used for increment or decrement variable value

Program 5: Write a program to generate numbers from 1 to 20
```
//Program to generate numbers from 1 to 20
class Natural
{public static void main(String args[])
        {int i;
                for (i=1; i<=20; i++)
                        System.out.print (i + "\t");
        }
}
```
Output:



```
D:\JQR>javac Natural.java

D:\JQR>java  Natural
1       2       3       4       5       6       7       8       9       10
11      12      13      14      15      16      17      18      19      20

D:\JQR>_
```

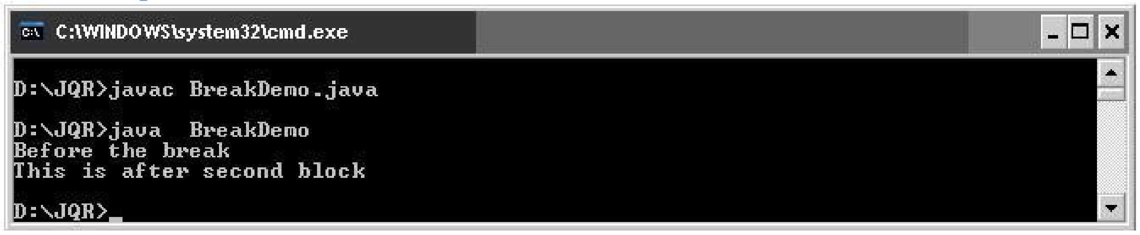
Java's Jump Statements: Java supports three jump statements: break, continue and return These statements transfer control to another part of the program

o **break:**
  break can be used inside a loop to come out of it
  break can be used inside the switch block to come out of the switch block
  break can be used in nested blocks to go to the end of a block Nested blocks represent a block written within another block

Syntax: break; (or) break label;//here label represents the name of the block

Program 6: Write a program to use break as a civilized form of goto
```
//using break as a civilized form of goto
class BreakDemo
{public static void main (String args[])
        {boolean t = true; first:
                {
                        second:
                        {
                                third:
                                {
                                        System.out.println ("Before the break");
                                        if (t) break second; // break out of second block
                                              System.out.println ("This won"t execute");
                                   }
                                   System.out.println ("This won"t execute");
                              }
                              System.out.println ("This is after second block");
                         }
                    }
               }
```

Output:



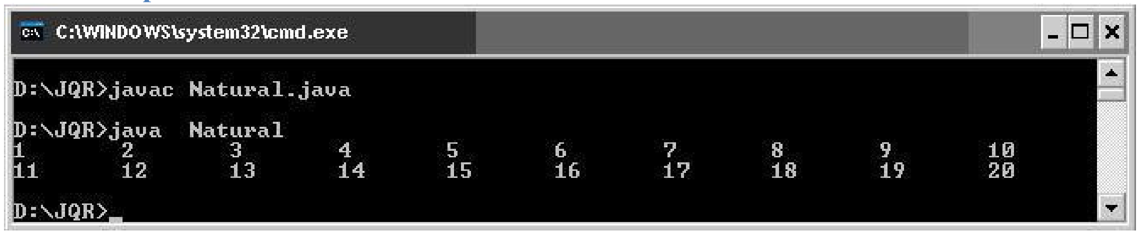
```
D:\JQR>javac BreakDemo.java

D:\JQR>java  BreakDemo
Before the break
This is after second block

D:\JQR>_
```

continue: This statement is useful to continue the next repetition of a loop/ iteration When continue is executed, subsequent statements inside the loop are not executed **Syntax:**continue;

Program 7: Write a program to generate numbers from 1 to 20

```
               //Program to generate numbers from 1 to 20
               class Natural
               {public static void main (String args[])
                    {int   i=1;   while
                         (true)
                         {System.out.print  (i  +  "\t");
                              i++; if (i <= 20 )
                                   continue
                                   ;
else
                                   break;
                    }
     }
}
```

Output:



```
D:\JQR>javac Natural.java

D:\JQR>java  Natural
1       2       3       4       5       6       7       8       9       10
11      12      13      14      15      16      17      18      19      20

D:\JQR>_
```

return statement:

return statement is useful to terminate a method and come back to the calling method

return statement in main method terminates the application

return statement can be used to return some value from a method to a calling method

Syntax:return;

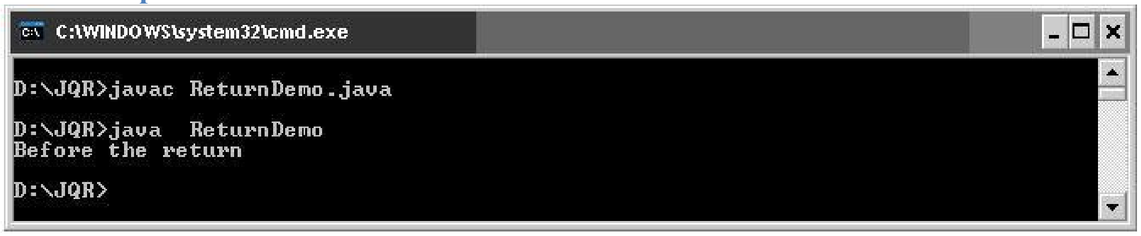
(or)

return value; // value may be of any type

Program 8: Write a program to demonstrate return statement

```
//Demonstrate return
class ReturnDemo
{public static void main(String args[])
        {boolean   t   =   true;   System.out.println
                (“Before the return”); if (t)


                        return;
                System.out.println (“This won‟t execute”);
        }
}
```

Output:



```
D:\JQR>javac ReturnDemo.java

D:\JQR>java  ReturnDemo
Before the return

D:\JQR>
```

Note: goto statement is not available in java, because it leads to confusion and forms infinite loops.

Arrays and Strings

Arrays: An array represents a group of elements of same data type Arrays are generally categorized into two types:

Single Dimensional arrays (or 1 Dimensional arrays)

Multi-Dimensional arrays (or 2 Dimensional arrays, 3 Dimensional arrays, …)

Single Dimensional Arrays: A one dimensional array or single dimensional array represents a row or a column of elements For example, the marks obtained by a student in 5 different subjects can be represented by a 1D array

We can declare a one dimensional array and directly store elements at the time of its declaration, as: int marks[] = {50, 60, 55, 67, 70};

We can create a 1D array by declaring the array first and then allocate memory for it by using new operator, as: int marks[]; //declare marks array

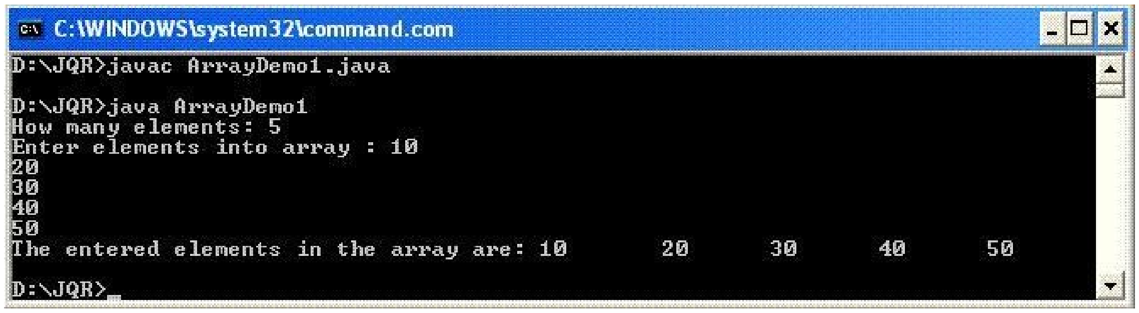
marks = new int[5]; //allot memory for storing 5 elements

These two statements also can be written as: int marks [] = new int [5];

Program 1: Write a program to accept elements into an array and display the same program to accept elements into an array and display the same import javaio*;

```
class ArrayDemo1
{public static void main (String args[]) throws IOException
        {//Create a BufferedReader class object (br)
                BufferedReader br = new BufferedReader (new InputStreamReader
                (Systemin));
                System.out.println ("How many elements: "
                ); int n = IntegerparseInt (brreadLine ());
                //create a 1D array with size n
                int a[] = new int[n];
                System.out.print ("Enter elements into array : ");
                for (int i = 0; i<n;i++)
                        a [i] = IntegerparseInt ( brreadLine ());
                System.out.print ("The entered elements in the array are:
                "); for (int i =0; i < n; i++)
                        System.out.print (a[i] + "\t");
        }
}
```

Output:



```
D:\JQR>javac ArrayDemo1.java

D:\JQR>java ArrayDemo1
How many elements: 5
Enter elements into array : 10
20
30
40
50
The entered elements in the array are: 10       20      30      40      50

D:\JQR>_
```

Multi-Dimensional Arrays (2D, 3D … arrays): Multi dimensional arrays represent 2D, 3D … arrays A two dimensional array is a combination of two or more (1D) one dimensional arrays A three dimensional array is a combination of two or more (2D) two dimensional arrays **Two Dimensional Arrays (2d array):** A two dimensional array represents several rows and columns of data To represent a two dimensional array, we should use two pairs of square braces [ ] [ ] after the array name For example, the marks obtained by a group of students in five different subjects can be represented by a 2D array

o We can declare a two dimensional array and directly store elements at the time of its declaration, as:

int marks[] [] = {{50, 60, 55, 67, 70},{62, 65, 70, 70, 81}, {72, 66, 77, 80, 69} };

We can create a two dimensional array by declaring the array first and then we can allot memory for it by using new operator as:

```
int marks[ ] [ ];    //declare marks array
marks = new int[3][5]; //allot memory for storing 15 elements
```

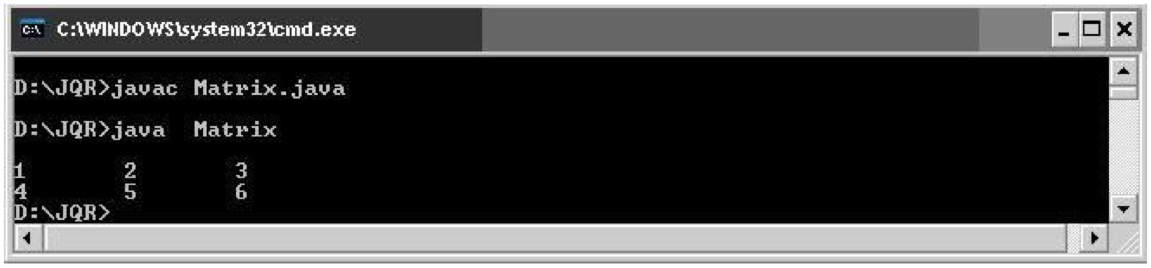
These two statements also can be written as:

```
int marks [ ][ ] = new int[3][5];
```

Program 2: Write a program to take a 2D array and display its elements in the form of a matrix

```
//Displaying a 2D array as a matrix
class Matrix
{public static void main(String args[])
        {//take a 2D array
                int x[ ][ ] = {{1, 2, 3}, {4, 5, 6}
                }; // display the array elements
                for (int i = 0 ; i < 2 ; i++)
                {System.out.println (); for (int j
                        = 0 ; j < 3 ; j++)
                                System.out.print(x[i][j] + "\t");
                }
        }
}
```

Output:



```
D:\JQR>javac Matrix.java

D:\JQR>java  Matrix

1       2       3
4       5       6
D:\JQR>
```

Three Dimensional arrays (3D arrays): We can consider a three dimensional array as a combination of several two dimensional arrays To represent a three dimensional array, we should use three pairs of square braces [ ] [ ] after the array name

o We can declare a three dimensional array and directly store elements at the time of its declaration, as:

int arr[ ] [ ] [ ] = {{{50, 51, 52},{60, 61, 62}}, {{70, 71, 72}, {80, 81, 82}}};

.

We can create a three dimensional array by declaring the array first and then we can allot memory for it by using new operator as:

int arr[ ] [ ] = new int[2][2][3]; //allot memory for storing 15 elements

arraynamelength: If we want to know the size of any array, we can use the property „length‟ of an array In case of 2D, 3D length property gives the number of rows of the array

Strings: A String represents group of characters Strings are represented as String objects in java

Creating Strings:

We can declare a String variable and directly store a String literal using assignment operator String str = "Hello";

We can create String object using new operator with some data

We can create a String by using character array also char arr[] = { 'p','r','o',"g","r","a","m"};

We can create a String by passing array name to it, as: String s2 = new String (arr);

We can create a String by passing array name and specifying which characters we need: String s3 = new String (str, 2, 3);

Here starting from 2nd character a total of 3 characters are copied into String s3

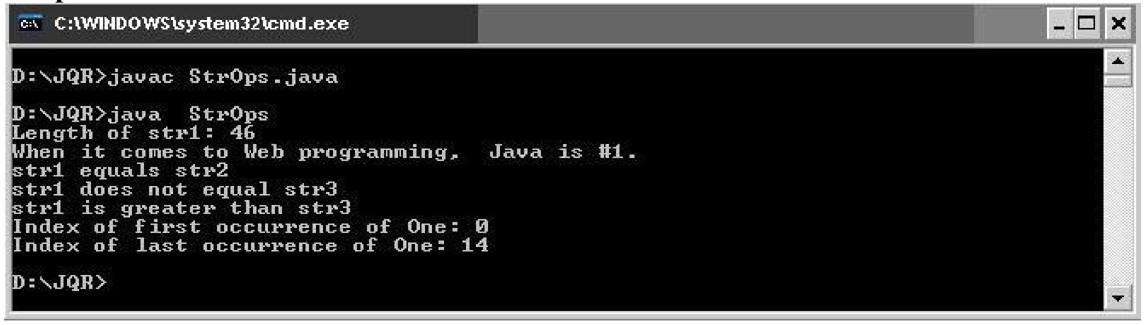
String Class Methods:

Method	Description
String concat (String str)	Concatenates calling String with str **Note:** + also used to do the same
int length ()	Returns length of a String
char charAt (int index)	Returns the character at specified location (from 0)
int compareTo (String str)	Returns a negative value if calling String is less than str, a positive value if calling String is greater than str or 0 if Strings are equal
boolean equals (String str)	Returns true if calling String equals str **Note:** == operator compares the references of the string objects It does not compare the contents of the objects equals () method compares the contents While comparing the strings, equals () method should be used as it yields the correct result
boolean equalsIgnoreCase (String str)	Same as above but ignores the case
boolean startsWith (String prefix)	Returns true if calling String starts with prefix
boolean endsWith (String suffix)	Returns true if calling String ends with suffix
int indexOf (String str)	Returns first occurrence of str in String
int lastIndexOf(String str)	Returns last occurrence of str in the String **Note:** Both the above methods return negative value, if str not found in calling String Counting starts from 0
String replace (char oldchar, char newchar)	returns a new String that is obtained by replacing all characters oldchar in String with newchar
String substring (int beginIndex)	returns a new String consisting of all characters from beginIndex until the end of the String
String substring (int beginIndex, int endIndex)	returns a new String consisting of all characters from beginIndex until the endIndex
String toLowerCase ()	converts all characters into lowercase
String toUpperCase ()	converts all characters into uppercase
String trim ()	eliminates all leading and trailing spaces

Program 3: Write a program using some important methods of String class

```
  program     using     String     class
methods class StrOps
{public static void main(String args [])
        {String str1 = "When it comes to Web programming, Java is #1";
                String str2 = new String (str1);
                String str3 = "Java strings are powerful";
                int result, idx;           char ch;
                System.out.println ("Length of str1: " + str1length
                   ()); display str1, one char at a
                time for(int i=0; i < str1length();
                i++)
                        System.out.print        (str1charAt        (i));
                System.out.println ();
                if (str1equals (str2) )
                        System.out.println ("str1 equals str2");
                else
                        System.out.println ("str1   does   not   equal
                str2");      if      (str1equals      (str3)      )
                         System.out.println      ("str1     equals
                         str3");
                else
                        System.out.println ("str1   does   not   equal
                str3"); result = str1compareTo (str3);
                if(result == 0)
                        System.out.println   ("str1   and   str3   are
                equal"); else if(result < 0)
                        System.out.println ("str1 is less than str3");
                else
                        System.out.println ("str1 is greater than str3");
                str2 = "One Two Three One"; // assign a new string to str2 idx =
                str2indexOf ("One");
                System.out.println ("Index of first occurrence of One: " +
                idx); idx = str2lastIndexOf("One");
                System.out.println ("Index of last occurrence of One: " + idx);
        }}
```

Output:



```
C:\WINDOWS\system32\cmd.exe

D:\JQR>javac StrOps.java

D:\JQR>java  StrOps
Length of str1: 46
When it comes to Web programming,  Java is #1.
str1 equals str2
str1 does not equal str3
str1 is greater than str3
Index of first occurrence of One: 0
Index of last occurrence of One: 14

D:\JQR>
```

We can divide objects broadly as mutable and immutable objects Mutable objects are those objects whose contents can be modified Immutable objects are those objects, once created can not be modified String objects are immutable The methods that directly manipulate data of the object are not available in String class

StringBuffer: StringBuffer objects are mutable, so they can be modified The methods that directly manipulate data of the object are available in StringBuffer class

Creating StringBuffer:

We can create a StringBuffer object by using new operator and pass the string to the object, as: StringBuffer sb = new StringBuffer ("Kiran");

We can create a StringBuffer object by first allotting memory to the StringBuffer object using new operator and later storing the String into it as:

StringBuffer sb = new StringBuffer (30);

In general a StringBuffer object will be created with a default capacity of 16 characters Here, StringBuffer object is created as an empty object with a capacity for storing 30 characters Even if we declare the capacity as 30, it is possible to store more than 30 characters into StringBuffer To store characters, we can use append () method as:

Sbappend (“Kiran”);

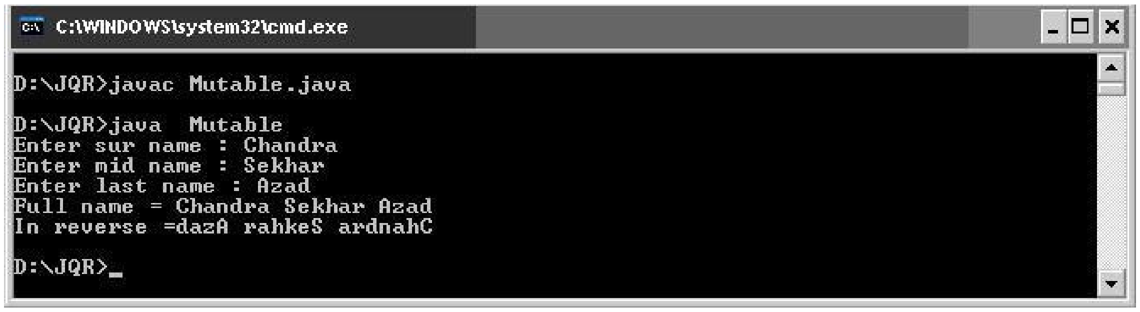
StringBuffer Class Methods:

Method	Description
StringBuffer append (x)	x may be int, float, double, char, String or StringBuffer It will be appended to calling StringBuffer
StringBuffer insert (int offset, x)	x may be int, float, double, char, String or StringBuffer It will be inserted into the StringBuffer at offset
StringBuffer delete (int start, int end)	Removes characters from start to end
StringBuffer reverse ()	Reverses character sequence in the StringBuffer
String toString ()	Converts StringBuffer into a String
int length ()	Returns length of the StringBuffer

Program 4: Write a program using some important methods of StringBuffer class

```
  program    using    StringBuffer    class
methods import javaio*;
class Mutable
{public static void main(String[] args) throws IOException
        {// to accept data from keyboard
        BufferedReader    br=new    BufferedReader    (new    InputStreamReader
                (Systemin)); System.out.print ("Enter sur name : ");
                String      sur=brreadLine      (      );
                System.out.print  ("Enter  mid  name  :
                ");    String    mid=brreadLine    (    );
                System.out.print  ("Enter  last  name  :
                "); String last=brreadLine ( );
                  create    String    Buffer    object
                StringBuffer sb=new StringBuffer (
                );
                  append sur, last to sb
                sbappend          (sur);
                sbappend (last);
                  insert   mid   after
                sur int n=surlength (
                ); sbinsert (n, mid);
                  display full name
                System.out.println  ("Full    name    =    "+sb);
                System.out.println  ("In  reverse  ="+sbreverse  (
                ));
        }
}
```

Output:



```
D:\JQR>javac Mutable.java

D:\JQR>java  Mutable
Enter sur name : Chandra
Enter mid name : Sekhar
Enter last name : Azad
Full name = Chandra Sekhar Azad
In reverse =dazA rahkeS ardnahC

D:\JQR>_
```

Introduction to OOPs

Languages like Pascal, C, FORTRAN, and COBOL are called procedure oriented programming languages Since in these languages, a programmer uses procedures or functions to perform a task When the programmer wants to write a program, he will first divide the task into separate sub tasks, each of which is expressed as functions/ procedures This approach is called procedure oriented approach

The languages like C++ and Java use classes and object in their programs and are called Object Oriented Programming languages The main task is divided into several modules and these are represented as classes Each class can perform some tasks for which several methods are written in a class This approach is called Object Oriented approach

Difference between Procedure Oriented Programming and OOP:

Procedure Oriented Programming	Object Oriented Programming
1 Main program is divided into small parts depending on the functions	1 Main program is divided into small object depending on the problem
2 The Different parts of the program connect with each other by parameter passing & using operating system	2 Functions of object linked with object using message passing
3 Every function contains different data	3 Data & functions of each individual object act like a single unit
4 Functions get more importance than data in program	4 Data gets more importance than functions in program
5 Most of the functions use global data	5 Each object controls its own data
6 Same data may be transfer from one function to another	6 Data does not possible transfer from one object to another
7 There is no perfect way for data hiding	7 Data hiding possible in OOP which prevent illegal access of function from outside of it This is one of the best advantages of OOP also
8 Functions communicate with other functions maintaining as usual rules	8 One object link with other using the message passing
9 More data or functions can not be added with program if necessary For this purpose full program need to be change	9 More data or functions can be added with program if necessary For this purpose full program need not to be change
10 To add new data in program user should be ensure that function allows it	10 Message passing ensure the permission of accessing member of an object from other object
11 Top down process is followed for program design	11 Bottom up process is followed for program design
12 Example: Pascal, Fortran	12 Example: C++, Java

Features of OOP:

Class: In object-oriented programming, a class is a programming language construct that is used as a blueprint to create objects This blueprint includes attributes and methods that the created objects all share Usually, a class represents a person, place, or thing - it is an abstraction of a concept within a computer program Fundamentally, it encapsulates the state and behavior of that which it conceptually represents It encapsulates state through data placeholders called member variables; it encapsulates behavior through reusable code called methods

General form of a class:

```
class class_name
{
        Properties (variables);
        Actions (methods);
}
```

eg: class Student

```
{//properties -- variables int
        rollNo;
        String        name;
        //methods          --
        actions          void
        display ()
        {
                System.out.println ("Student Roll Number is: " +
                rollNo); System.out.println ("Student Name is: " +
                name);
        }
}
```

Note: Variables inside a class are called as instance variables
Variables inside a method are called as method variables

Object: An Object is a real time entity An object is an instance of a class Instance means physically happening An object will have some properties and it can perform some actions Object contains variables and methods The objects which exhibit similar properties and actions are grouped under one class "To give a real world analogy, a house is constructed according to a specification Here, the specification is a blueprint that represents a class, and the constructed house represents the object"

- o To access the properties and methods of a class, we must declare a variable of that class type This variable does not define an object Instead, it is simply a variable that can refer to an object
- o We must acquire an actual, physical copy of the object and assign it to that variable We can do this using **new** operator The new operator dynamically allocates memory for an object and returns a reference to it This reference is, more or less, the address in memory of the object allocated by new This reference is then stored in the variable Thus, in Java, all class objects must be dynamically allocated

General form of an Object:

```
Class_name variable_name;            // declare reference to object
variable_name = new Class_name ( ); // allocate an object
```

eg:

```
Student s; // s is reference variable
s = new Student ();              // allocate an object to reference variable s
```

The above two steps can be combined and rewritten in a single statement as:

```
Student s = new Student ();
```

Now we can access the properties and methods of a class by using object with dot operator as:

```
srollNo, sname, sdisplay ()
```

.**Encapsulation:** Wrapping up of data (variables) and methods into single unit is called Encapsulation Class is an example for encapsulation Encapsulation can be described as a protective barrier that prevents the code and data being randomly accessed by other code defined outside the class Encapsulation is the technique of making the fields in a class private and providing access to the fields via methods If a field is declared private, it cannot be accessed by anyone outside the class

eg:

```
class Student
{
        private int rollNo;
        private String name;
        //methods -- actions
        void display ()
        {
                System.out.println ("Student Roll Number is: " +
                rollNo); System.out.println ("Student Name is: " +
                    name);
                }}
```

Abstraction: Providing the essential features without its inner details is called abstraction (or) hiding internal implementation is called Abstraction We can enhance the internal implementation without effecting outside world Abstraction provides security A class contains lot of data and the user does not need the entire data The advantage of abstraction is that every user will get his own view of the data according to his requirements and will not get confused with unnecessary data A bank clerk should see the customer details like account number, name and balance amount in the account He should not be entitled to see the sensitive data like the staff salaries, profit or loss of the bank etc So such data can be abstracted from the clerks view

eg:
```
        class Bank
        {private  int  accno;  private
                String name; private
                float           balance;
                private  float  profit;
                private float loan;
                void display_to_clerk ()
                {
                        System.out.println  ("Accno  =  "  +  accno);
                        System.out.println  ("Name  =  "  +  name);
                        System.out.println    ("Balance    =    "    +
                        balance);
                }
        }
```

In the preceding class, inspite of several data items, the display_to_clerk () method is able to access and display only the accno, name and balance values It cannot access profit and loan of the customer This means the profit and loan data is hidden from the view of the bank clerk

Inheritance: Acquiring the properties from one class to another class is called inheritance (or) producing new class from already existing class is called inheritance Reusability of code is main advantage of inheritance In Java inheritance is achieved by using extends keyword The properties with access specifier private cannot be inherited

eg:
```
        class Parent
  {
                    String parentName;
                    String familyName;
                }
                class Child extends Parent
                {
                        String childName;
                        int childAge;
                        void printMyName()
                        {
                                System.out.println (“My name is“+childName+” ”+familyName);
                        }
                }
```

In the above example, the child has inherited its family name from the parent class just by inheriting the class

Polymorphism: The word polymorphism came from two Greek words „poly" means „many" and „morphos" means „forms" Thus, polymorphism represents the ability to assume several different forms The ability to define more than one function with the same name is called Polymorphism

eg:int add (int a, int b) float add
(float a, int b) float
add (int a , float b)
void add (float a)
int add (int a)

Message Passing: Calling a method in a class is called message passing We can call methods of a class by using object with dot operator as:

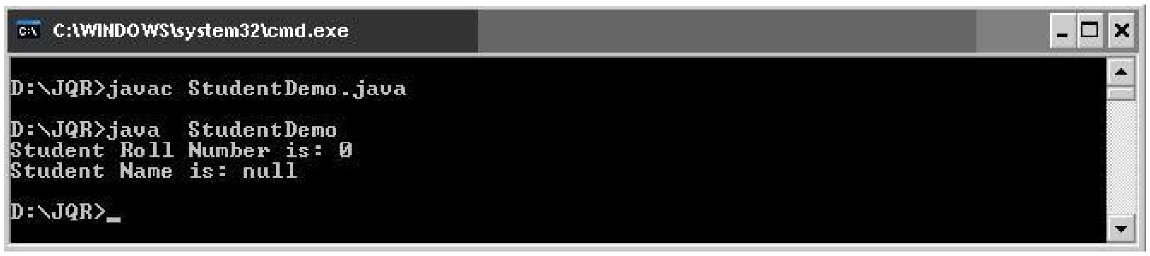
object_namemethod_name ();

eg: sdisplay (); obadd (2, 5); obprintMyName();

Program 1: Write a program to display details of student using class and object //Program to display the details of a student using class and object

```
class Student
{       int rollNo;             //properties -- variables
        String name;
        void display ()         //method -- action
        {       System.out.println ("Student Roll Number is: " +
                rollNo); System.out.println ("Student Name is: " +
                name);
        }
}
class StudentDemo
{public static void main(String args[])

        {



                //create an object to Student class
                Student s = new Student ();
                //call display () method inside Student class using object
                s sdisplay ();
        }
}
```

Output:



```
D:\JQR>javac StudentDemo.java

D:\JQR>java  StudentDemo
Student Roll Number is: 0
Student Name is: null

D:\JQR>_
```

When the programmer does not initialize the instance variables, java compiler will write code and initializes the variables with default values

Data Type	Default Value
Int	0
Float	00
Double	00
Char	Space
String	null
Class	null
Boolean	false

Initializing Instance Variables:

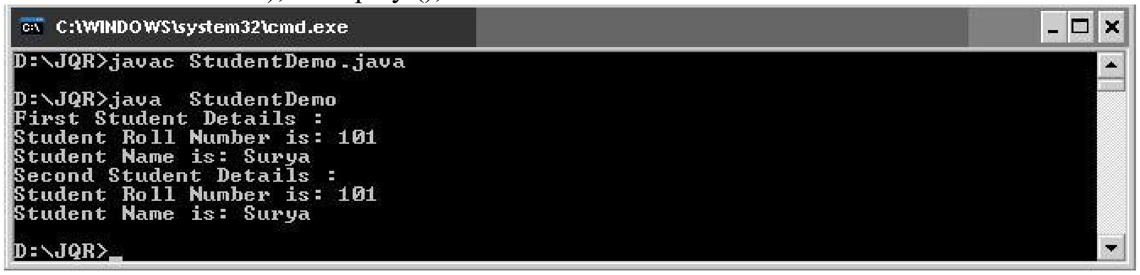
Type 1: We can initialize instance variables directly in the class using assignment operator In this type every object is initialized with the same data

```
int   rollNo   =   101;
String     name      =
"Kiran";
```

Program 2: Let us rewrite the Program 1

```
//Program to display the details of a student using class and
object class Student
{int rollNo = 101; String name =
          "Surya"; void display
          ()
          {System.out.println ("Student Roll Number is: " + rollNo);
                System.out.print ("Student Name is: " + name);
          }
}
class StudentDemo
{public static void main(String args[])
          {Student s1 = new Student (); System.out.println
                ("First Student Details : " );

                s1display ();
          }     Student s2 = new Student ();
}               System.out.println ("Second Student Details : "
                ); s2display ();
```

Output:



```
D:\JQR>javac StudentDemo.java

D:\JQR>java  StudentDemo
First Student Details :
Student Roll Number is: 101
Student Name is: Surya
Second Student Details :
Student Roll Number is: 101
Student Name is: Surya

D:\JQR>_
```

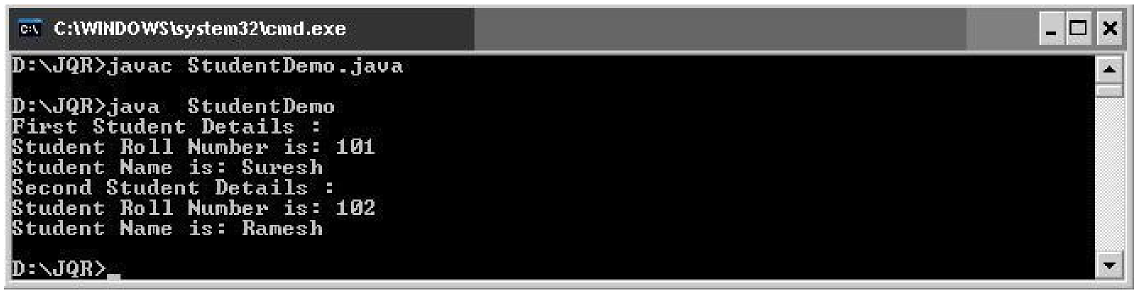
Type 2: We can initialize one class instance variables in another class using reference variable

```
                srollNo   =   101;
                sname = “Kiran”;
```

Program 3: Let us rewrite the Program 1

```
//Program to display the details of a student using class and
object class Student
{               int       rollNo;
                String   name;
                void display ()
                {
                          System.out.println  ("Student  Roll  Number  is:  "  +
                          rollNo); System.out.print ("Student Name is: " + name);
                }
}
class StudentDemo
{public static void main(String args[])
                {Student  s1  =  new  Student  ();  System.out.println
                          ("First Student Details : " ); s1rollNo = 101;
                          s1name = "Suresh";
                          s1display ();
                          Student s2 = new Student ();
                          System.out.println ("Second Student Details : "
                          );  s2rollNo  =  102;
                          s2name = "Ramesh";
                          s2display ();
                }
}
```

.**Output:**



```
D:\JQR>javac StudentDemo.java

D:\JQR>java  StudentDemo
First Student Details :
Student Roll Number is: 101
Student Name is: Suresh
Second Student Details :
Student Roll Number is: 102
Student Name is: Ramesh

D:\JQR>_
```

In this type of initialization the properties (variables in the class) are not available, if they are declared as private

Access Specifiers: An access specifier is a key word that represents how to access a member of a class There are four access specifiers in java

private: private members of a class are not available outside the class

public: public members of a class are available anywhere outside the class o **protected:** protected members are available outside the class

o **default:** if no access specifier is used then default specifier is used by java compiler Default members are available outside the class

Type 3: We can initialize instance variables using a constructor

Constructor:

o A constructor is similar to a method that initializes the instance variables of a class o A constructor name and classname must be same

o A constructor may have or may not have parameters Parameters are local variables to receive data

o A constructor without any parameters is called default constructor

egclass Student

```
        {int rollNo; String
                name;
                Student ()
                {rollNo  =  101;  name  =
                        “Kiran”;
                }
        }
```

A constructor with one or more parameters is called parameterized constructor

egclass Student

```
        {int    rollNo;    String
        name;

{rollNo  =    r;
 name = n;
                }
        }
```

A constructor does not return any value, not even void

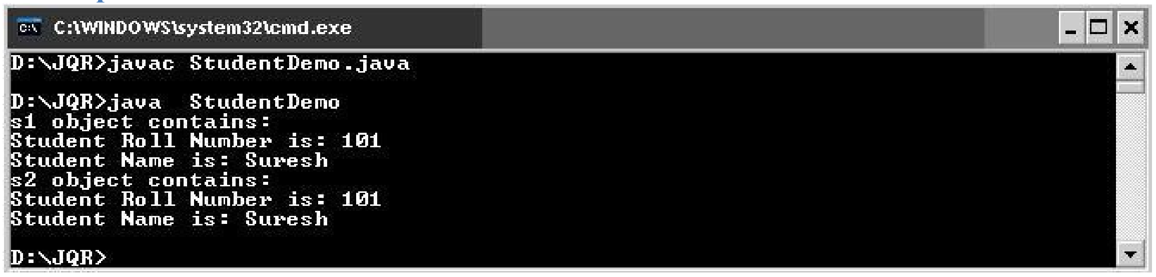
.A constructor is called and executed at the time of creating an object o A constructor is called only once per object

- o Default constructor is used to initialize every object with same data where as parameterized constructor is used to initialize each object with different data
- o If no constructor is written in a class then java compiler will provide default values

Program 4: Write a program to initialize student details using default constructor and display the same

```
//Program to initialize student details using default constructor and displaying the
same class Student
{int    rollNo;    String
            name;
            Student ()
            {rollNo = 101;
                    name = "Suresh";
            }
            void display ()
            {System.out.println  ("Student  Roll  Number  is:  "  +
                    rollNo); System.out.println ("Student Name is: " +
                    name);
            }
}
class StudentDemo
{public static void main(String args[])
            {Student     s1     =     new     Student     ();
                    System.out.println  ("s1  object  contains:
                    "); s1display (); Student     s2          =
                                new  Student     ();
                    System.out.println  ("s2  object  contains:
                    "); s2display ();
            }
}
```

Output:

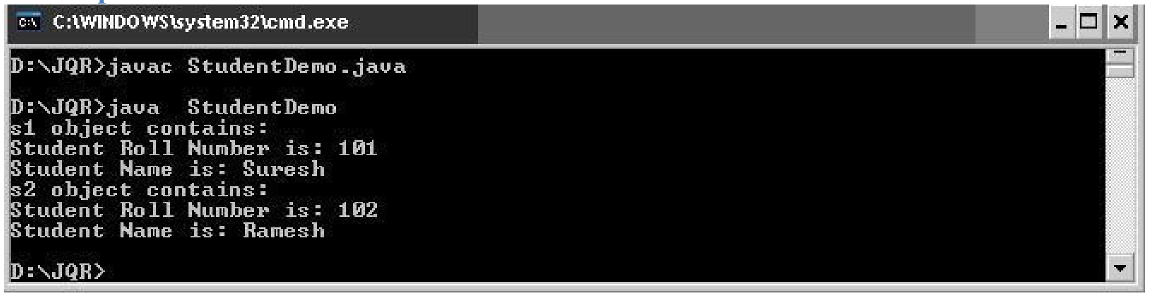


```
D:\JQR>javac StudentDemo.java

D:\JQR>java  StudentDemo
s1 object contains:
Student Roll Number is: 101
Student Name is: Suresh
s2 object contains:
Student Roll Number is: 101
Student Name is: Suresh

D:\JQR>
```

Program 5: Write a program to initialize student details using Parameterized constructor and display the same

```
//Program   to   initialize   student   details   using   parameterized
constructor class Student
{int rollNo;
.                                                                    .
                String name;
                Student (int r, String n)
                {rollNo    =    r;
                        name = n;
                }
                void display ()
                {System.out.println ("Student Roll Number is: " + rollNo);
                        System.out.println ("Student Name is: " + name);
                }
    }
    class StudentDemo
    {public static void main(String args[])
                {Student s1 = new Student (101, “Suresh”);
                        System.out.println (“s1 object contains: “
                        ); s1display ();
                        Student s2 = new Student (102, “Ramesh”);
                        System.out.println (“s2 object contains: “ );
                        s2display ();
        }
    }
```

Output:



```
D:\JQR>javac StudentDemo.java

D:\JQR>java  StudentDemo
s1 object contains:
Student Roll Number is: 101
Student Name is: Suresh
s2 object contains:
Student Roll Number is: 102
Student Name is: Ramesh

D:\JQR>
```

The keyword ‘this’: There will be situations where a method wants to refer to the object which invoked it To perform this we use „this‟ keyword There are no restrictions to use „this‟ keyword we can use this inside any method for referring the current object This keyword is always a reference to the object on which the method was invoked We can use „this‟ keyword wherever a reference to an object of the current class type is permitted „this‟ is a key word that refers to present class object It refers to

Present class instance variables
Present class methods
Present class constructor

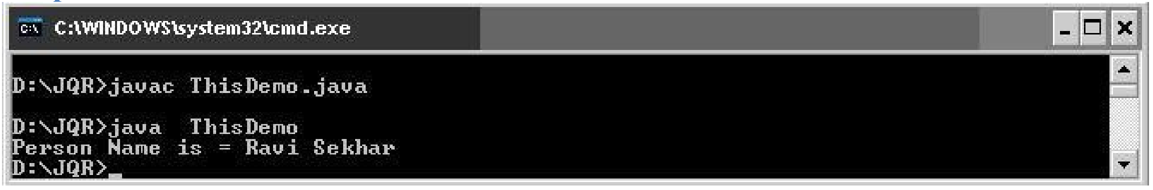
Program 6: Write a program to use „this‟ to refer the current class parameterized constructor and current class instance variable

```
//this   demo
class Person
{String name;
```

.   .

```
        Person ( )
        {this   ("Ravi   Sekhar");   //   calling   present   class   parameterized   constructor
                thisdisplay ( ); // calling present class method
        }
        Person (String name)
        {thisname = name; // assigning present class variable with parameter "name"
        }
        void display( )
        {System.out.println ("Person Name is = " + name);
        }
}
class ThisDemo
{public static void main(String args[])
        {
                Person p = new Person ( );
        }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe

D:\JQR>javac ThisDemo.java

D:\JQR>java  ThisDemo
Person Name is = Ravi Sekhar
D:\JQR>_
```

Garbage Collection: Generally memory is allocated to objects by using „new" operator and deleting an allocated memory is uncommon This deletion of memory is supported by delete operator in C++ but this deletion of allocated memory works automatically in Java This automatic deletion of already allocated but unused memory is called as garbage collection This operation of garbage collection is accomplished by a method named "gc ()" This method is used for garbage collection

The finalize( ) Method: It is possible to define a method that will be called just before an object's final destruction by the garbage collector This method is called finalize( ) method To add a finalizer to a class, simply define the finalize( ) method The Java runtime calls that method whenever it is about to recycle an object of that class Inside the finalize( ) method specify those actions that must be performed before an object is destroyed The finalize( ) method has this general form:

```
protected void finalize( )
{
        // finalization code here
}
```

Here, the keyword protected is a specifier that prevents access to finalize ( ) by code defined outside its class This means that you cannot know whenor even iffinalize ( ) will be executed For example, if your program ends before garbage collection occurs, finalize ( ) will not execute

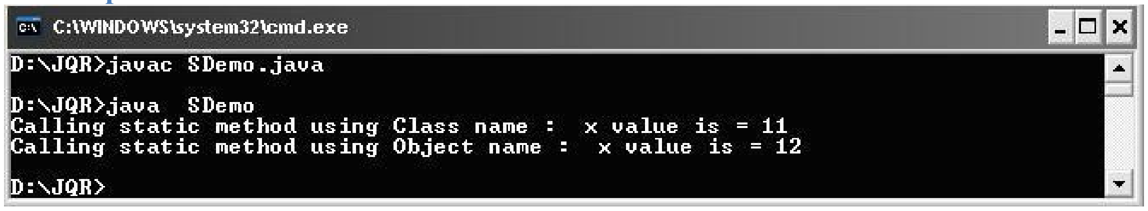
Static Methods:

o Static methods can read and act upon static variables
o Static methods cannot read and act upon instance variables
o Static variable is a variable whose single copy is shared by all the objects
o Static methods are declared using keyword static
o Static methods can be called using objectnamemethodname (or) classnamemethodname
o From any object, if static variable is modified it affects all the objects Static variables are stored on method area

Program 2: Write a program to access static variable using static method //static method accessing static variable

```
class Sample
{static int x = 10; static void
            display( )
            {x++;
                    System.out.println (" x value is = " + x);
            }
}
class SDemo
{public static void main(String args[])
            {System.out.print ("Calling static method using Class name : ");
                    Sampledisplay ();
                    Sample s1 = new Sample ( );
                    System.out.print ("Calling static method using Object name :
                    "); s1display ();
            }
    }
```

Output:



```
D:\JQR>javac SDemo.java

D:\JQR>java  SDemo
Calling static method using Class name :  x value is = 11
Calling static method using Object name :  x value is = 12

D:\JQR>
```

Inheritance

Inheritance: Creating new class from existing class such that the features of existing class are available to the new class is called inheritance Already existing class is called super class & produced class is called sub class Using inheritance while creating sub classes a programmer can reuse the super class code without rewriting it
Syntax: class subclass_name extends superclass_name
eg: class Child extends Parent

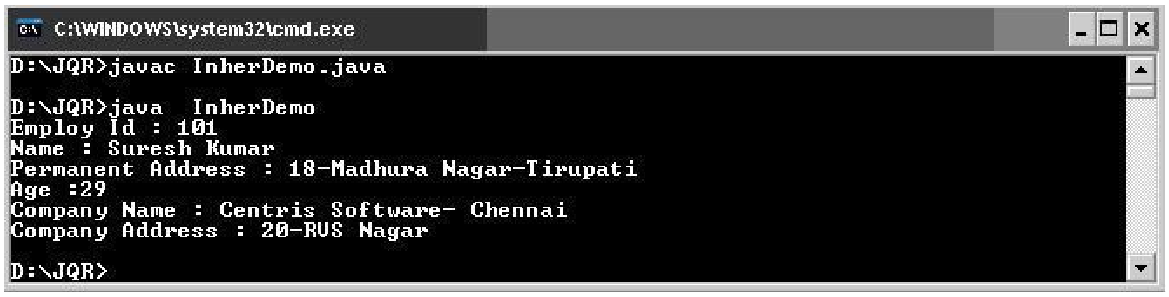
Program 1: Write a program to create a Person class which contains general details of a person and create a sub class Employ which contains company details of a person Reuse the general details of the person in its sub class

```
  Inheritance
Example class Person
{String name;
        String
        permanentAddress;    int
        age;
        void set_PermanentDetails (String name, String permanentAddress, int age)
        {thisname = name; thispermanentAddress =
                permanentAddress; thisage = age;

        }
        void get_PermanentDetails ()
        {System.out.println ("Name : " + name);
                System.out.println ("Permanent Address : " +
                permanentAddress); System.out.println ("Age :" + age);
        }
}
class Employ extends Person
{int id;
        String  companyName;
        String
        companyAddress;
        Employ (int id, String name, String permanentAddress, int age,
                                String companyName, String companyAddress)
        {thisid = id;
                set_PermanentDetails (name, permanentAddress, age);
                thiscompanyName = companyName;
                thiscompanyAddress = companyAddress;
        }
        void get_EmployDetails ()
        {System.out.println ("Employ Id : " + id);
                get_PermanentDetails ();
                System.out.println ("Company Name : "+ companyName);
                System.out.println ("Company Address : "+companyAddress);
        }
```

.   .

```
}
class InherDemo
{public static void main (String args [])
{Employ e1 = new Employ (101, "Suresh Kumar", "18-Madhura Nagar-Tirupati", 29,
                "Centris Software- Chennai", "20-RVS Nagar");
                e1get_EmployDetails ();
        }
}
```

Output:



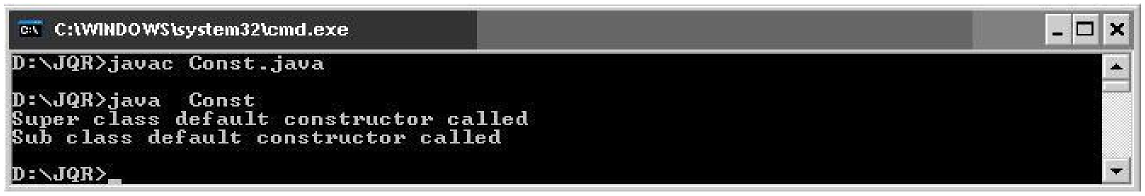
```
D:\JQR>javac InherDemo.java

D:\JQR>java  InherDemo
Employ Id : 101
Name : Suresh Kumar
Permanent Address : 18-Madhura Nagar-Tirupati
Age :29
Company Name : Centris Software- Chennai
Company Address : 20-RVS Nagar

D:\JQR>
```

Program 2: Write a program to illustrate the order of calling of default constructor in super and sub class

Default constructors in super and sub

```
class class One
{       One ( )         //super class default constructor
        {
                System.out.println ("Super class default constructor called");
        }
}
class Two extends One
{       Two ( )         //sub class default constructor
        {
                System.out.println ("Sub class default constructor called");
        }
}
class Const
{public static void main (String args[])

        { Two t=new Two ( ); //create sub class object }

 }
```

Output:



```
D:\JQR>javac Const.java

D:\JQR>java  Const
Super class default constructor called
Sub class default constructor called

D:\JQR>_
```

.Super class default constructor is available to sub class by default
First super class default constructor is executed then sub class default constructor is executed
Super class parameterized constructor is not automatically available to subclass super is the key word that refers to super class

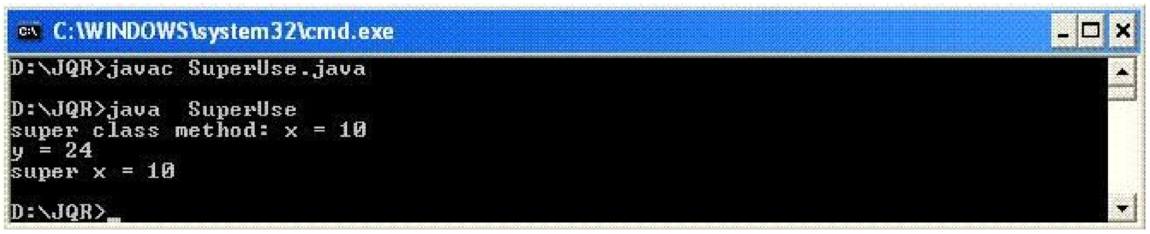
The keyword 'super':

- super can be used to refer super class variables as: supervariable
- super can be used to refer super class methods as: supermethod ()
- super can be used to refer super class constructor as: super (values)

Program 3: Write a program to access the super class method, super class parameterized constructor and super class instance variable by using super keyword from sub class

```
   super refers to super class- constructors, instance variables and
methods class A
  {int x; A (int
  x)
{
            thisx = x;
        }
        void show( )
        {System.out.println("super class method: x = "+x);
        }
}
class B extends A
{       int y;
        B (int a,int b)
        {
            super(a)        // (or) x=a;
            ;
            y=b;
        }
        void show( )
        {       supershow                    ();
            System.out.println    ("y    =
            "+y);
            System.out.println (" super x = " + superx);
        }
}
class SuperUse
{
        public static void main(String args[])
        {B ob = new B (10, 24);
            obshow ( );
        }
}
```

Output:



```
D:\JQR>javac SuperUse.java

D:\JQR>java  SuperUse
super class method: x = 10
y = 24
super x = 10

D:\JQR>_
```

Super key word is used in sub class only
The statement calling super class constructor should be the first one in sub class constructor

Polymorphism

Polymorphism came from the two Greek words „poly" means many and morphos means forms If the same method has ability to take more than one form to perform several tasks then it is called polymorphism It is of two types: Dynamic polymorphism and Static polymorphism

Dynamic Polymorphism: The polymorphism exhibited at run time is called dynamic polymorphism In this dynamic polymorphism a method call is linked with method body at the time of execution by JVM Java compiler does not know which method is called at the time of compilation This is also known as dynamic binding or run time polymorphism Method overloading and method overriding are examples of Dynamic Polymorphism in Java o **Method Overloading:** Writing two or more methods with the same name, but with a

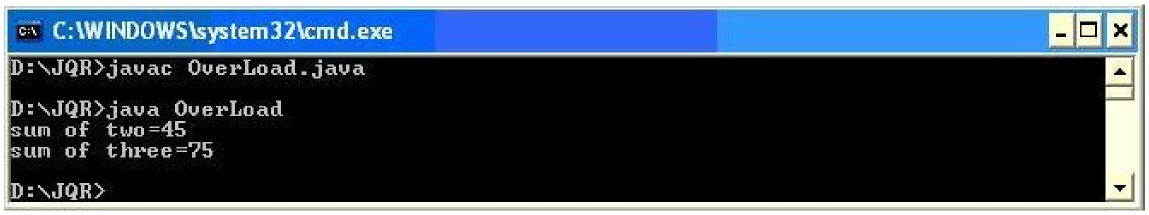
difference in the method signatures is called method over loading Method signature represents the method name along with the method parameters In method over loading JVM understands which method is called depending upon the difference in the method signature The difference may be due to the following:

There is a difference in the no of
parameters void add (int a,int b)
void add (int a,int b,int c)
There is a difference in the data types of
parameters void add (int a,float b)
void add (double a,double b)
There is a difference in the sequence of
parameters void swap (int a,char b)
void swap (char a,int b)

Program 1: Write a program to create a class which contains two methods with the same name but with different signatures

```
  overloading    of    methods    ---------    Dynamic
polymorphism class Sample
{void add(int a,int b)
            {
                    System.out.println ("sum of two="+ (a+b));
            }
            void add(int a,int b,int c)
            {
                    System.out.println ("sum of three="+ (a+b+c));
            }
}
class OverLoad
{public static void main(String[] args)
            {Sample s=new Sample ( ); sadd
                    (20, 25);
                    sadd (20, 25, 30);
            }
}
```

Output:



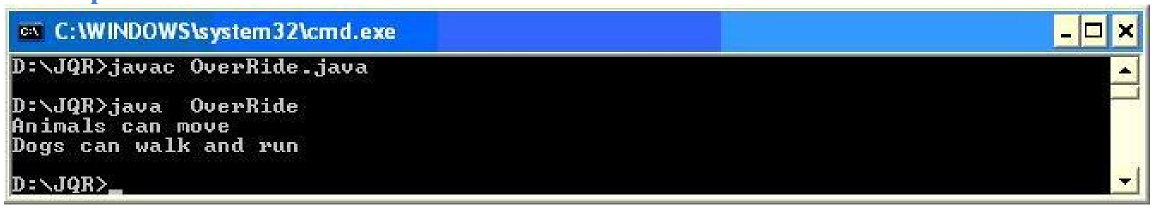
```
D:\JQR>javac OverLoad.java

D:\JQR>java OverLoad
sum of two=45
sum of three=75

D:\JQR>
```

Method Overriding: Writing two or more methods in super & sub classes with same name and same signatures is called method overriding In method overriding JVM executes a method depending on the type of the object

Program 2: Write a program that contains a super and sub class which contains a method with same name and same method signature, behavior of the method is dynamically decided //overriding of methods---------------- Dynamic polymorphism

```
class Animal
{ void move()
{
            System.out.println ("Animals can move");
    }
}
class Dog extends Animal
{ void move()
    {
            System.out.println ("Dogs can walk and run");
    }
}
public class OverRide
{ public static void main(String args[])
    {Animal a = new Animal (); // Animal reference and object
            Animal b = new Dog (); // Animal reference but Dog
            object amove (); // runs the method in Animal class
            bmove (); //Runs the method in Dog class
     } }
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac OverRide.java

D:\JQR>java  OverRide
Animals can move
Dogs can walk and run

D:\JQR>_
```

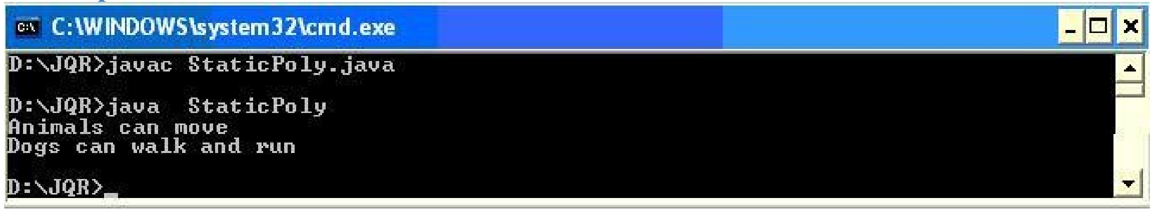
Achieving method overloading & method overriding using instance methods is an example of dynamic polymorphism

Static Polymorphism: The polymorphism exhibited at compile time is called Static polymorphism Here the compiler knows which method is called at the compilation This is also called compile time polymorphism or static binding Achieving method overloading & method overriding using private, static and final methods is an example of Static Polymorphism

Program 3: Write a program to illustrate static polymorphism

```
//Static Polymorphism
class Animal
{static void move ()
                {System.out.println ("Animals can move");
                }
}
class Dog extends Animal
{static void move ()
                {System.out.println ("Dogs can walk and run");
                }
}
public class StaticPoly
{public static void main(String args[])
                {Animalmove         ();
                        Dogmove ();
                }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac StaticPoly.java

D:\JQR>java  StaticPoly
Animals can move
Dogs can walk and run

D:\JQR>_
```

The keyword 'final':

final keyword before a class prevents inheritance **eg:** final class A
    class B extends A //invalid

final keyword before a method prevents overriding

final keyword before a variable makes that variable as a constant **eg:** final double PI = 314159; //PI is a constant

Type Casting: Converting one data type into another data type is called casting Type cast operator is used to convert one data type into another data type Data type represents the type of the data stored into a variable There are two kinds of data types:

Primitive Data type: Primitive data type represents singular values
eg: byte, short, int, long, float, double, char, boolean

Using casting we can convert a primitive data type into another primitive data type This is done in two ways, widening and narrowing

Widening: Converting a lower data type into higher data type is called widening byte, short, int, long , float, double
eg: char ch = 'a'; int
n = (int) ch;
eg: int n = 12;
float f = (float) n;
Narrowing: Converting a higher data type into lower data type is called narrowing
eg: int i = 65;
char ch = (char)
i; **eg:** float f = 125;
int i = (int) f;

Referenced Data type: Referenced data type represents multiple values
eg: class, String

Using casting we can convert one class type into another class type if they are related by means of inheritance

Generalization: Moving back from subclass to super class is called generalization or widening or upcasting

Specialization: Moving from super class to sub class is called specialization or narrowing or downcasting

Program 4: Write a program to convert one class type into another class type
conversion of one class type into another class

```
type class One
{void show1()
            {System.out.println ("One's method");
            }
}
class Two extends One
{void show2()
            {System.out.println ("Two's method");
            }
}
class Ex3
{
            public static void main(String args[])
            {
    /* If super class reference is used to refer to super class object then only super class
    members are available to programmer */
                    One ob1 = new One ();
                    ob1show1 ();
    /* If sub class reference is used to refer to sub class object then super class members as
    well as sub class members are available to the programmer */
                    Two ob2 = new Two();
                    ob2show1();
                    ob2show2();
    /* If super class reference is used to refer to sub class object then super class methods
    are available, sub class methods are not available unless they override super class
    methods */
                    One ob3 = (One) new Two(); // Generalization
                    ob3show1();
    /* It is not possible to access any methods if we use subclass object to refer to super
    class as above */
                    Two ob4 = (Two) new One();
                    ob4show1();
                    ob4show2();
              // Specialization
                    One ob5 = (One) new Two();
                    Two ob6 = (Two) ob5;
                    ob6show1();
                    ob6show2();
              }
}
```

Note: Using casting it is not possible to convert a primitive data type into a referenced data type and vice-versa For this we are using Wrapper classes

Abstract Class

A method with method body is called concrete method In general any class will have all concrete methods A method without method body is called abstract method A class that contains abstract method is called abstract class It is possible to implement the abstract methods differently in the subclasses of an abstract class These different implementations will help the programmer to perform different tasks depending on the need of the sub classes Moreover, the common members of the abstract class are also shared by the sub classes

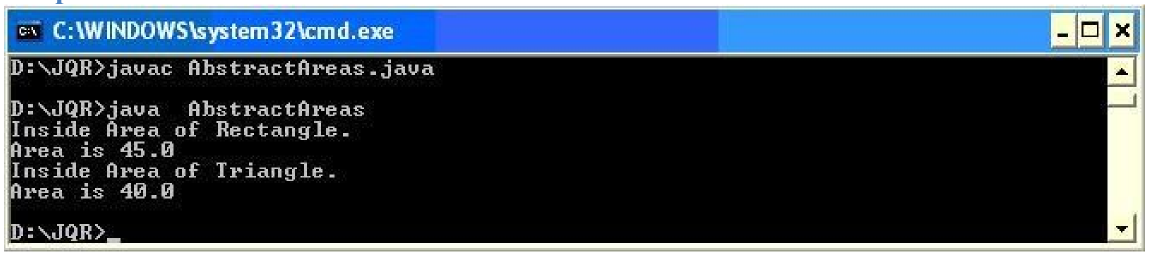
The abstract methods and abstract class should be declared using the keyword abstract We cannot create objects to abstract class because it is having incomplete code Whenever an abstract class is created, subclass should be created to it and the abstract methods should be implemented in the subclasses, then we can create objects to the subclasses

An abstract class is a class with zero or more abstract methods
An abstract class contains instance variables & concrete methods in addition to abstract methods
It is not possible to create objects to abstract class
But we can create a reference of abstract class type
All the abstract methods of the abstract class should be implemented in its sub classes
If any method is not implemented, then that sub class should be declared as „abstract"
Abstract class reference can be used to refer to the objects of its sub classes
Abstract class references cannot refer to the individual methods of sub classes
A class cannot be both „abstract" & „final"
eg: final abstract class A // invalid

Program 1: Write an example program for abstract class

```
  Using    abstract    methods    and
classes abstract class Figure
{double        dim1;
        double
        dim2;

        {       dim1 = a;
dim2 = b;

        }
        abstract double area ();

// area is now an abstract method
}
class Rectangle extends Figure
{Rectangle (double a, double b)
        {super (a, b);
        }
        double area ()          // override area for rectangle
        {System.out.println ("Inside Area of Rectangle"); return
                dim1 * dim2;
        }
}

class Triangle extends Figure
{Triangle (double a, double b)
        {super (a, b);
        }
        double area()           // override area for right triangle
        {System.out.println ("Inside Area of Triangle"); return dim1 * dim2 / 2;
        }
}
class AbstractAreas
{public static void main(String args[])
        {// Figure f = new Figure(10, 10); // illegal now Rectangle r = new
                Rectangle(9, 5); Triangle t = new Triangle(10, 8);
                System.out.println("Area is " + rarea()); System.out.println("Area is
                "
                + tarea());
        }
}
```

Output:



```
D:\JQR>javac AbstractAreas.java

D:\JQR>java  AbstractAreas
Inside Area of Rectangle.
Area is 45.0
Inside Area of Triangle.
Area is 40.0

D:\JQR>_
```

Wrapper Classes

Wrapper Classes are used to convert primitive data types into objects A wrapper class is a class whose object wraps the primitive data type Wrapper classes are available in javalang package Different applications on internet send data or recieve data in the form of objects The classes in javautil package act upon objects only

Primitive Data type	Wrapper class
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

Character Class: The Character class wraps a value of the primitive type char in an object An object of type character contains a single field whose type is char We can create Character class object as: Character obj = new Character (ch); // where ch is a character

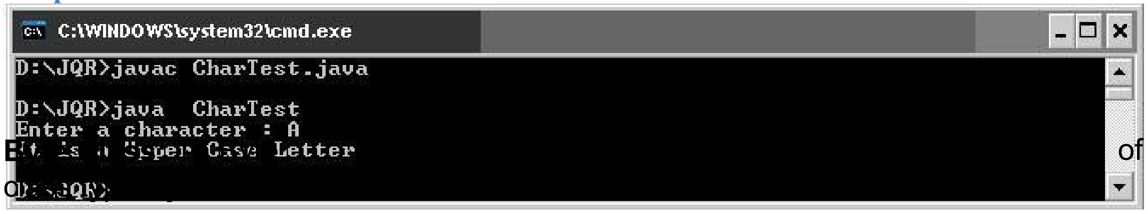
Methods of Character Class:

Method	Description
char charValue ()	returns the char value of the invoking object
int compareTo (Character obj)	This method is useful to compare the contents of two Character class objects
static boolean isDigit (char ch)	returns true if ch is a digit (0 to 9) otherwise returns false
static boolean isLetter (char ch)	returns true if ch is a letter (A to Z or a to z)
static boolean isUpperCase (char ch)	returns true if ch is an uppercase letter (A to Z)
static boolean isLowerCase (char ch)	returns true if ch is a lower case letter (a to z)
static boolean isSpaceChar (char ch)	returns true if ch is coming from SPACEBAR
static boolean isWhiteSpace(char ch)	returns true if ch is coming from TAB, ENTER, BackSpace
static char toUpperCase (char ch)	converts ch into uppercase
static char toLowerCase (char ch)	converts ch into lowercase

Program 1: Write a program which shows the use of Character class methods

```
//Testing a char
import javaio*;
class CharTest
{public static void main(String args[]) throws IOException
        {BufferedReader br = new BufferedReader (new InputStreamReader (Systemin));
                System.out.print ("Enter a character : " );
                char ch = (char) brread();

                if      (CharacterisDigit      (ch)      )
                        System.out.println   ("It   is   a
                        digit");
                else if (CharacterisUpperCase (ch)) System.out.println
                        ("It is a Upper Case Letter");
                else    if   (CharacterisLowerCase    (ch)          )
                        System.out.println  ("It  is  a  Lower  Case
                        Letter");
                else   if  (   CharacterisSpaceChar   (ch))
                        System.out.println  ("It  is  a  Space
                        bar");
                else if ( CharacterisWhitespace (ch) )
                        System.out.println ("Dont know what is this character");
        }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac CharTest.java

D:\JQR>java  CharTest
Enter a character : A
It is a Upper Case Letter

D:\JQR>
```

Constructors:

- Byte (byte num)
  eg: Byte b1 = new Byte (98);
- Byte (String str)
  eg: String str = "98";
  Byte b2 = new Byte (str);

Methods of Byte Class:

Method	Description
byte byteValue ()	returns the value of invoking object as a byte
int compareTo (Byte b)	This method is useful to compare the contents of two Byte class objects
static byte parseByte(String str)	returns byte equivalent of the number contained in the string specified by 'str'
String toString ()	returns a String that contains the decimal equivalent of the invoking object
static Byte valueOf (String str)	returns a Byte object that contains the value specified by the String 'str'

Program 2: Write a program which shows the use of Byte class methods

```
//Creating and comparing Byte Objects
import javaio*;
class Bytes
{public static void main(String args[]) throws IOException

     {BufferedReader br = new BufferedReader (new InputStreamReader (Systemin));
          System.out.print ("Enter a byte number: ");
          String str = brreadLine ();
          //convert str into Byte
          object Byte b1 = new Byte
          (str);
          System.out.print ("Enter a another byte number:
          "); str = brreadLine ();
          //convert str into Byte obj
          Byte b2 = BytevalueOf
          (str); //compare b1 and b2
          int n =
          b1compareTo(b2); if
          (n==0)
               System.out.println ("Both are
          Same"); else if(n>0)
               System.out.println (b1 + "is bigger");
else
               System.out.println (b2 + " is bigger");
  }
}
```

Short Class: Short class wraps a value of primitive data type 'short' in its object Short class object contains a short type field that stores a short number

Constructors:

- Short (short num)
- Short (String str)

Methods:

Method	Description
int compareTo (Short obj)	This method compares the numerical value of two Short class objects and returns 0,-ve, +ve value
Boolean equals (Object obj)	This method compares the Short object with any other object obj and returns true if both have same content
static short parseShort (String str)	This method returns int equivalent of the String str
String toString ()	This method returns a String form of the Short object
static Short valueOf (String str)	This method converts a String str that contains a short number into Short class object and returns that object

Integer Class: Integer class wraps a value of the primitive type 'int' in an object An object of type Integer contains a single field whose type is int

Constructors:

- Integer (int num)
- Integer (String str)

Methods:

Method	Description
int intVlaue ()	returns the value of the invoking object as an int
int compareTo (Integer obj)	compares the numerical value of the invoking object with that of 'obj' returns zero or -ve value or +ve value
static int parseInt (String str)	returns int equivalent of the String str
String toString ()	returns a String form of the invoking object
static Integer valueOf (String str)	returns an Integer object that contains the value shown by str
static String toBinaryString (int i)	returns a String representation of the integer argument in base2
static String toHexString (int i)	returns a String representation of the integer argument in base 16
static String toOctalString (int i)	returns a String representation of the integer argument in base 8

Float Class: Float class wraps a value of primitive type float in an object An object of type float contains a single field whose type is float

Constructors:

- Float (float num)

· Float (String str)

Methods:

Method	Description
float floatValue ()	returns the value of the invoking object as a float
double doubleValue ()	returns the value of the invoking object as a double
int compareTo (Float f)	Compares the numerical value of the invoking object with that of 'f' returns zero or +ve or -ve value
static float parseFloat (String str)	returns the float equivalent of the String str
String toString ()	returns the String equivalent of invoking object
static Float valueOf (String str)	returns the Float object with the value specified by String str

Long Class: The Long class contains a primitive long type data The object of Long class contains a field where we can store a long value

Constructors: Long has two constructors

Long (long num): Long object can be created as: Long obj = new Long (123000);
Long(String str): String str = "12300044";
Long obj = new Long (str);

Methods:

Method	Description
int compareTo(Long obj)	This method compares the numerical value of two Long class objects and returns),-ve,+ve value
static long parseLong(String str)	This method returns long equivalent of the String str
String toString()	This method converts Long object into String object and returns the String object
Static Long valueOf(String str)	This method converts a string str that contains some long number into Long object and returns that object

Boolean class: The Boolean class object contains a primitive 'boolean' type data The object of Boolean class contains a field where we can store a boolean value

Constructors:

Boolean obj = new Boolean (true);
String str ="false";
Boolean obj = new Boolean (str);

Methods:

Method	Description
int compareTo(Boolean obj)	This method compares the numerical value of two Boolean class objects and returns 0,-ve,+ve value
static boolean parseBoolean(String str)	This method returns boolean equivalent of the String str
String toString()	This method converts Boolean object into a String

	object and returns the String object
static Boolean valueOf(String str)	This method converts a String str that contains a boolean value into Boolean object and returns that object

Double Class: Double class wraps a value of primitive type Double in an Object

Constructors:

- Double (double num)
- Double (String str)

Methods:

Method	Description
double doubleValue()	returns the value of the invoking object as a double
float floatValue()	returns the value of the invoking object as a float
int compareTo(Double d)	This method compares the numerical value of two Double class objects and returns 0,-ve,+ve value
static double parseDouble(String str)	returns the double equivalent of the String str
String toString()	This method converts Double object into a String object and returns the String object
static Double valueOf(String str)	returns the Double object with the value specified by String str

Math class: The class Math contains methods for performing basic numeric operations

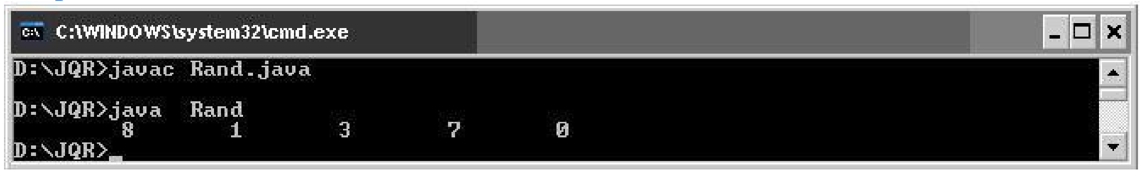
Methods:

Method	Description
static double sin(double arg)	returns the sine value of the arg arg is in radians
static double cos(double arg)	returns the cosine value of the arg
static double tan(double arg)	returns the tangent value of the arg
static double log(double arg)	returns the natural logarithm value or arg
static double pow(double x, double n)	returns x to the power of n value
static double sqrt(double arg)	returns the square root of arg
static double abs(double arg)	returns the absolute value of arg
static double ceil(double arg)	returns the smallest integer which is greater or equal to arg
static double floor(double arg)	returns the greatest integer which is lower or equal to arg
static double min(arg1,arg2)	returns the minimum of arg1 and arg2
static double max(arg1,arg2)	returns the maximum of arg1 and arg2
static long round(arg)	returns the rounded value of arg
static double random()	returns a random number between 0 and 1
static double toRadians(double angle)	converts angle in degrees into radians
static double toDegrees(double angle)	converts angle in radians into degrees

Program 3: Write a program to print random numbers using Math class

```
//Generating random numbers
class Rand
{
        public   static   void   main(String   args[])   throws
        Exception {
                while(true)
                {
                        double d = 10 * Mathrandom();
                        int i = (int) d;
                        System.out.print  ("\t"  +
                        i); if ( i == 0)
                                Systemexit (0);
                }
        }
}
```

Output:



```
D:\JQR>javac Rand.java

D:\JQR>java  Rand
        8       1       3       7       0
D:\JQR>_
```

UNIT - II

Packages- Defining a Package, CLASSPATH, Access protection, importing packages.

Interfaces- defining an interface, implementing interfaces, Nested interfaces, applying interfaces, variables in interfaces and extending interfaces.

Stream based I/O (java.io) – The Stream classes-Byte streams and Character streams, Reading console Input and Writing Console Output, File class, Reading and writing Files, Random access file operations, The Console class, Serialization, Enumerations, auto boxing, generics.

Packages

A package is a container of classes and interfaces A package represents a directory that contains related group of classes and interfaces For example, when we write statemens like:

import javaio*;

Here we are importing classes of javaio package Here, java is a directory name and io is another sub directory within it The „*" represents all the classes and interfaces of that io sub directory We can create our own packages called user-defined packages or extend the available packages User-defined packages can also be imported into other classes and used exactly in the same way as the Built-in packages Packages provide reusability

General form for creating a package:

package packagename;

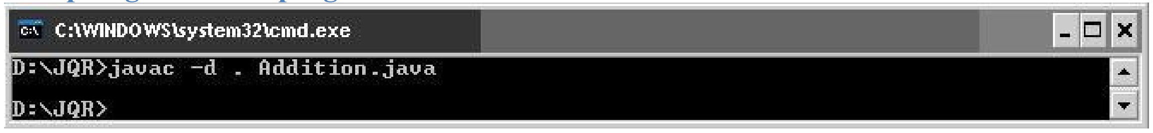
eg: package pack;

The first statement in the program must be package statement while creating a package While creating a package except instance variables, declare all the members and the class itself as public then only the public members are available outside the package to other programs

Program 1: Write a program to create a package pack with Addition class

```
//creating a package
package pack;
public class Addition
{private double d1,d2;
        public Addition(double a,double b)
        {d1 = a; d2 =
                b;
        }
        public void sum()
        {System.out.println ("Sum of two given numbers is : " + (d1+d2) );
        }
}
```

Compiling the above program:



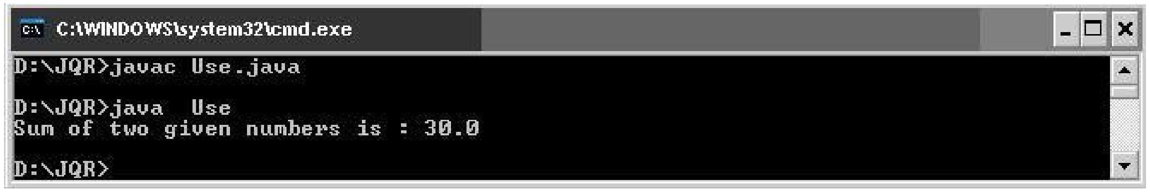
```
D:\JQR>javac -d . Addition.java

D:\JQR>
```

The –d option tells the Java compiler to create a separate directory and place the class file in that directory (package) The () dot after –d indicates that the package should be created in the current directory So, out package pack with Addition class is ready

Program 2: Write a program to use the Addition class of package pack

```
//Using the package pack
import       packAddition;
class Use

{public static void main(String args[])
        {Addition   ob1   =   new   Addition(10,20);
                ob1sum();
        }
}
```

Output:

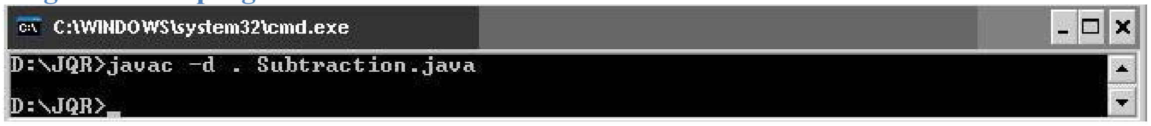


```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Use.java

D:\JQR>java  Use
Sum of two given numbers is : 30.0

D:\JQR>
```

Program 3: Write a program to add one more class Subtraction to the same package pack

```
//Adding  one  more  class  to  package  pack:
package pack;
public class Subtraction
{private double d1,d2;
        public Subtraction(double a, double b)
        {d1 = a; d2 =
                b;
        }
        public void difference()
        {System.out.println ("Sum of two given numbers is : " + (d1 - d2) );
        }
}
```

Compiling the above program:



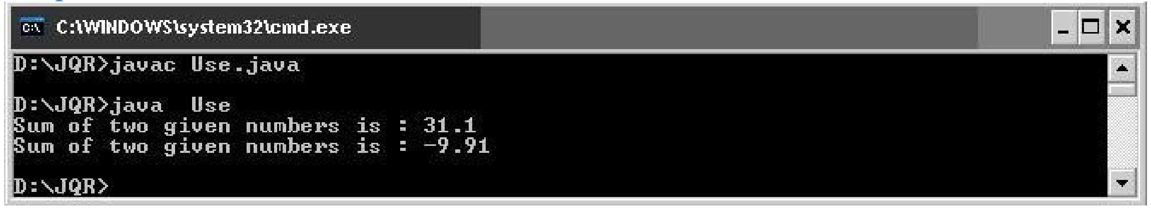
```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac -d . Subtraction.java

D:\JQR>_
```

Program 4: Write a program to access all the classes in the package pack

```
//To import all the classes and interfaces in a class using import pack*;
import  pack*;
class Use
{public static void main(String args[])
        {Addition   ob1   =   new   Addition(105,206);
                ob1sum();
                Subtraction        ob2        =        new
                Subtraction(302,4011); ob2difference();
        }
}
```

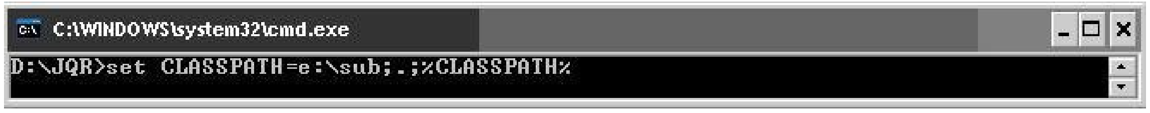
In this case, please be sure that any of the Additionjava and Subtractionjava programs will not exist in the current directory Delete them from the current directory as they cause confusion for the Java compiler The compiler looks for byte code in Additionjava and Subtractionjava files and there it gets no byte code and hence it flags some errors

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac Use.java

D:\JQR>java  Use
Sum of two given numbers is : 31.1
Sum of two given numbers is : -9.91

D:\JQR>
```

If the package pack is available in different directory, in that case the compiler should be given information regarding the package location by mentioning the directory name of the package in the classpath The CLASSPATH is an environment variable that tells the Java compiler where to look for class files to import If our package exists in e:\sub then we need to set class path as follows:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>set CLASSPATH=e:\sub;.;%CLASSPATH%
```

We are setting the classpath to **e:\sub** directory and current directory () and %CLASSPATH% means retain the already available classpath as it is

Creating Sub package in a package: We can create sub package in a package in the format:

package packagenamesubpackagename;

eg: package pack1pack2;

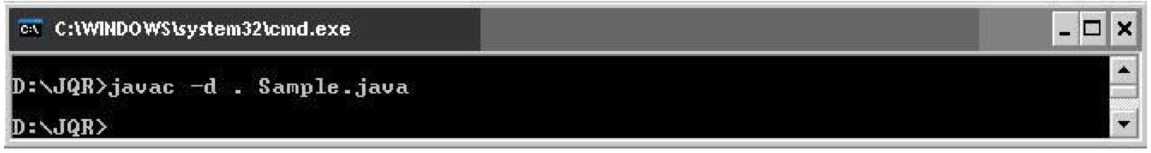
Here, we are creating pack2 subpackage which is created inside pack1 package To use the classes and interfaces of pack2, we can write import statement as:

import pack1pack2;

Program 5: Program to show how to create a subpackage in a package

```
//Creating a subpackage in a package
package pack1pack2;
public class Sample
{public void show ()
        {
                System.out.println ("Hello Java Learners");
        }
}
```

Compiling the above program:



```
C:\WINDOWS\system32\cmd.exe

D:\JQR>javac -d . Sample.java

D:\JQR>
```

Access Specifier: Specifies the scope of the data members, class and methods

private members of the class are available with in the class only The scope of private members of the class is “CLASS SCOPE” public members of the class are available anywhere The scope of public members of the class is "GLOBAL SCOPE"

default members of the class are available with in the class, outside the class and in its sub class of same package It is not available outside the package So the scope of default members of the class is "PACKAGE SCOPE"

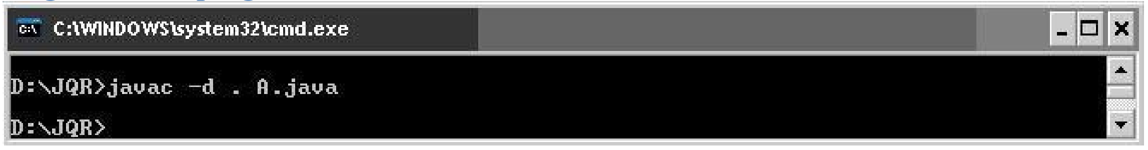
protected members of the class are available with in the class, outside the class and in its sub class of same package and also available to subclasses in different package also

Class Member Access	private	No Modifier	protected	public
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Program 6: Write a program to create class A with different access specifiers

```
//create a package same
package same;
public class A
{private     int     a=1;
        public int b =
        2;
        protected int c = 3;
        int d = 4;
}
```

Compiling the above program:

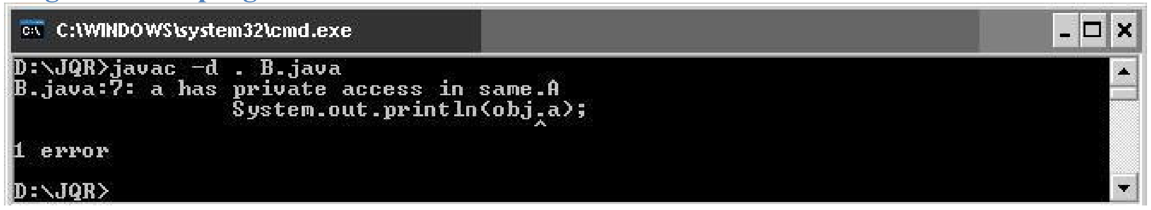


```
D:\JQR>javac -d . A.java
D:\JQR>
```

Program 7: Write a program for creating class B in the same package

```
//class B of same package
package same;
import sameA;
public class B
{
        public static void main(String args[])
        {
                A   obj   =   new   A();
                System.out.println(obja
                );
                System.out.println(objb
                );
                System.out.println(objc
                );
                System.out.println(objd
                );
        }
}
```

Compiling the above program:

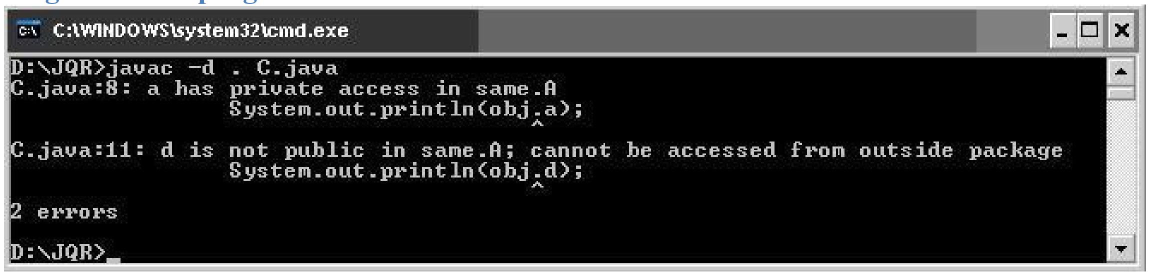


```
D:\JQR>javac -d . B.java
B.java:7: a has private access in same.A
                System.out.println(obj.a);
                                      ^

1 error

D:\JQR>
```

Program 8: Write a program for creating class C of another package

```
package another;
import sameA;
public class C extends A
{public static void main(String args[])
        {C      obj     =       new     C();
                System.out.println(obja);
                System.out.println(objb)
                ;
                System.out.println(objc);
                System.out.println(objd)
                ;
        }
}
```

Compiling the above program:



```
D:\JQR>javac -d . C.java
C.java:8: a has private access in same.A
                System.out.println(obj.a);
                                      ^

C.java:11: d is not public in same.A; cannot be accessed from outside package
                System.out.println(obj.d);
                                      ^

2 errors

D:\JQR>_
```

Inner Class

Inner Class: A class with in another class is called Inner class When the programmer wants to restrict the access of entire code of a class, creates an inner class as a private class The way to access the inner class is through its outer class only

· Inner class is a safety mechanism
· Inner class is hidden in outer class from other classes
· Only inner class can be private
· An object to Inner class can be created only in its outer class
· An object to Inner class cannot be created in any other class
· Outer class object and Inner class objects are created in separate memory locations
· Outer class members are available to Inner class object
· Inner class object will have an additional invisible field called „this$0" that stores a reference of outer class object
· Inner class members are referenced as: thismember;
· Outer class members are referred as: Outerclassthismember;

Interface

A programmer uses an abstract class when there are some common features shared by all the objects A programmer writes an interface when all the features have different implementations for different objects Interfaces are written when the programmer wants to leave the implementation to third party vendors An interface is a specification of method prototypes All the methods in an interface are abstract methods

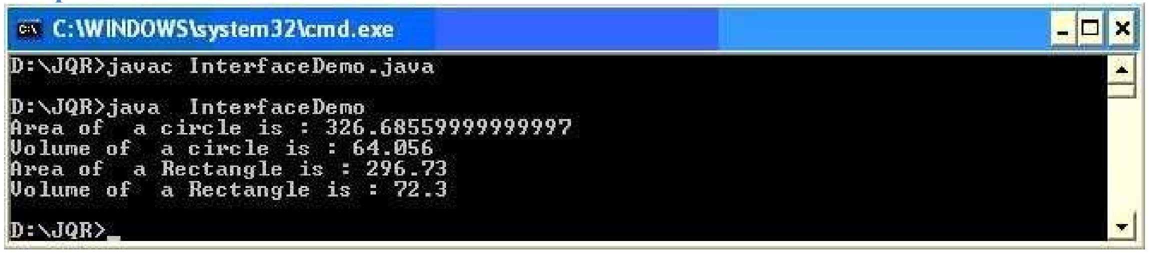
- An interface is a specification of method prototypes
- An interface contains zero or more abstract methods
- All the methods of interface are public, abstract by default
- An interface may contain variables which are by default public static final
- Once an interface is written any third party vendor can implement it
- All the methods of the interface should be implemented in its implementation classes
- If any one of the method is not implemented, then that implementation class should be declared as abstract
- We cannot create an object to an interface
- We can create a reference variable to an interface
- An interface cannot implement another interface
- An interface can extend another interface
- A class can implement multiple interfaces

Program 1: Write an example program for interface

```
interface Shape
{void   area   ();   void
        volume        ();
        double pi = 314;
}
class Circle implements Shape
{double r;
        Circle (double radius)
        {r = radius;
        }
        public void area ()
        {System.out.println ("Area of a circle is : " + pi*r*r );
        }
        public void volume ()
        {System.out.println ("Volume of a circle is : " + 2*pi*r);
        }
}
class Rectangle implements Shape
{double l,b;
        Rectangle (double length, double breadth)
        {l = length; b =
                breadth;.}
         public void area ()
         {System.out.println ("Area of a Rectangle is : " + l*b );
         }
```

```
            public void volume ()
            {System.out.println ("Volume of a Rectangle is : " + 2*(l+b));
            }
}
class InterfaceDemo
{public static void main (String args[])
            {Circle  ob1  =  new  Circle  (102);
                        ob1area ();
                        ob1volume ();
                        Rectangle ob2 = new Rectangle (126,
                        2355); ob2area ();
                        ob2volume ();
            }
}
```

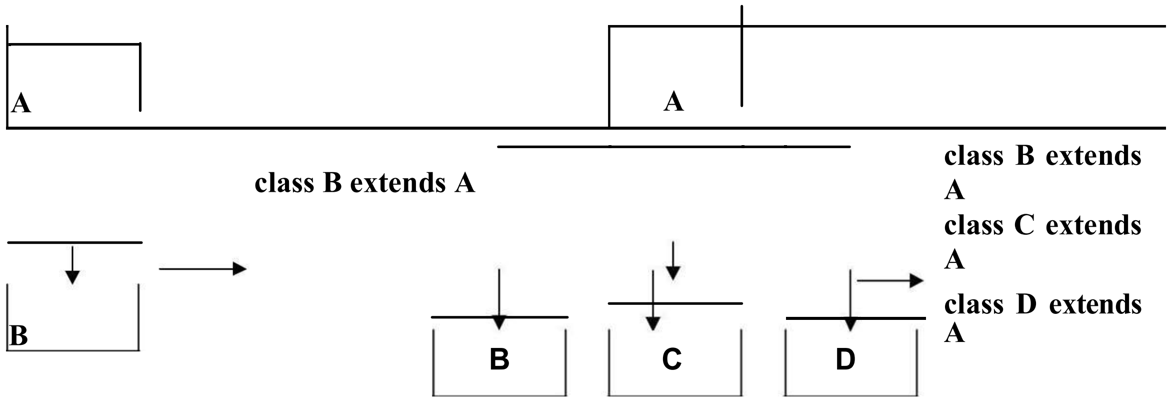
Output:



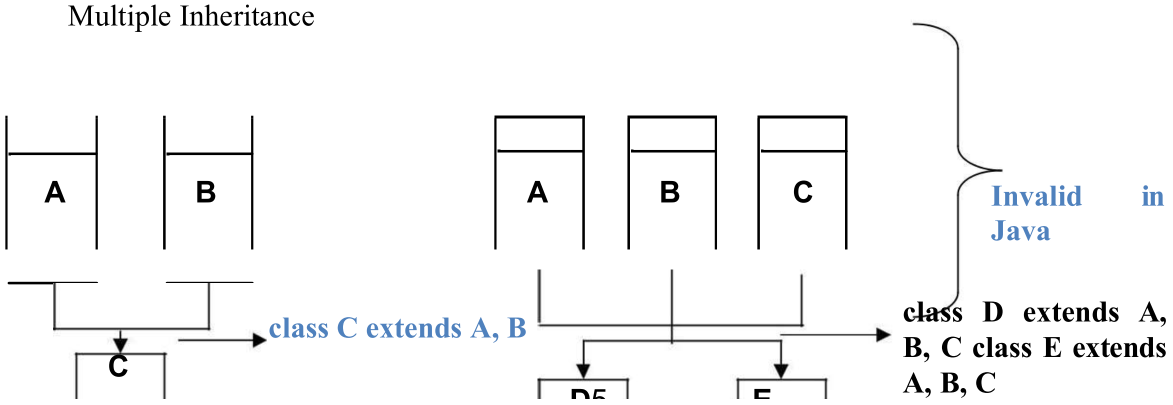
```
D:\JQR>javac InterfaceDemo.java

D:\JQR>java  InterfaceDemo
Area of  a circle is : 326.68559999999997
Volume of  a circle is : 64.056
Area of  a Rectangle is : 296.73
Volume of  a Rectangle is : 72.3

D:\JQR>_
```

Types of inheritance:

Single Inheritance: Producing subclass from a single super class is called single inheritance



Multiple Inheritance: Producing subclass from more than one super class is called Multiple Inheritance

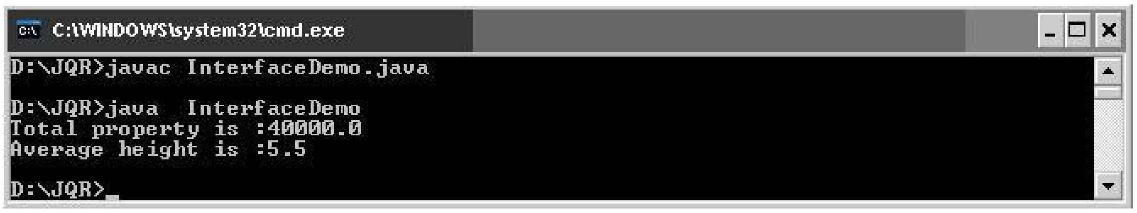


Java does not support multiple inheritance But multiple inheritance can be achieved by using interfaces

Program 2: Write a program to illustrate how to achieve multiple inheritance using multiple interfaces

```
        //interface Demo
        interface Father
    {double PROPERTY   =   10000;
       double HEIGHT = 56;
        }
        interface Mother
    {double PROPERTY   =   30000;
       double HEIGHT = 54;
        }
        class MyClass implements Father, Mother
        {void show()
              {   System.out.println("Total   property   is   :"   +(FatherPROPERTY+MotherPROPERTY));
                  System.out.println ("Average height is :" + (FatherHEIGHT + MotherHEIGHT)/2 );
              }
        }
        class InterfaceDemo
        {public static void main(String args[])
              {MyClass    ob1    =    new    MyClass();
                    ob1show();
              }
        }
```

Output:

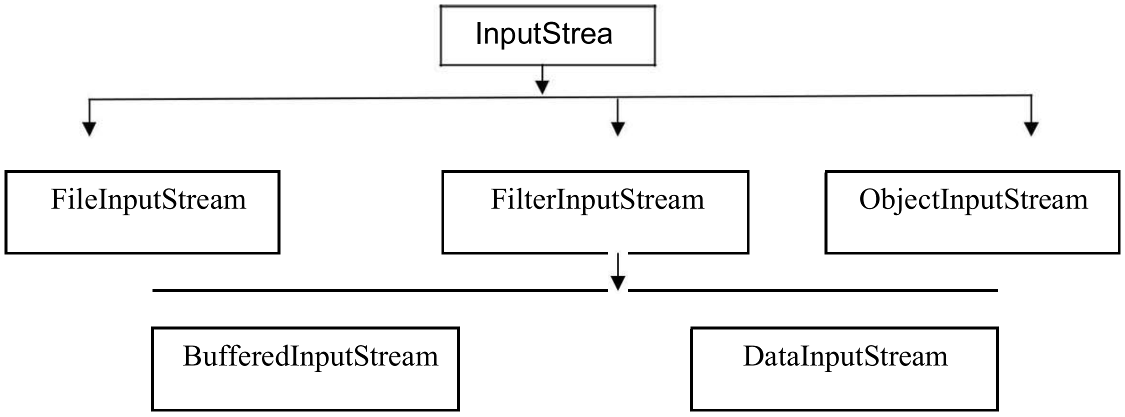


```
D:\JQR>javac InterfaceDemo.java

D:\JQR>java  InterfaceDemo
Total property is :40000.0
Average height is :5.5

D:\JQR>_
```

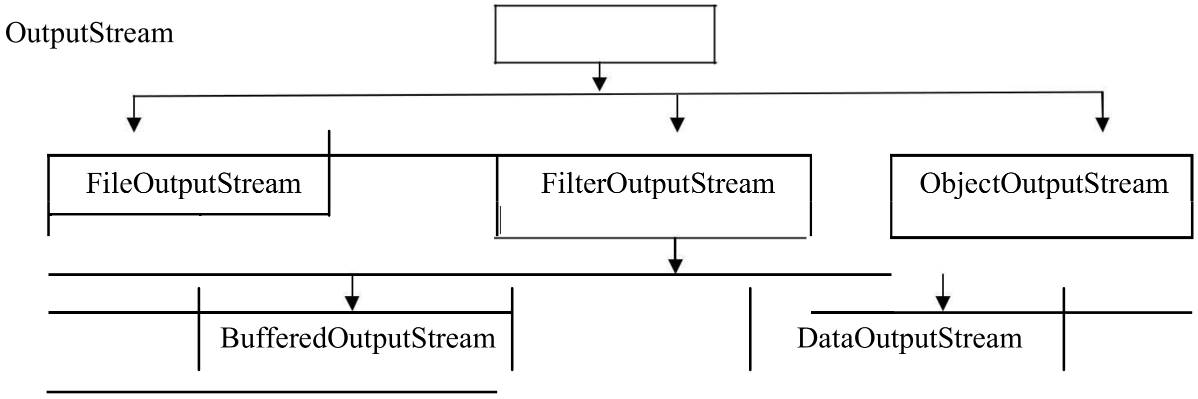
Java IO

A Stream represents flow of data from one place to another place Input Streams reads or accepts data Output Streams sends or writes data to some other place All streams are represented as classes in javaio package The main advantage of using stream concept is to achieve hardware independence This is because we need not change the stream in our program even though we change the hardware Streams are of two types in Java:

Byte Streams: Handle data in the form of bits and bytes Byte streams are used to handle any characters (text), images, audio and video files For example, to store an image file (gif or
jpg), we should go for a byte stream To handle data in the form of 'bytes' the abstract classes: InputStream and OutputStream are used The important classes of byte streams are:



Byte Stream Classes for Reading Data

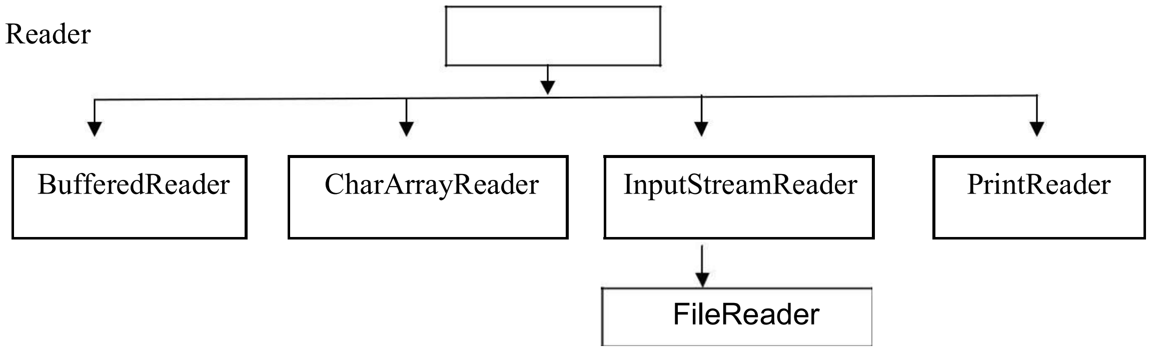


Byte Stream Classes for Writing Data

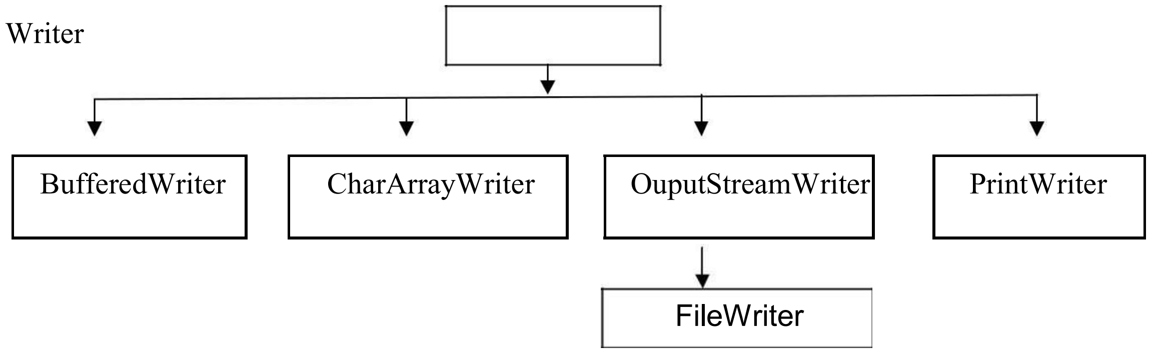
FileInputStream/FileOutputStream: They handle data to be read or written to disk files
FilterInputStream/FilterOutputStream: They read data from one stream and write it to another stream
ObjectInputStream/ObjectOutputStream: They handle storage of objects and primitive data

Character or Text Streams: Handle data in the form of characters Character or text streams can always store and retrieve data in the form of characters (or text) only It means text streams are more suitable for handling text files like the ones we create in Notepad They are not suitable to handle the images, audio or video files To handle data in the form of 'text'

the abstract classes: Reader and Writer are used The important classes of character streams are:



Text Stream Classes for Reading Data



Text Stream Classes for Writing Data

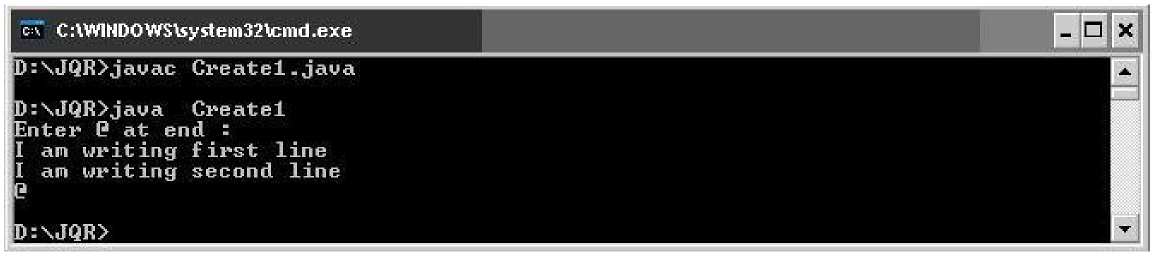
BufferedReader/BufferedWriter: - Handles characters (text) by buffering them They provide efficiency
CharArrayReader/CharArrayWriter: - Handles array of characters
InputStreamReader/OutputStreamWriter: - They are bridge between byte streams and character streams Reader reads bytes and then decodes them into 16-bit unicode characters Writer decodes characters into bytes and then writes
PrintReader/PrintWriter: - Handle printing of characters on the screen

File: A file represents organized collection of data Data is stored permanently in the file Once data is stored in the form of a file we can use it in different programs

Program 1: Write a program to read data from the keyboard and write it to a text file using byte stream classes

```
//Creating a text file using byte stream classes
import javaio*;
class Create1
{public static void main(String args[]) throws IOException
        {//attach keyboard to DataInputStream DataInputStream dis =
                new DataInputStream (Systemin); //attach the file to
                FileOutputStream FileOutputStream fout = new
                FileOutputStream ("myfile");
                //read data from DataInputStream and write into
                FileOutputStream char ch;
                System.out.println ("Enter @ at end : " )
                ; while( (ch = (char) disread() ) != '@' )
                        foutwrite (ch);
                foutclose ();
        }
}
```

Output:

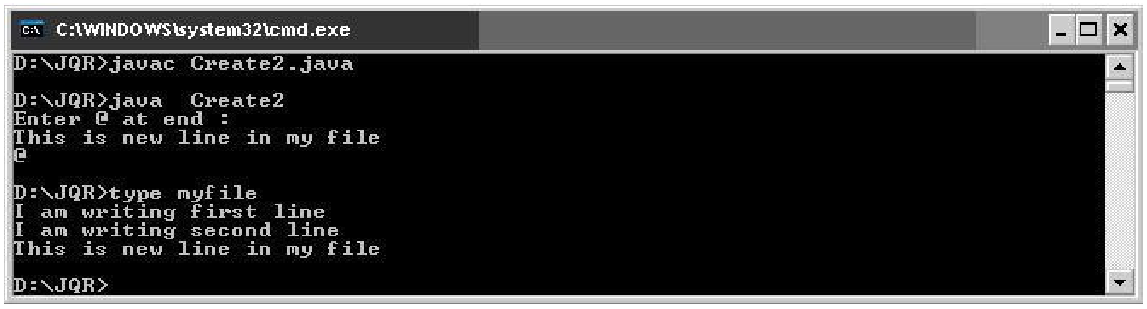


```
D:\JQR>javac Create1.java

D:\JQR>java  Create1
Enter @ at end :
I am writing first line
I am writing second line
@

D:\JQR>
```

Program 2: Write a program to improve the efficiency of writing data into a file using BufferedOutputStream

```
//Creating a text file using byte stream classes
import javaio*;
class Create2
{public static void main(String args[]) throws IOException
        {//attach keyboard to DataInputStream DataInputStream dis =
                new DataInputStream (Systemin);
                //attach file to FileOutputStream, if we use true then it will open in append
                mode FileOutputStream fout = new FileOutputStream ("myfile", true);
                BufferedOutputStream bout = new BufferedOutputStream (fout, 1024); //Buffer
                size is declared as 1024 otherwise default buffer size of 512 bytes is used //read
                data from DataInputStream and write into FileOutputStream
                char ch;
                System.out.println ("Enter @ at end : " )
                ; while ( (ch = (char) disread() ) != '@' )
                        boutwrite (ch);
                boutclose        ();
                foutclose ();
        }
}
```

Output:

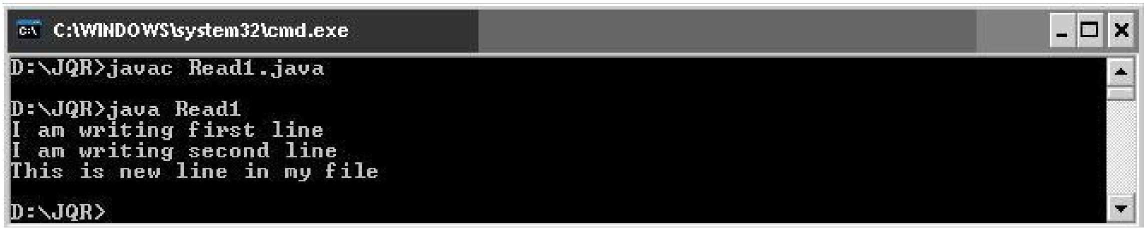


```
D:\JQR>javac Create2.java

D:\JQR>java  Create2
Enter @ at end :
This is new line in my file
@

D:\JQR>type myfile
I am writing first line
I am writing second line
This is new line in my file

D:\JQR>
```

Program 3: Write a program to read data from *myfile* using FileInputStream

```
//Reading a text file using byte stream classes
import javaio*;
class Read1
{public static void main (String args[]) throws IOException
        {//attach the file to FileInputStream FileInputStream fin =
                new FileInputStream ("myfile");
                //read data from FileInputStream and display it on the
                monitor int ch;
                while ( (ch = finread() ) != -1 )
                        System.out.print ((char) ch);
                finclose ();
        }
}
```

Output:

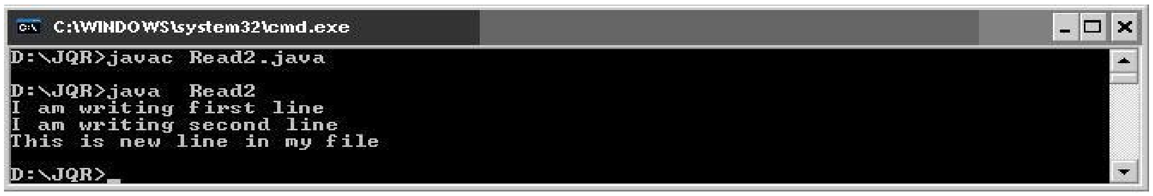


```
D:\JQR>javac Read1.java

D:\JQR>java Read1
I am writing first line
I am writing second line
This is new line in my file

D:\JQR>
```

Program 4: Write a program to improve the efficiency while reading data from a file using BufferedInputStream

```
//Reading a text file using byte stream classes
import javaio*;
class Read2
{public static void main(String args[]) throws IOException
        {//attach the file to FileInputStream FileInputStream fin = new
                FileInputStream ("myfile"); BufferedInputStream bin = new
                BufferedInputStream (fin); //read data from FileInputStream
                and display it on the monitor int ch;

                while ( (ch = binread() ) != -1 )
                        System.out.print ( (char)
                        ch);
                finclose ();
        }
}
```

Output:

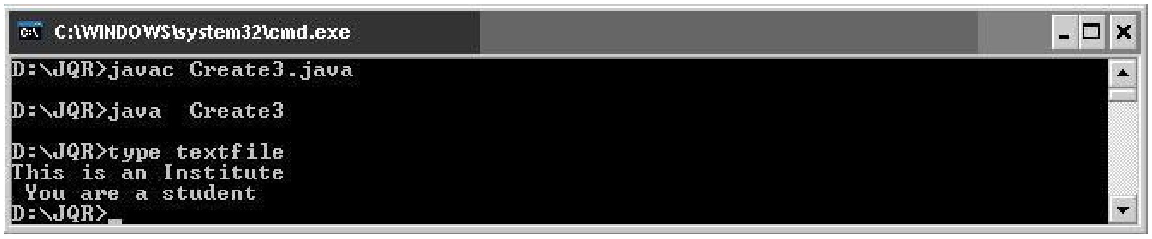


```
D:\JQR>javac Read2.java

D:\JQR>java  Read2
I am writing first line
I am writing second line
This is new line in my file

D:\JQR>_
```

Program 5: Write a program to create a text file using character or text stream classes //Creating a text file using character (text) stream classes import javaio*;

```
class Create3
{public static void main(String args[]) throws IOException
      {       String str = "This is an Institute" + "\n You are a student";        // take a String
              //Connect a file to FileWriter
              FileWriter fw = new FileWriter ("textfile");
              //read chars from str and send to fw
              for (int i = 0; i<strlength () ; i++)
                      fwwrite   (strcharAt   (i)   );
              fwclose ();
      }
}
```

Output:

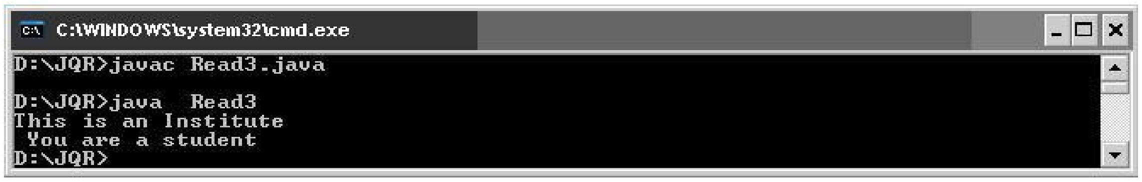


```
D:\JQR>javac Create3.java

D:\JQR>java  Create3

D:\JQR>type textfile
This is an Institute
 You are a student
D:\JQR>_
```

Program 6: Write a program to read a text file using character or text stream classes //Reading data from file using character (text) stream classes import javaio*;

```
class Read3
{public static void main(String args[]) throws IOException
      {//attach file to FileReader
              FileReader    fr    =    new    FileReader
              ("textfile"); //read data from fr and display
              int ch;
              while   ((ch   =   frread())   !=   -1)
                      System.out.print     ((char)
                      ch);
              //close      the
              file frclose ();
      }
}
```

Output:



```
D:\JQR>javac Read3.java

D:\JQR>java  Read3
This is an Institute
 You are a student
D:\JQR>
```

Note: Use BufferedReader and BufferedWriter to improve the efficiency of the above two programs

Serialization of objects:

Serialization is the process of storing object contents into a file The class whose objects are stored in the file should implement "Serializable' interface of javaio package Serializable interface is an empty interface without any members and methods, such an interface is called 'marking interface' or 'tagging interface'

Marking interface is useful to mark the objects of a class for a special purpose For example, 'Serializable' interface marks the class objects as 'serializable' so that they can be written into a file If serializable interface is not implemented by the class, then writing that class objects into a file will lead to NotSerializableException

static and transient variables cannot be serialized

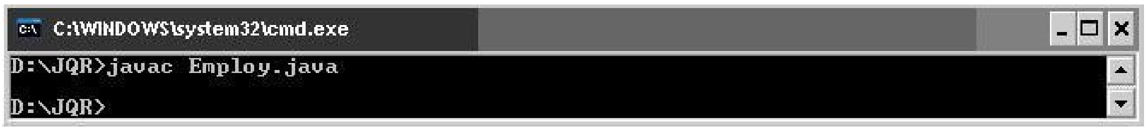
De-serialization is the process of reading back the objects from a file

Program 7: Write a program to create Employ class whose objects is to be stored into a file

```
//Employ information
import        javaio*;
import javautil*;
class Employ implements Serializable
{private   int   id;   private
        String          name;
        private    float    sal;
        private Date doj;
        Employ (int i, String n, float s, Date d)
        {id = i; name =
                n; sal = s;
                doj = d;
        }
        void display ()
        {
                System.out.println (id+ "\t" + name + "\t" + sal + "\t" + doj);
        }
        static Employ getData() throws IOException
        {BufferedReader br = new BufferedReader (new InputStreamReader (Systemin));
                System.out.print ("Enter employ id : ");
                int   id   =   IntegerparseInt(brreadLine());
                System.out.print  ("Enter  employ  name  :
                ");   String   name   =   brreadLine   ();
                System.out.print ("Enter employ salary : "
                );  float  sal  =  FloatparseFloat(brreadLine
                ());
                Date d = new Date ();
                Employ e = new Employ (id, name, sal, d);
                return e;
        }
}
```

. .

Output:

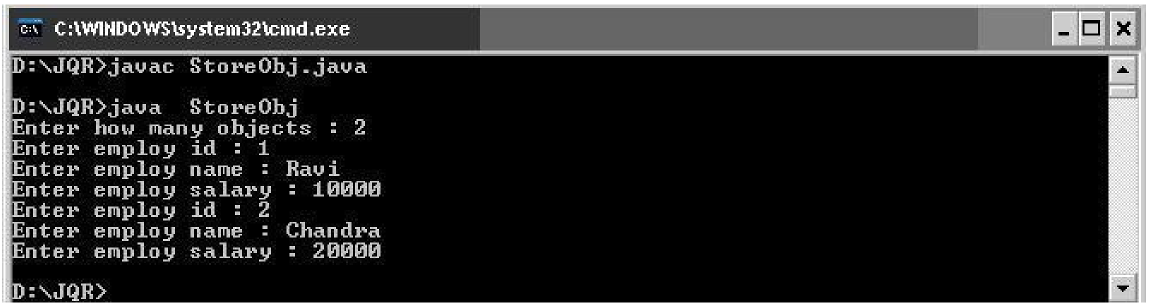


```
D:\JQR>javac Employ.java

D:\JQR>
```

Program 8: Write a program to show serialization of objects

```
//ObjectOutputStream is used to store objects to a file
import javaio*;
import javautil*;
class StoreObj
{public static void main (String args[]) throws IOException
        {BufferedReader br = new BufferedReader (new InputStreamReader (Systemin));
                FileOutputStream fos = new FileOutputStream ("objfile");
                ObjectOutputStream oos = new ObjectOutputStream ( fos ); System.out.print
                ("Enter how many objects : ");
                int n = IntegerparseInt(brreadLine () );
                for(int i = 0;i<n;i++)
                {Employ e1 = EmploygetData ();
                        ooswriteObject (e1);
                }
                oosclose ();
                fosclose ();
        }
}
```

Output:



```
D:\JQR>javac StoreObj.java

D:\JQR>java  StoreObj
Enter how many objects : 2
Enter employ id : 1
Enter employ name : Ravi
Enter employ salary : 10000
Enter employ id : 2
Enter employ name : Chandra
Enter employ salary : 20000

D:\JQR>
```

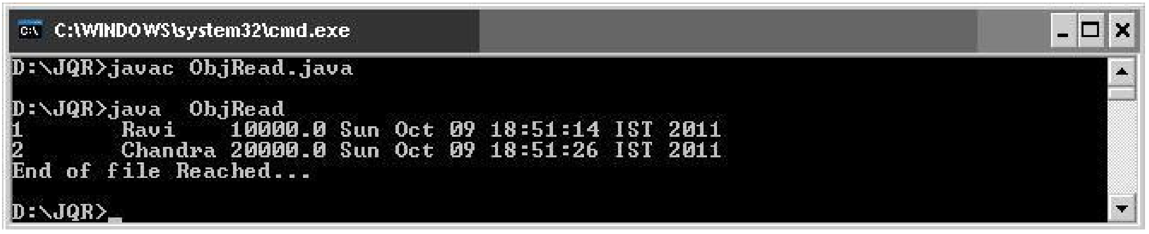
Program 9: Write a program showing deserialization of objects

```
//ObjectInputStream is used to read objects from a file
import javaio*;
class ObjRead
{public static void main(String args[]) throws Exception
        {
                FileInputStream fis = new FileInputStream
                ("objfile"); ObjectInputStream ois = new
                ObjectInputStream (fis); try
                {Employ e;
```

.   .

```
                        while ( (e = (Employ) oisreadObject() ) != null)
                                 edisplay ();
                }
                catch(EOFException ee)
                {
                        System.out.println ("End of file Reached");
                }
                finally
                {oisclose          ();
                        fisclose ();
                }
        }
}
```

Output:



```
D:\JQR>javac ObjRead.java

D:\JQR>java  ObjRead
1       Ravi    10000.0 Sun Oct 09 18:51:14 IST 2011
2       Chandra 20000.0 Sun Oct 09 18:51:26 IST 2011
End of file Reached...

D:\JQR>_
```

File Class: File class of javaio package provides some methods to know the properties of a file or a directory We can create the File class object by passing the filename or directory name to it

```
File obj = new File (filename);
File  obj  =  new  File  (directoryname);
File obj = new File ("path", filename);
File obj = new File ("path", directoryname);
```

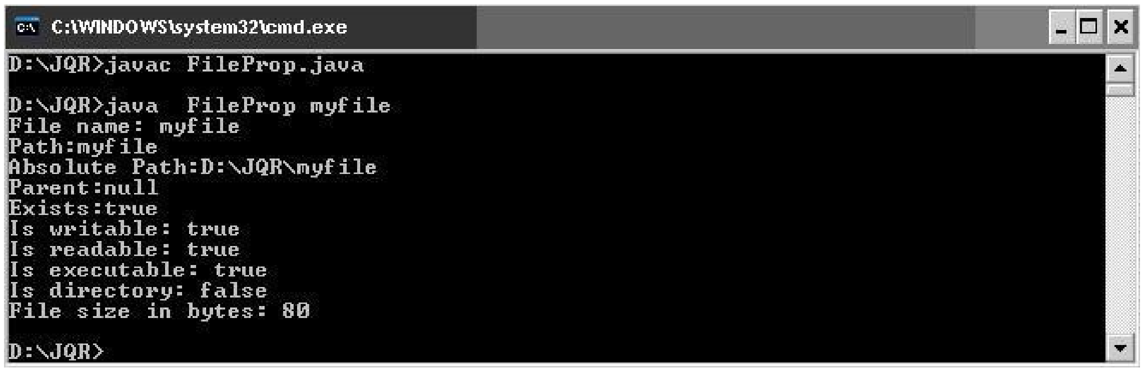
File class Methods:

Methods	Description
boolean isFile ()	Returns true if the File object contains a filename, otherwise false
boolean isDirectory ()	Returns true if the File object contains a directory name
boolean canRead ()	Returns true if the File object contains a file which is readable
boolean canWrite ()	Returns true if the File object contains a file which is writable
boolean canExecute ()	Returns true if the File object contains a file which is executable
Boolean exists ()	Returns true when the File object contains a file or directory which physically exists in the computer
String getParent ()	Returns the name of the parent directory
String getPath ()	Gives the name of directory path of a file or directory
String getAbsolutePath ()	Gives the fully qualified path
long length ()	Returns a number that represents the size of the file in bytes
boolean delete ()	Deletes the file or directory whose name is in File object
boolean createNewFile ()	Automatically crates a new, empty file indicated by File object, if and only if a file with this name does not yet exist

Program 10: Write a program that uses File class methods

```
//Displaying file properties
import javaio*;
class FileProp
{public static void main(String args[])
        {String fname = args [0]; File f
                = new File (fname);
                System.out.println  ("File  name:  "  +  fgetname  ());
                System.out.println        ("Path:"+        fgetPath        ());
                System.out.println  ("Absolute  Path:"+  fgetAbsolutePath
                ());   System.out.println   ("Parent:"+   fgetParent   ());
                System.out.println ("Exists:"+ fexists ());
                if ( fexists() )
                {System.out.println  ("Is   writable:   "+   fcanWrite   ());
                        System.out.println  ("Is  readable:  "+  fcanRead  ());
                        System.out.println  ("Is  executable:  "+  fcanExecute
                        ()); System.out.println ("Is directory: "+ fisDirectory
                        ()); System.out.println ("File size in bytes: "+ flength
                        ());
                }
        }
}
```

Ouput:



```
D:\JQR>javac FileProp.java

D:\JQR>java  FileProp myfile
File name: myfile
Path:myfile
Absolute Path:D:\JQR\myfile
Parent:null
Exists:true
Is writable: true
Is readable: true
Is executable: true
Is directory: false
File size in bytes: 80

D:\JQR>
```

UNIT - III

Exception handling - Fundamentals of exception handling, Exception types, Termination or resumptive models, Uncaught exceptions, using try and catch, multiple catch clauses, nested try statements, throw, throws and finally, built- in exceptions, creating own exception sub classes.
Multithreading- Differences between thread-based multitasking and process-based multitasking, Java thread model, creating threads, thread priorities, synchronizing threads, inter thread communication.

EXCEPTIONS

An error in a program is called bug Removing errors from program is called debugging There are basically three types of errors in the Java program:

Compile time errors: Errors which occur due to syntax or format is called compile time errors These errors are detected by java compiler at compilation time Desk checking is solution for compile-time errors

Runtime errors: These are the errors that represent computer inefficiency Insufficient memory to store data or inability of the microprocessor to execute some statement is examples to runtime errors Runtime errors are detected by JVM at runtime

Logical errors: These are the errors that occur due to bad logic in the program These errors are rectified by comparing the outputs of the program manually

Exception: An abnormal event in a program is called Exception

Exception may occur at compile time or at runtime
Exceptions which occur at compile time are called **Checked exceptions**
eg: ClassNotFoundException, NoSuchMethodException, NoSuchFieldException etc
Exceptions which occur at run time are called **Unchecked exceptions**
eg: ArrayIndexOutOfBoundsException, ArithmeticException, NumberFormatException etc

Exception Handling: Exceptions are represented as classes in java

An exception can be handled by the programmer where as an error cannot be handled by the programmer When there is an exception the programmer should do the following tasks:

If the programmer suspects any exception in program statements, he should write them inside try block

```
try
{
        statements;
}
```

When there is an exception in try block JVM will not terminate the program abnormally
JVM stores exception details in an exception stack and then JVM jumps into catch block
The programmer should display exception details and any message to the user in catch block

```
catch ( ExceptionClass obj)
{statements;
}
```

Programmer should close all the files and databases by writing them inside finally block Finally block is executed whether there is an exception or not

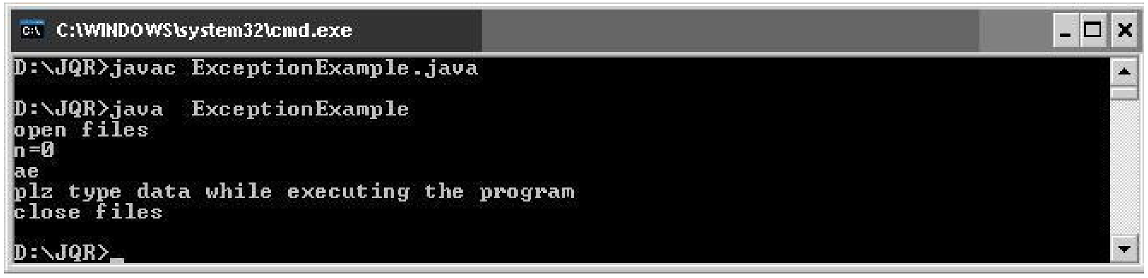
```
finally
{           statements;
}
```

Performing above tasks is called Exception Handling

Program 1: Write a program which tells the use of try, catch and finally block

```
  Exception    example
class
ExceptionExample
{public static void main(String args[])
        {try
                {System.out.println  ("open  files");  int
                        n=argslength;
                        System.out.println ("n="+n); int
                        a=45/n;         System.out.println
                        ("a="+a);
                        int b[]={10,19,12,13};
                        b[50]=100;
                }
                catch (ArithmeticException ae)
                {System.out.println ("ae");
                        System.out.println ("plz type data while executing the program");
                }
                catch (ArrayIndexOutOfBoundsException aie)
                {System.out.println ("aie");
                        System.out.println ("please see that array index is not within the range");
                }
                finally
                {           System.out.println ("close files");
                }
        }
}
```

Output:



```
D:\JQR>javac ExceptionExample.java

D:\JQR>java  ExceptionExample
open files
n=0
ae
plz type data while executing the program
close files

D:\JQR>_
```

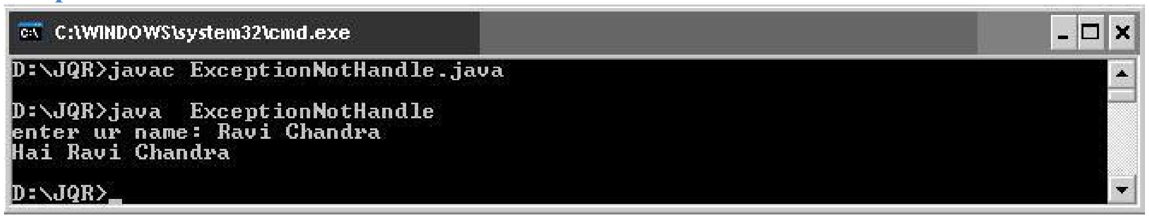
.Even though multiple exceptions are found in the program, only one exception is raised at a time.We can handle multiple exceptions by writing multiple catch blocks.A single try block can be followed by several catch blocks.Catch block does not always exit without a try, but a try block exit without a catch block.Finally block is always executed whether there is an exception or not

throws Clause: throws clause is useful to escape from handling an exception throws clause is useful to throw out any exception without handling it

Program 2: Write a program which shows the use of throws clause

```
  not      handling      the
exception import javaio*;
class Sample
{void accept( )throws IOException
        {BufferedReader br=new BufferedReader (new InputStreamReader(Systemin));
                System.out.print ("enter ur name: ");
                String     name=brreadLine     (     );
                System.out.println ("Hai "+name);
        }
}
class ExceptionNotHandle
{public static void main (String args[])throws IOException
        {Sample  s=new  Sample  (  );
                saccept ( );
        }
}
```

Output:



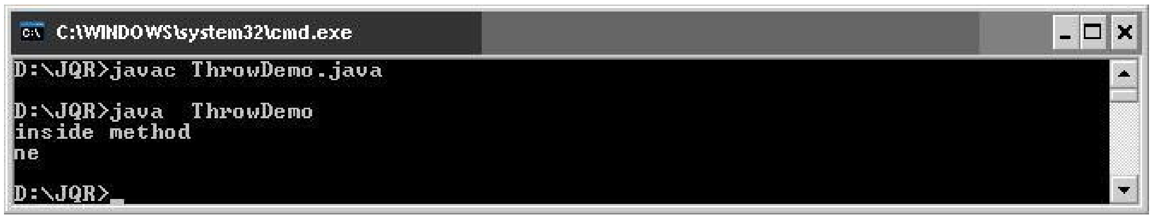
```
D:\JQR>javac ExceptionNotHandle.java

D:\JQR>java  ExceptionNotHandle
enter ur name: Ravi Chandra
Hai Ravi Chandra

D:\JQR>_
```

throw Clause: throw clause can be used to throw out user defined exceptions It is useful to create an exception object and throw it out of try block

Program 3: Write a program which shows the use of throw clause

```
//Throw  Example
class ThrowDemo
{static void Demo( )
        {try
                {System.out.println    ("inside    method");    throw    new
                        NullPointerException("my data");

                }
                catch (NullPointerException ne)
                {
                        System.out.println ("ne");
                }
        }
        public static void main(String args[])
        {
                ThrowDemoDemo ( );
        }
}
```

Output:



```
D:\JQR>javac ThrowDemo.java

D:\JQR>java  ThrowDemo
inside method
ne

D:\JQR>_
```

Types of Exceptions:

Built-in exceptions: These are the exceptions which are already available in java
eg: ArithmeticException, ArrayIndexOutOfBoundsException, NullPointerException, StringIndexOutOfBoundsException, NoSuchMethodException, InterruptedException, ClassNotFoundException, FileNotFoundException, NumberFormatException, RuntimeException etc

User-defined exceptions: - These are the exceptions created by the programmer

Write user exception class extending Exception class **eg:** class MyException extends Exception
Write a default constructor in the user exception class
eg: MyException ( ) {   }

o Write a parameterized constructor with String as a parameter, from there call the parameterized constructor of Exception class

eg: MyException (String str)
```
        {
                super (str);
        }
```
Whenever required create user exception object and throw it using throw statement Ex: - throw me;

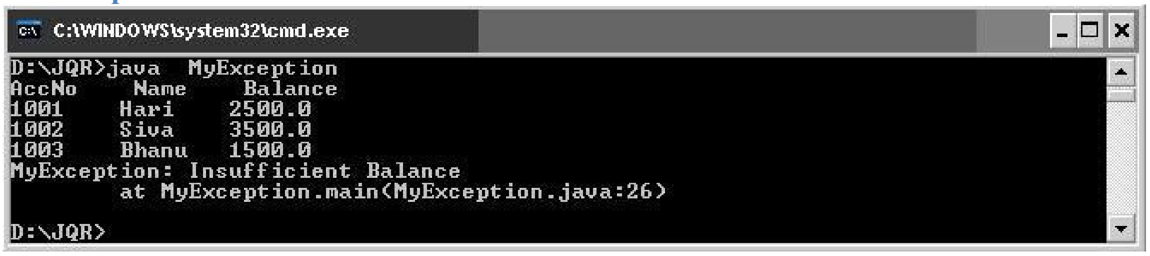
Program 4: Write a program to throw a user defined exception

```
// user defined exception
class MyException extends Exception
{
        int accno[] = {1001,1002,1003,1004,1005};

        String  name[]  =  {"Hari","Siva","Bhanu","Rama","Chandu"};
        double bal[] = {2500,3500,1500,1000,6000};
        MyException()
        {
        }
        MyException(String str)
        {
                super(str);
        }
```

```
		public static void main(String args[])
		{
			try
			{
				MyException   me   =   new   MyException("");
				System.out.println("AccNo  \t  Name  \t  Balance  ");
				for(int i=0;i<5;i++)
				{
					System.out.println(meaccno[i]+ "\t" + mename[i] + "\t" +
										mebal[i] );
					if( mebal[i] < 2000 )
					{
						MyException me1 = new MyException
								("Insufficient Balance");
throw me1;
					}
				}
			}
```

```
                        catch(MyException  e)
                        {
                                eprintStackTrace();
                        }
                }
        }
```

Output:



```
D:\JQR>java  MyException
AccNo     Name     Balance
1001     Hari     2500.0
1002     Siva     3500.0
1003     Bhanu    1500.0
MyException: Insufficient Balance
        at MyException.main(MyException.java:26)

D:\JQR>
```

Threads

Executing the tasks is of two types:

Single Tasking: Executing only one task at a time is called single tasking In this single tasking the microprocessor will be sitting idle for most of the time This means micro processor time is wasted

Multi tasking: Executing more than one task at a time is called multi tasking Multitasking is of two types:

Process Based Multitasking: Executing several programs simultaneously is called process based multi tasking

Thread Based Multitasking: Executing different parts of the same program simultaneously with the help of a thread is called thread based multitasking

Advantage of multitasking is utilizing the processor time in an optimum way

Java provides built-in support for multithreaded programming A multithreaded program contains two or more parts that can run concurrently Each part of such a program is called a thread Thread is a smallest unit of code Thread is also defined as a subprocess A Thread sometimes called an execution context or a light weight process

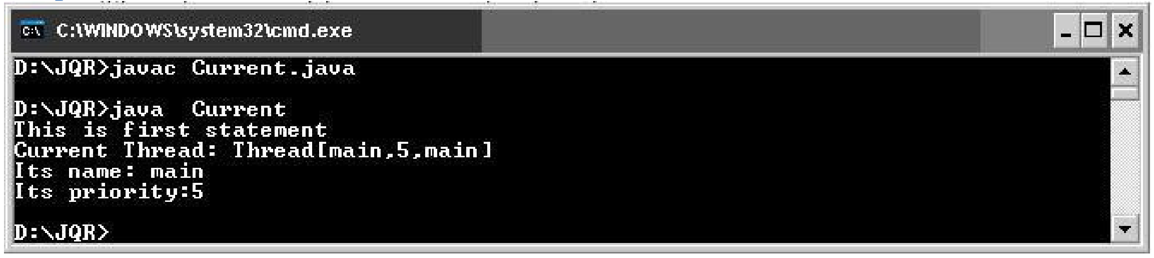
Uses of Threads:

Threads are used in designing serverside programs to handle multiple clients at a time
Threads are used in games and animations

Program 1: Write a program to know the currently running Thread

```
//Currently running thread
class Current
{
        public static void main(String args[])
        {
                System.out.println ("This is first statement");
                Thread t = ThreadcurrentThread ();
                System.out.println ("Current Thread: " + t);
                System.out.println ("Its name: " + tgetName ());
                System.out.println ("Its priority:" + tgetPriority
                ());
        }
}
```

Output:



```
D:\JQR>javac Current.java

D:\JQR>java  Current
This is first statement
Current Thread: Thread[main,5,main]
Its name: main
Its priority:5

D:\JQR>
```

Creating a Thread:

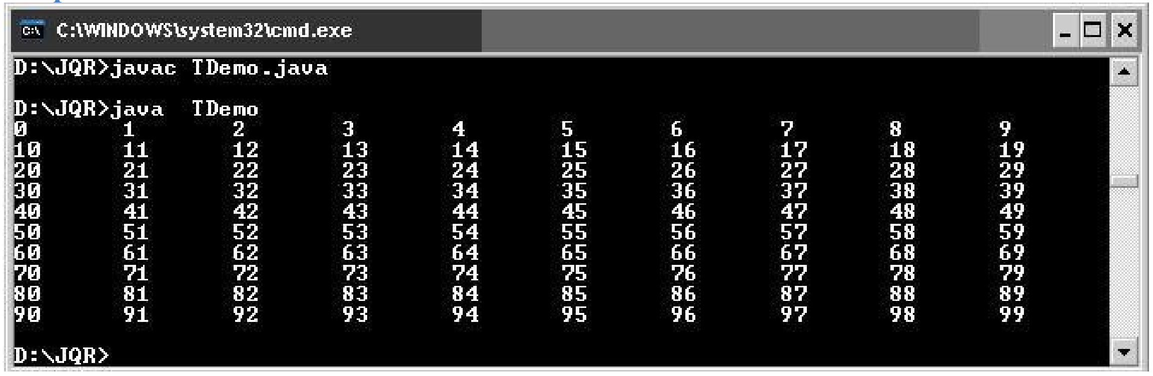
Write a class that extends Thread class or implements Runnable interface this is available in lang package
Write public void run () method in that class This is the method by default executed by any thread
Create an object to that class
Create a thread and attach it to the object
Start running the threads

Program 2: Write a program to create and run a Thread

```
//creating and running a Thread
class MyThread extends Thread
{public void run ()
        {for (int i = 0;i<100;i++)
                {
                        System.out.print (i + "\t");
                }
        }
}
```

```
class TDemo
{public static void main(String args[])
        {MyThread  obj  =  new  MyThread  ();
                Thread  t  =  new  Thread  (obj);
                tstart ();
        }
}
```

Output:



```
D:\JQR>javac TDemo.java

D:\JQR>java  TDemo
0       1       2       3       4       5       6       7       8       9
10      11      12      13      14      15      16      17      18      19
20      21      22      23      24      25      26      27      28      29
30      31      32      33      34      35      36      37      38      39
40      41      42      43      44      45      46      47      48      49
50      51      52      53      54      55      56      57      58      59
60      61      62      63      64      65      66      67      68      69
70      71      72      73      74      75      76      77      78      79
80      81      82      83      84      85      86      87      88      89
90      91      92      93      94      95      96      97      98      99

D:\JQR>
```

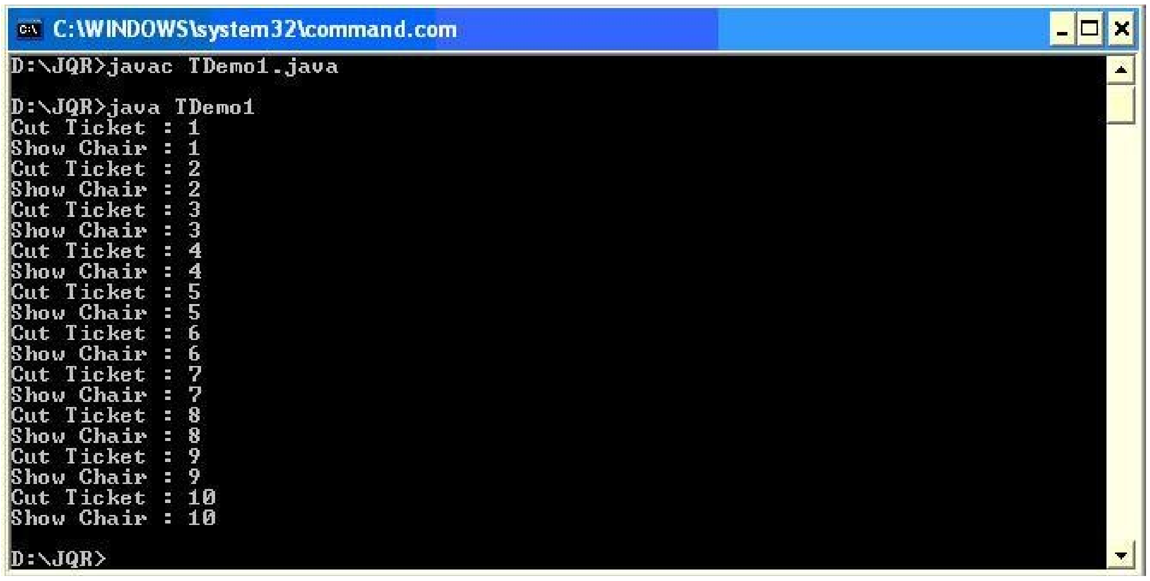
Multi Tasking Using Threads: In multi tasking, several tasks are executed at a time For this purpose, we need more than one thread For example, to perform 2 tasks we can take 2 threads and attach them to the 2 tasks Then those tasks are simultaneously executed by the two threads Using more than one thread is called „multi threading"

Program 3: Write a program to create more than one thread //using more than one thread is called Multi Threading class Theatre extends Thread

```
{String     str;     Theatre
        (String str)
        {thisstr = str;
        }
        public void run()
        {for (int i = 1; i <= 10 ; i++)
                {System.out.println (str + " : " + i); try
                        {Threadsleep (2000);
                        }
                        catch (InterruptedException ie) {     ieprintStackTrace ();     }
                }
        }
}
class TDemo1
{public static void main(String args[])
        {Theatre  obj1  =  new  Theatre  ("Cut  Ticket");
                Theatre  obj2  =  new  Theatre  ("Show
                Chair"); Thread t1 = new Thread (obj1);
                Thread  t2  =  new  Thread
                (obj2); t1start ();
                t2start ();
        }
```

}

Output:



C:\WINDOWS\system32\command.com

```
D:\JQR>javac TDemo1.java

D:\JQR>java TDemo1
Cut Ticket : 1
Show Chair : 1
Cut Ticket : 2
Show Chair : 2
Cut Ticket : 3
Show Chair : 3
Cut Ticket : 4
Show Chair : 4
Cut Ticket : 5
Show Chair : 5
Cut Ticket : 6
Show Chair : 6
Cut Ticket : 7
Show Chair : 7
Cut Ticket : 8
Show Chair : 8
Cut Ticket : 9
Show Chair : 9
Cut Ticket : 10
Show Chair : 10

D:\JQR>
```

In the preceding example, we have used 2 threads on the 2 objects of TDemo1 class First we have taken a String variable str in Theatre class Then we passed two strings- cut ticket and show chair into that variable from TDemo1 class When t1 start () is executed, it starts execution run () method code showing cut ticket Note that in run () method, we used: Thread sleep (2000) is a static method in Thread class, which is used to suspend execution of a thread for some specified milliseconds Since this method can throw InterruptedException, we caught it in catch block When Thread t1 is suspended immediately t2 start () will make the thread t2 to execute and when it encounters Threadsleep(2000), it will suspend for specified time meanwhile t1 will get executed respectively In this manner, both the threads are simultaneously executed

Multiple Threads Acting on Single Object: When two people (threads) want to perform same task then they need same object (run () method) to be executed each time Take the case of railway reservation Every day several people want reservation of a berth for them The procedure to reserve the berth is same for all the people So we need some object with same run () method to be executed repeatedly for all the people (threads)

Let us think that only one berth is available in a train and two passengers (threads) are asking for that berth in two different counters The clerks at different counters sent a request to the server to allot that berth to their passengers Let us see now to whom that berth is allotted

Program 4: Write a program to create multiple threads and make the threads to act on single object

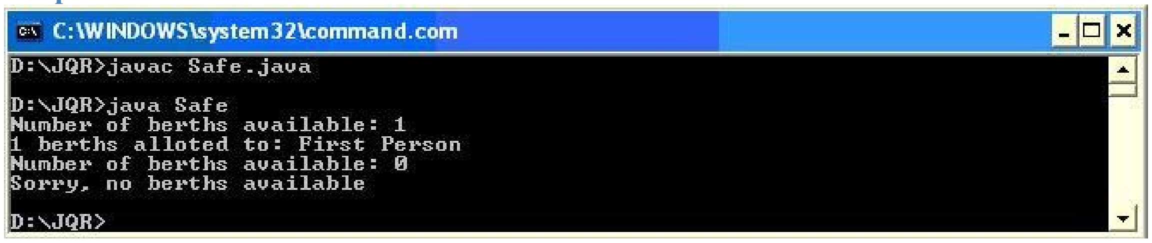
```
//Multiple  Threads  acting  on  single  object
class Reserve implements Runnable
{int  available  =  1;  int
        wanted;
        Reserve (int i)
        {wanted = i;
        }
        public void run()
        {synchronized (this)
```

```
                    {System.out.println ("Number of berths available: " + available); if (
                            available >= wanted)
                            {String    name    =    ThreadcurrentThread    ()getName    ();
                                    System.out.println  (wanted  +  "  berths  alloted  to:  "  +
                                    name); try
                                    {Threadsleep  (2000);  //  wait  for  priniting  the  ticket
                                            available = available - wanted;
                                    }
                                    catch (InterruptedException ie)
                                    {       ieprintStackTrace ();           }
                            }
                            else
                            {               System.out.println ("Sorry, no berths available");
                            }
                    }
            }
    }
```

.

```
class Safe
{public static void main(String args[])
        {Reserve  obj  =  new  Reserve  (1);
                Thread t1 =new Thread (obj);
                Thread t2 = new Thread (obj);
                t1setName   ("First   Person");
                t2setName ("Second Person");
                t1start ();
                t2start ();
        }
}
```

Output:



```
D:\JQR>javac Safe.java

D:\JQR>java Safe
Number of berths available: 1
1 berths alloted to: First Person
Number of berths available: 0
Sorry, no berths available

D:\JQR>
```

If we would not use synchronized (this) block in the preceding program then when thread t1 enter into the run () method, it sees available number of berths as 1 and hence it allots it to First Person and displays “1 Berths reserved for First Person” Then it enters try { } block inside run () method, where it will sleep for 2 seconds In this time, the ticket will be printed on the printer When the first thread is sleeping thread t2 also enters the run () method, it also sees that there is 1 berth remaining The reason is for this is that the available number of berths is not yet updated by the first thread So the second thread also sees 1 berth as available and it allots the same berth to the Second Person Then the thread t2 will also go into sleep state Thread t1 wakes up first and then it updates the available number of berths to zero (0) But at the same time the second thread has already allotted the same berth to the Second

Person also Since both the threads are acting on the same object simultaneously, the result will be unreliable

Thread Synchronization or Thread Safe: When a thread is acting on an object preventing other threads from acting on the same object is called Thread Synchronization or Thread Safe The Object on which the Threads are synchronized is called synchronized object or Mutex (Mutually Exclusive Lock) Thread synchronization is done in two ways:

Using synchronized block we can synchronize a block of statements **eg:** synchronized (obj)

```
{
        statements;
}
```

To synchronize an entire method code we can use synchronized word before method name **eg:** synchronized void method ()

```
{

}
```

Thread Creation: To create a Thread, we can use the following forms:

```
Thread t1 = new Thread ();
Thread t2 = new Thread (obj);
Thread t3 = new Thread (obj, "thread-name");
```

Thread Class Methods:

· To know the currently running thread:	Thread t = ThreadcurrentThread ();
· To start a thread:	tstart ();
· To stop execution of a thread for a specific time:	Threadsleep (milliseconds);
· To get the name of the thread:	String name = tgetName ();
· To set the new name to the thread:	tsetName ("New Name");
· To get the priority of the thread:	int priority = tgetPriority();
· To set the priority of the thread:	tsetPriority (int priority);

Thread priorities can change from 1 to 10 We can also use the following constants to represent

priorities: ThreadMAX_PRIORITY value is 10
ThreadMIN_PRIORITY value is 1
ThreadNORM_PRIORITY value is 5

· To test if a thread is still alive:	tisAlive () returns true/false
· To wait till a thread dies:	tjoin ();
· To send a notification to a waiting thread:	objnotify ();
· To send notification to all waiting threads:	objnotifyAll ();
To wait till the obj is released (till notification is sent):	objwait ();

Deadlock: When a Thread locked an object and waiting for another object to be released by another Thread, and the other thread is also waiting for the first thread to release the first object, both the threads will continue waiting forever This is called "Thread Deadlock"

Even if we synchronize the threads, there is possibility of other problems like deadlock Daily, thousands of people book tickets in trains and cancel tickets also If a programmer is to develop code for this, he may visualize that booking tickets and canceling them are reverse procedures Hence, he will write these 2 tasks as separate and opposite tasks and assign 2 different threads to do these tasks simultaneously

To book a ticket, the thread will enter the train object to verify that the ticket is available or not When there is a ticket, it updates the available number of tickets in the train object For this, it takes say 150 milli seconds Then it enters the compartment object In compartment object, it should allot the ticket for the passenger and update its status to reserved This means the thread should go through both the train and compartment objects Similarly, let us think if a thread has to cancel a ticket, it will first enter compartment object and updates the status of the ticket as available For this it is taking say 200 milliseconds Then it enters train object and updates the available number of tickets there So, this thread alsoshould fo through both the compartment and train objects

When the BookTicket thread is at train object for 150 milliseconds, the CancelTicket thread will be at compartment object for 200 milliseconds Because we are using multiple (more than one) threads, we should synchronize them So, the threads will lock those objects When 150 milliseconds time is over, BookTicket thread tries to come out of train object and wants to lock on compartment object, by entering it At that time, it will find that the compartment object

is already locked by another thread (CancelTicket) and hence it will wait BookTicket thread will wait for compartment object for another 50 milli seconds

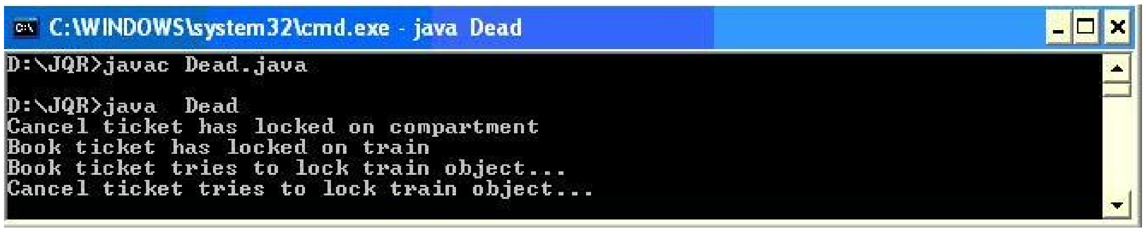
After 200 milliseconds time is up, the CancelTicket thread which is in compartment object completes its execution and wants to eneter and lock on train object But it will find that the train object is already under lock by BookTicket thread and hence is not available Now, CancelTicket will wait for the train object which should be unlocked by BookTicket

In this way, BookTicket thread keeps on waiting for the CancelTicket thread to unlock the compartment object and the CancelTicket thread keeps on waiting for the BookTicket to unlock the train object Both the threads will wait forever in this way, this situation is called DealLock

Program 5: Write a program to get a deadlock situation using threads

```
//to cancel the ticket
class CancelTicket extends Thread
{Object train, comp;
        CancelTicket (Object train, Object comp)
        {thistrain = train;
                thiscomp = comp;
        }
        public void run()
        {
                synchronized (comp)
                {System.out.println ("Cancel ticket has locked on compartment"); try
                        {
                                Threadsleep (2000);
                        }
                        catch (InterruptedException ie)        {    }
                        System.out.println ("Cancel   ticket   tries   to   lock   train
                        object"); synchronized (train)
                        {
                                System.out.println ("Cancel ticket has locked train");
                        }
                }
        }
}
//to book the ticket
class BookTicket extends Thread
{Object train, comp;

        {thistrain      =      train;
                thiscomp = comp;
        }
        public void run()
        {synchronized (train)
                {
  System.out.println ("Book ticket has locked on train");
.                                                                     .


                        try
                        {
                                Threadsleep (2000);
                        }
                        catch (InterruptedException ie)
                        {
                        }
                        System.out.println  ("Book   ticket   tries   to   lock   train
                        object"); synchronized (comp)
                        {
                                System.out.println ("Book ticket has locked compartment");
                        }
                }
        }
}
class Dead
{public static void main (String args[])
        {Object train = new Object ();
                Object compartment = new Object ();
                CancelTicket obj1 = new CancelTicket (train, compartment);
                BookTicket  obj2  =  new  BookTicket  (train,  compartment);
                Thread t1 = new Thread (obj1);
                Thread   t2   =   new   Thread   (obj2);
                t1start ();
                t2start ();
        }
}
```

Output:



C:\WINDOWS\system32\cmd.exe - java Dead

```
D:\JQR>javac Dead.java

D:\JQR>java  Dead
Cancel ticket has locked on compartment
Book ticket has locked on train
Book ticket tries to lock train object...
Cancel ticket tries to lock train object...
```

There is no specific solutioin for preventing deadlock The programmer should exercise proper caution while planning the logic of the program to avoid deadlocks

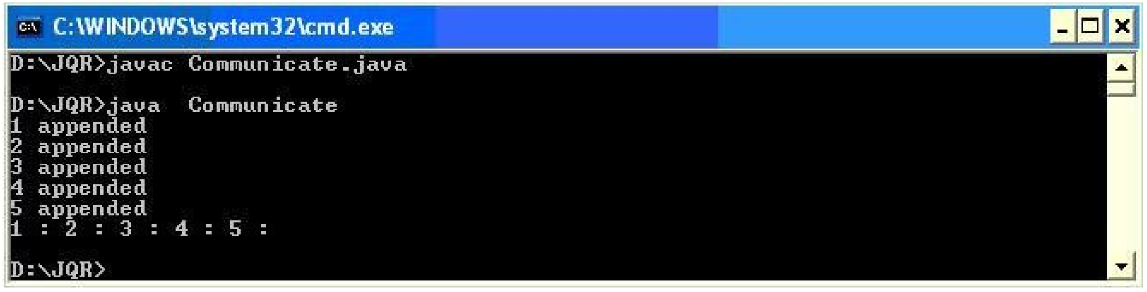
Thread Communication: In some cases two or more threads should communicate with each other One thread output may be send as input to other thread For example, a consumer thread is waiting for a Producer to produce the data (or some goods) When the Producer thread completes production of data, then the Consumer thread should take that data and use it
In producer class we take a StringBuffer object to store data, in this case; we take some numbers from 1 to 5 These numbers are added to StringBuffer object Until producer completes placing the data into StringBuffer the consumer has to wait Producer sends a notification immediately after the data production is over

Program 6: Write a program to demonstrate Thread communication //inter thread communication

```
class Producer implements Runnable
{StringBuffer       sb;
        Producer ()
        {
                sb = new StringBuffer();
        }
        public void run ()
        {synchronized (sb)
                {
                        for (int i=1;i<=5;i++)
                        {try
                                {sbappend  (i  +  "  :  ");  Threadsleep  (500);
                                        System.out.println (i + " appended");
                                }
                                catch (InterruptedException ie){}
                        }
                        sbnotify ();
                }
        }
}
```

```
class Consumer implements Runnable
{         Producer prod;
          Consumer (Producer prod)
          {
                    thisprod = prod;
          }
          public void run()
{       synchronized (prodsb)
        {       try
                {
                        prodsbwait ();
                        }

                        catch (Exception e)     {

                        System.out.println

                        (prodsb);

                        }
                        }

                        class Communicate

                        {public static void main(String args[])

                        {

                        Producer obj1 = new Producer (); Consumer obj2 = new
                        Consumer (obj1); Thread t1 = new Thread (obj1);

                        Thread t2 = new Thread (obj2);

                        t2start ();

                        t1start ();

                        }

                        }

                }
                catch (Exception e)   {
                System.out.println
                (prodsb);


        }
}
```

Output:



```
D:\JQR>javac Communicate.java

D:\JQR>java  Communicate
1 appended
2 appended
3 appended
4 appended
5 appended
1 : 2 : 3 : 4 : 5 :

D:\JQR>
```

Both sleep () and wait () methods are used to suspend a thread execution for a specified time When sleep () is executed inside a synchronized block, the object is still under lockWhen wait () method is executed, it breaks the synchronized block, so that the object lock is removed and it is available

Thread Group: A ThreadGroup represents a group of threads The main advantage of taking several threads as a group is that by using a single method, we will be able to control all the threads in the group

· Creating a thread group: ThreadGroup tg = new ThreadGroup (“groupname”);

· To add a thread to this group (tg): Thread t1 = new Thread (tg, targetobj, “threadname”);

To add another thread group to this group (tg):

ThreadGroup tg1 = new ThreadGroup (tg, “groupname”);

· To know the parent of a thread: tggetParent ();

· To know the parent thread group: tgetThreadGroup ();

This returns a ThreadGroup object to which the thread t belongs

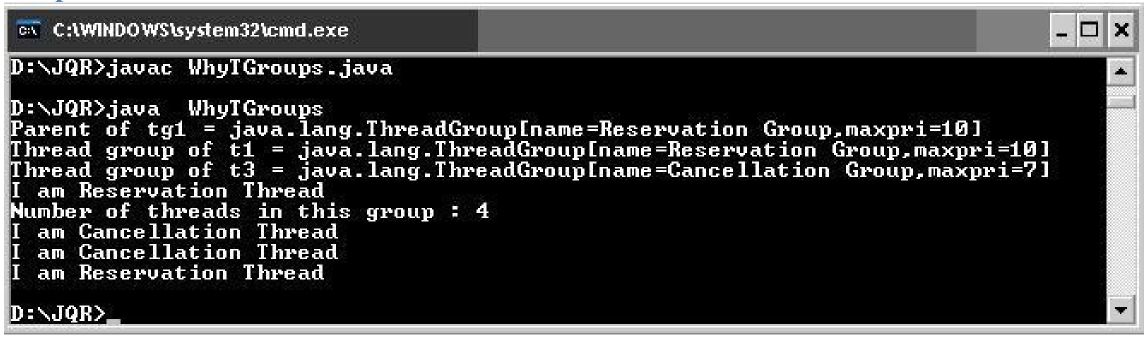
· To know the number of threads actively running in a thread group: tactiveCount ();

· To change the maximum priority of a thread group tg: tgsetMaxPriority ();

Program 7: Write a program to demonstrate the creation of thread group

```
//Using ThreadGroup
import javaio*;
class WhyTGroups
{public static void main (String args[]) throws IOException
        {Reservation   res   =   new   Reservation   ();
                Cancellation can = new Cancellation
                (); //Create a ThreadGroup
                ThreadGroup   tg   =   new   ThreadGroup   ("Reservation
                Group"); //Create 2 threads and add them to thread group
                Thread t1 = new Thread (tg, res, "First Thread");
                Thread  t2  =  new  Thread  (tg,  res,  "Second
                Thread");
                //Create another thread group as a child to tg
                ThreadGroup   tg1   =   new   ThreadGroup   (tg,   "Cancellation
                Group"); Thread t3 = new Thread (tg1, can, "Third Thread");
                Thread t4 = new Thread (tg1, can, "Fourth
                Thread"); //find parent group of tg1

                System.out.println ("Parent of tg1 = " + tg1getParent ());
                //set maximum priority
                tg1setMaxPriority (7);
                System.out.println ("Thread group of t1 = " + t1getThreadGroup
                ());   System.out.println   ("Thread   group   of   t3   =   "   +
                t3getThreadGroup ()); t1start ();
                t2start ();
                t3start ();
                t4start ();
                System.out.println ("Number of threads in this group : " + tgactiveCount () );
        }
}
class Reservation extends Thread
{public void run ()
        {System.out.println ("I am Reservation Thread");
        }
}
class Cancellation extends Thread
{public void run ()
        {System.out.println ("I am Cancellation Thread");
        }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac WhyTGroups.java

D:\JQR>java  WhyTGroups
Parent of tg1 = java.lang.ThreadGroup[name=Reservation Group,maxpri=10]
Thread group of t1 = java.lang.ThreadGroup[name=Reservation Group,maxpri=10]
Thread group of t3 = java.lang.ThreadGroup[name=Cancellation Group,maxpri=7]
I am Reservation Thread
Number of threads in this group : 4
I am Cancellation Thread
I am Cancellation Thread
I am Reservation Thread

D:\JQR>_
```

Thread States (Life-Cycle of a Thread): The life cycle of a thread contains several states At any time the thread falls into any one of the states

The thread that was just created is in the born state

The thread remains in this state until the threads start method is called This causes the thread to enter the ready state

The highest priority ready thread enters the running state when system assigns a processor to the thread ie, the thread begins executing

When a running thread calls wait the thread enters into a waiting state for the particular object on which wait was called Every thread in the waiting state for a given object becomes ready on a call to notify all by another thread associated with that object

When a sleep method is called in a running thread that thread enters into the suspended (sleep) state A sleeping thread becomes ready after the designated sleep time expires A sleeping thread cannot use a processor even if one is available A thread enters the dead state when its run () method completes (or) terminates for any reason A dead thread is eventually be disposed of by the system

One common way for a running thread to enter the blocked state is when the thread issues an input or output request In this case a blocked thread becomes ready when the input or output waits for completes A blocked thread can‟t use a processor even if one is available

UNIT - IV

The Collections Framework (java.util)- Collections overview, Collection Interfaces, The Collection classes- Array List, Linked List, Hash Set, Tree Set, Priority Queue, Array Deque. Accessing a Collection via an Iterator, Using an Iterator, The For-Each alternative, Map Interfaces and Classes, Comparators, Collection algorithms, Arrays, The Legacy Classes and Interfaces- Dictionary, Hashtable ,Properties, Stack, Vector

More Utility classes, String Tokenizer, Bit Set, Date, Calendar, Random, Formatter, Scanner

The Collection Framework

In order to handle group of objects we can use array of objects If we have a class called Employ with members name and id, if we want to store details of 10 Employees, create an array of object to hold 10 Employ details

Employ ob [] = new Employ [10];

We cannot store different class objects into same array

Inserting element at the end of array is easy but at the middle is difficult

After retriving the elements from the array, in order to process the elements we dont have any methods

Collection Object:

A collection object is an object which can store group of other objects

A collection object has a class called Collection class or Container class

All the collection classes are available in the package called 'javautil" (util stands for utility)

Group of collection classes is called a Collection Framework

A collection object does not store the physical copies of other objects; it stores references of other objects

All the collection classes in javautil package are the implementation classes of different interfaces

Interface Type	Implementation Classes
Set <T>	HashSet<T>
	LinkedHashSet<T>
List <T>	Stack<T>
	LinkedList<T>
	ArrayList<T>
	Vector<T>
Queue <T>	LinkedList<T>
Map<T>	HashMap<K,V>
	Hashtable<K,V>

Set: A Set represents a group of elements (objects) arranged just like an array The set will grow dynamically when the elements are stored into it A set will not allow duplicate elements

List: Lists are like sets but allow duplicate values to be stored

Queue: A Queue represents arrangement of elements in FIFO (First In First Out) order This means that an element that is stored as a first element into the queue will be removed first from the queue

Map: Maps store elements in the form of key value pairs If the key is provided its corresponding value can be obtained

Retrieving Elements from Collections: Following are the ways to retrieve any element from a collection object:

Using Iterator interface
Using ListIterator interface
Using Enumeration interface

Iterator Interface: Iterator is an interface that contains methods to retrieve the elements one by one from a collection object It retrieves elementsonly in forward direction It has 3 methods:

Method	**Description**
boolean hasNext()	This method returns true if the iterator has more elements
element next()	This method returns the next element in the iterator
void remove()	This method removes the last element from the collection returned by the iterator

ListIterator Interface: ListIterator is an interface that contains methods to retrieve the elements from a collection object, both in forward and reverse directions It can retrieve the elements in forward and backward direction It has the following important methods:

Method	**Description**
boolean hasNext()	This method returns true if the ListIterator has more elements when traversing the list in forward direction
element next()	This method returns the next element
void remove()	This method removes the list last element that was returned by The next () or previous () methods
boolean hasPrevious()	This method returns true if the ListIterator has more elements when traversing the list in reverse direction
element previous()	This method returns the previous element in the list

Enumeration Interface: This interface is useful to retrieve elements one by one like Iterator It has 2 methods

Method	**Description**
boolean hasMoreElements()	This method tests Enumeration has any more elements
element nextElement()	This returns the next element that is available in Enumeration

HashSet Class: HashSet represents a set of elements (objects) It does not guarantee the order of elements Also it does not allow the duplicate elements to be stored

- We can write the HashSet class as: class HashSet<T>
- We can create the object as: HashSet<String> hs = new HashSet<String> ();

HashSet();
HashSet (int capacity); Here capacity represents how many elements can be stored into th
HashSet initially This capacity may increase automatically when more number of elements is being stored

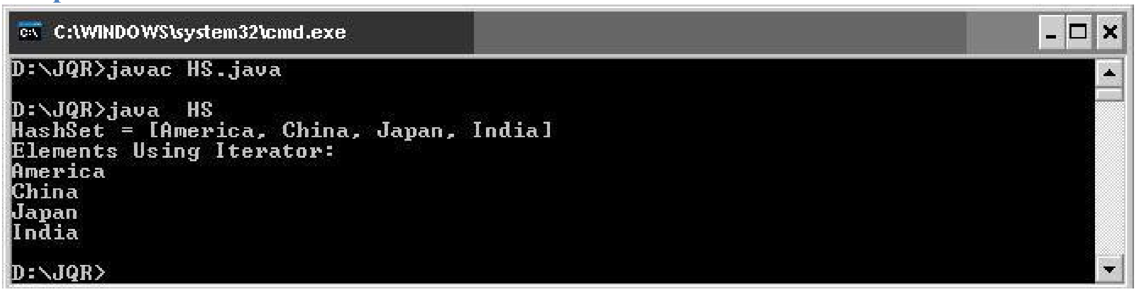
:

	Description
boolean add(obj) element is added to the	This method adds an element obj to the HashSet It returns true if the HashSet, else it returns false If the same element is already available in the HashSet, then the present element is not added
boolean remove(obj)	This method removes the element obj from the HashSet, if it is present It returns true if the element is removed successfully otherwise false
void clear()	This removes all the elements from the HashSet
boolean contains(obj)	This returns true if the HashSet contains the specified element obj
boolean isEmpty()	This returns true if the HashSet contains no elements
int size()	This returns the number of elements present in the HashSet

Program 1: Write a program which shows the use of HashSet and Iterator

```
//HashSet Demo
import javautil*;
class HS
{public static void main(String args[])
        {//create a HashSet to store Strings
                HashSet <String> hs = new HashSet<String> ();
                //Store some String elements
                hsadd       ("India");
                hsadd    ("America");
                hsadd       ("Japan");
                hsadd       ("China");
                hsadd    ("America");
                //view the HashSet
                System.out.println ("HashSet = " + hs);
                //add an Iterator to hs
                Iterator it = hsiterator ();
                //display element by element using Iterator
                System.out.println ("Elements Using Iterator: ");
                while (ithasNext() )
                {Strings = (String)      itnext   ();
                        System.out.println(s);
                }
        }
}
```

Output:



```
D:\JQR>javac HS.java

D:\JQR>java  HS
HashSet = [America, China, Japan, India]
Elements Using Iterator:
America
China
Japan
India

D:\JQR>
```

LinkedHashSet Class: This is a subclass of HashSet class and does not contain any additional members on its own LinkedHashSet internally uses a linked list to store the elements It is a generic class that has the declaration:
class LinkedHashSet<T>

Stack Class: A stack represents a group of elements stored in LIFO (Last In First Out) order This means that the element which is stored as a last element into the stack will be the first element to be removed from the stack Inserting the elements (Objects) into the stack is called push operation and removing the elements from stack is called pop operation Searching for an element in stack is called peep operation Insertion and deletion of elements take place only from one side of the stack, called top of the stack We can write a Stack class as:

class Stack<E>

eg: Stack<Integer> obj = new Stack<Integer> ();

Stack Class Methods:

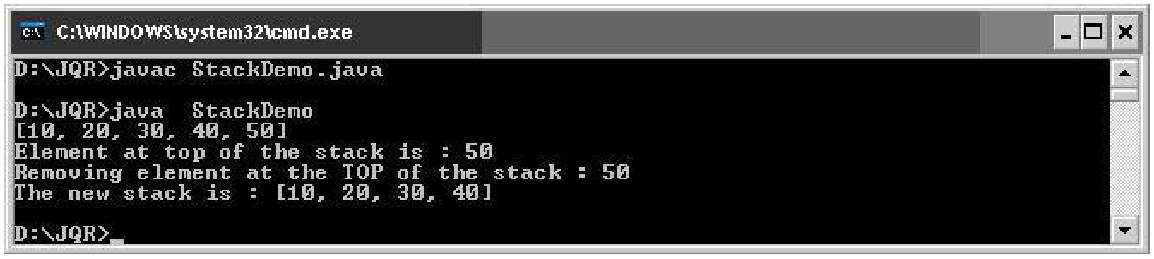
Method	Description
boolean empty()	this method tests whether the stack is empty or not If the stack is empty then true is returned otherwise false
element peek()	this method returns the top most object from the stack without removing it
element pop()	this method pops the top-most element from the stack and returns it
element push(element obj)	this method pushes an element obj onto the top of the stack and returns that element
int search(Object obj)	This method returns the position of an element obj from the top of the stack If the element (object) is not found in the stack then it returns -1

Program 2: Write a program to perform different operations on a stack

```
//pushing, popping, searching elements in a stack
import javautil*;
class StackDemo
{
        public static void main(String args[])
        {//create an empty stack to contain Integer objects
                Stack<Integer> st = new Stack<Integer>();
                stpush (new Integer(10) );
                stpush (new Integer(20)
                ); stpush (new
                Integer(30) ); stpush
                (new Integer(40) );
                stpush (new Integer(50) );
                System.out.println (st);
                System.out.println ("Element at top of the stack is : " + stpeek() );
                System.out.println ("Removing element at the TOP of the stack : " +
                stpop()); System.out.println ("The new stack is : " + st);
        }
}
```

.

Output:


C:\WINDOWS\system32\cmd.exe

```
D:\JQR>javac StackDemo.java

D:\JQR>java  StackDemo
[10, 20, 30, 40, 50]
Element at top of the stack is : 50
Removing element at the TOP of the stack : 50
The new stack is : [10, 20, 30, 40]

D:\JQR>_
```

LinkedList Class: A linked list contains a group of elements in the form of nodes Each node will have three fields- the data field contatins data and the link fields contain references to previous and next nodesA linked list is written in the form of:

class LinkedList<E>

we can create an empty linked list for storing String type elements (objects) as:

LinkedList <String> ll = new LinkedList<String> ();

LinkedList Class methods:

Method	Description
boolean add (element obj)	This method adds an element to the linked list It returns true if the element is added successfully
void add(int position, element obj)	This method inserts an element obj into the linked list at a specified Position
void addFirst(element obj)	This method adds the element obj at the first position of the linked List
void addLast(element obj)	This method adds the element obj at the last position of the linked List
element removeFirst ()	This method removes the first element from the linked list and returns it
element removeLast ()	This method removes the last element from the linked list and returns it
element remove (int position)	This method removes an element at the specified position in the linked list
void clear ()	This method removes all the elements from the linked list
element get (int position)	This method returns the element at the specified position in the linked list
element getFirst ()	This method returns the first element from the list
element getLast ()	This method returns the last element from the list
element set(int position, element obj)	This method replaces the element at the specified position in the list with the specified element obj
int size ()	Returns number of elements in the linked list
int indexOf (Object obj)	This method returns the index of the first occurrence of the specified element in the list, or -1 if the list does not contain the Element
int lastIndexOf (Object obj)	This method returns the index of the last occurrence of the Specified element in the list, or -1 if the list does not contain the element

.

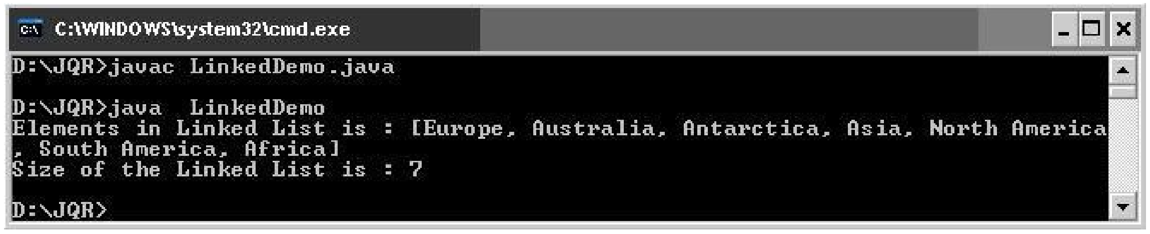
Object[] toArray()	This method converts the linked list into an array of Object class type All the elements of the linked list will be stored into the array in the same sequence

Note: In case of LinkedList counting starts from 0 and we start counting from 1

Program 3: Write a program that shows the use of LinkedList class

```
import javautil*;
//Linked List
class LinkedDemo
{public static void main(String args[])
        {LinkedList <String> ll = new LinkedList<String>();
                lladd ("Asia");
                lladd ("North America");
                lladd ("South America");
                lladd          ("Africa");
                lladdFirst     ("Europe");
                lladd       (1,"Australia");
                lladd (2,"Antarctica");
                System.out.println ("Elements in Linked List is : " + ll);
                System.out.println ("Size of the Linked List is : " + llsize()
                );
        }
}
```

Output:



```
D:\JQR>javac LinkedDemo.java

D:\JQR>java  LinkedDemo
Elements in Linked List is : [Europe, Australia, Antarctica, Asia, North America
, South America, Africa]
Size of the Linked List is : 7

D:\JQR>
```

ArrayList Class: An ArrayList is like an array, which can grow in memory dynamically ArrayList is not synchronized This means that when more than one thread acts simultaneously on the ArrayList object, the results may be incorrect in some cases

ArrayList class can be written as: class ArrayList <E>

We can create an object to ArrayList as: ArrayList <String> arl = new ArrayList<String> ();

ArrayList Class Methods:

Method	Description
boolean add (element obj)	This method appends the specified element to the end of the ArrayList If the element is added successfully then the method returns true
void add(int position, element obj)	This method inserts the specified element at the specified position in the ArrayList
element remove(int position)	This method removes the element at the specified position in the ArrayList and returns it
boolean remove (Object	This method removes the first occurrence of the specified element

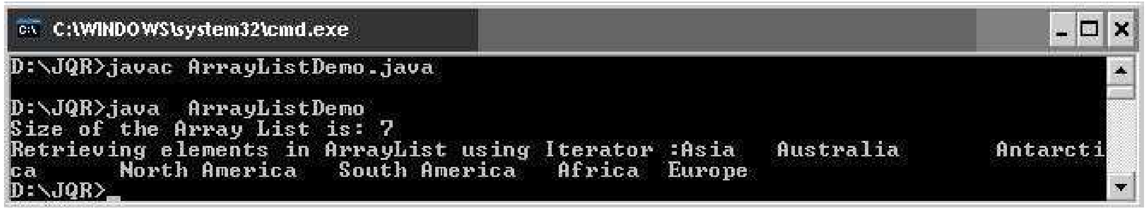
.   .

obj)	obj from the ArrayList, if it is present
void clear ()	This method removes all the elements from the ArrayList
element set(int position, element obj)	This method replaces an element at the specified position in the ArrayList with the specified element obj
boolean contains (Object obj)	This method returns true if the ArrayList contains the specified element obj
element get (int position)	This method returns the element available at the specified position in the ArrayList
int size ()	Returns number of elements in the ArrayList
int indexOf (Object obj)	This method returns the index of the first occurrence of the specified element in the list, or -1 if the list does not contain the Element
int lastIndexOf (Object obj)	This method returns the index of the last occurrence of the specified element in the list, or -1 if the list does not contain the element
Object[] toArray ()	This method converts the ArrayLlist into an array of Object class type All the elements of the ArrayList will be stored into the array in the same sequence

Program 4: Write a program that shows the use of ArrayList class

```
import javautil*;
//ArrayList      Demo
class ArrayListDemo
{public static void main(String args[])
        {ArrayList <String> al = new ArrayList<String>();
                aladd ("Asia");
                aladd                   ("North
                America");              aladd
                ("South          America");
                aladd   ("Africa");   aladd
                ("Europe");             aladd
                (1,"Australia");       aladd
                (2,"Antarctica");
                System.out.print  ("Size  of  the  Array  List  is:  "  +  alsize  ());
                System.out.print ("\nRetrieving elements in ArrayList using Iterator
                :"); Iterator it = aliterator ();
                while (ithasNext () )
                        System.out.print (itnext () + "\t");
        }
}
```

Output:



```
D:\JQR>javac ArrayListDemo.java

D:\JQR>java  ArrayListDemo
Size of the Array List is: 7
Retrieving elements in ArrayList using Iterator :Asia    Australia       Antarcti
ca      North America   South America   Africa  Europe
D:\JQR>_
```

Vector Class: Similar to ArrayList, but Vector is synchronized It means even if several threads act on Vector object simultaneously, the results will be reliable
Vector class can be written as: class Vector <E>
We can create an object to Vector as: Vector <String> v = new Vector<String> ();

Vector Class Methods:

Method	Description
boolean add(element obj)	This method appends the specified element to the end of the Vector If the element is added successfully then the method returns true
void add (int position, element obj)	This method inserts the specified element at the specified position in the Vector
element remove (int position)	This method removes the element at the specified position in the Vector and returns it
boolean remove (Object obj)	This method removes the first occurrence of the specified element obj from the Vector, if it is present
void clear ()	This method removes all the elements from the Vector
element set (int position, element obj)	This method replaces an element at the specified position in the Vector with the specified element obj
boolean contains (Object obj)	This method returns true if the Vector contains the specified element obj
element get (int position)	This method returns the element available at the specified position in the Vector
int size ()	Returns number of elements in the Vector
int indexOf (Object obj)	This method returns the index of the first occurrence of the specified element in the Vector, or -1 if the Vector does not contain the element
int lastIndexOf (Object obj)	This method returns the index of the last occurrence of the Specified element in the Vector, or -1 if the Vector does not contain the Element
Object[] toArray ( )	This method converts the Vector into an array of Object class type All the elements of the Vector will be stored into the array in the same sequence
int capacity ()	This method returns the current capacity of the Vector

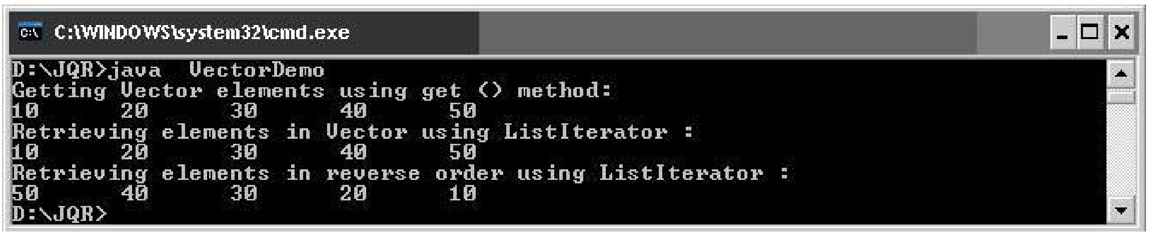
Program 5: Write a program that shows the use of Vector class

```
import javautil*;
//Vector      Demo
class VectorDemo
{public static void main(String args[])
        {Vector <Integer> v = new Vector<Integer> ();
                int x[] = {10,20,30,40,50};
                //When x[i] is stored into v below, x[i] values are converted into Integer Objects
               //and stored into v This is auto boxing
                for (int i = 0; i<xlength; i++)
                        vadd(x[i]);
System.out.println ("Getting Vector elements using get () method: ");
```

.

```
                    for  (int  i  =  0;  i<vsize();  i++)
                        System.out.print   (vget   (i)   +
                        "\t");
                System.out.println  ("\nRetrieving  elements  in  Vector  using  ListIterator
                :"); ListIterator lit = vlistIterator ();
                while (lithasNext () )
                        System.out.print (litnext () + "\t");
                System.out.println ("\nRetrieving elements in reverse order using ListIterator
                :");      while      (lithasPrevious      ()      )
                        System.out.print   (litprevious   ()   +
                        "\t");
        }
}
```

Output:



```
D:\JQR>java  VectorDemo
Getting Vector elements using get () method:
10      20      30      40      50
Retrieving elements in Vector using ListIterator :
10      20      30      40      50
Retrieving elements in reverse order using ListIterator :
50      40      30      20      10
D:\JQR>
```

HashMap Class: HashMap is a collection that stores elements in the form of key-value pairs If key is provided later its corresponding value can be easily retrieved from the HAshMap Keys should be unique HashMap is not synchronized and hence while using multiple threads on HashMap object, we get unreliable results

We can write HashMap class as: class HashMap<K, V>

For example to store a String as key and an integer object as its value, we can create the HashMap as: HashMap<String, Integer> hm = new HashMap<String, Integer> ();

The default initial capacity of this HashMap will be taken as 16 and the load factor as 075 Load factor represents at what level the HashMap capacity should be doubled For example, the product of capacity and load factor = 16 * 075 = 12 This represents that after storing 12th key-value pair into the HashMap, its capacity will become 32

HashMap Class Methods:

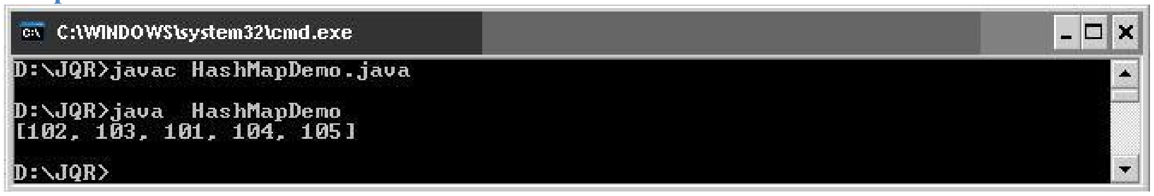
Method	Description
value put (key, value)	This method stores key-value pair into the HashMap
value get (Object key)	This method returns the corresponding value when key is given If the key does not have a value associated with it, then it returns null
Set<K> keyset()	This method, when applied on a HashMap converts it into a set where only keys will be stored
Collection <V> values()	This method, when applied on a HashMap object returns all the values of the HashMap into a Collection object
value remove (Object key)	This method removes the key and corresponding value from the HashMap
void clear ()	This method removes all the key-value pairs from the map
boolean isEmpty ()	This method returns true if there are no key-value pairs in the HashMap
int size ()	This method returns number of key-value pairs in the HashMap

. .

Program 6: Write a program that shows the use of HashMap class

```
//HashMap      Demo
import       javautil*;
class HashMapDemo
{public static void main(String args[])
        {HashMap<Integer,  String>  hm  =  new  HashMap<Integer,  String>  ();
                hmput (new Integer (101),"Naresh");
                hmput   (new   Integer   (102),"Rajesh");
                hmput   (new   Integer   (103),"Suresh");
                hmput  (new  Integer  (104),"Mahesh");
                hmput  (new  Integer  (105),"Ramesh");
                Set<Integer>         set         =         new
                HashSet<Integer>(); set = hmkeySet();

                `System.out.println (set);
        }
}
```

Output:



```
D:\JQR>javac HashMapDemo.java

D:\JQR>java  HashMapDemo
[102, 103, 101, 104, 105]

D:\JQR>
```

Hashtable Class: Hashtable is a collection that stores elements in the form of key-value pairs If key is provided later its corresponding value can be easily retrieved from the HAshtable Keys should be unique Hashtable is synchronized and hence while using multiple threads on Hashtable object, we get reliable results
We can write Hashtable class as: class Hashtable<K,V>
For example to store a String as key and an integer object as its value, we can create the Hashtable as: Hashtable<String, Integer> ht = new Hashtable<String, Integer> ();
The default initial capacity of this Hashtable will be taken as 11 and the load factor as 075 Load factor represents at what level the Hashtable capacity should be doubled For example, the product of capacity and load factor = 11 * 075 = 825 This represents that after storing 8th key-value pair into the Hashtable, its capacity will become 22

Hashtable Class Methods:

Method	Description
value put(key, value)	This method stores key-value pair into the Hashtable
value get(Object key)	This method returns the corresponding value when key is given If the key does not have a value associated with it, then it returns null
Set<K> keyset()	This method, when applied on a Hashtable converts it into a set where only keys will be stored
Collection <V> values()	This method, when applied on a Hashtable object returns all the values of the Hashtable into a Collection object
value remove(Object key)	This method removes the key and corresponding value from the Hashtable

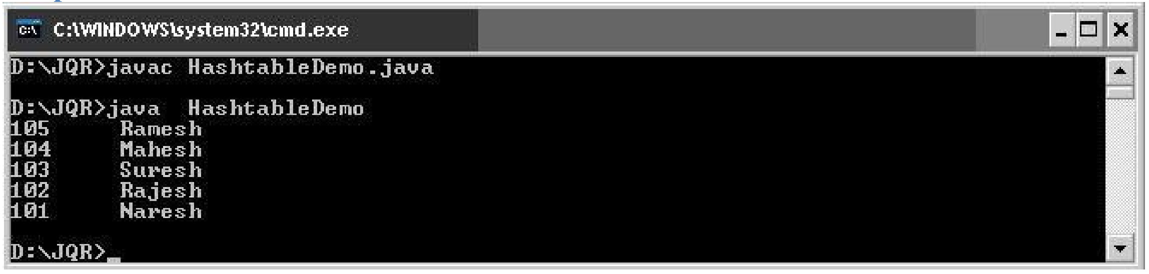
.

void clear()	This method removes all the key-value pairs from the Hashtable
boolean isEmpty()	This method returns true if there are no key-value pairs in the Hashtable
int size()	This method returns number of key-value pairs in the Hashtable

Program 7: Write a program that shows the use of Hashtable class

```
//Hashtable      Demo
import       javautil*;
class HashtableDemo
{public static void main(String args[])
        {
                Hashtable<Integer,   String>  ht   =  new   Hashtable<Integer,
                String> (); htput (new Integer (101),"Naresh");
                htput           (new            Integer
                (102),"Rajesh");      htput      (new
                Integer    (103),"Suresh");    htput
                (new   Integer   (104),"Mahesh");
                htput           (new            Integer
                (105),"Ramesh");
                Enumeration   e   =   htkeys   ();
                while ( ehasMoreElements () )
                {
                        Integer  i1  =  (Integer)  enextElement  ();
                        System.out.println  (i1   +   "\t"   +   htget
                        (i1));
                }
        }
}
```

Output:



```
D:\JQR>javac HashtableDemo.java

D:\JQR>java  HashtableDemo
105     Ramesh
104     Mahesh
103     Suresh
102     Rajesh
101     Naresh

D:\JQR>_
```

Arrays Class: Arrays class provides methods to perform certain operations on any single dimensional array All the methods of the Arrays class are static, so they can be called in the form of Arraysmethodname ()

Arrays Class Methods:

Method	Description
static void sort (array)	This method sorts all the elements of an array into ascending order This method internally uses QuickSort algorithm
static void sort (array, int start, int end)	This method sorts the elements in the range from start to end within an array into ascending order

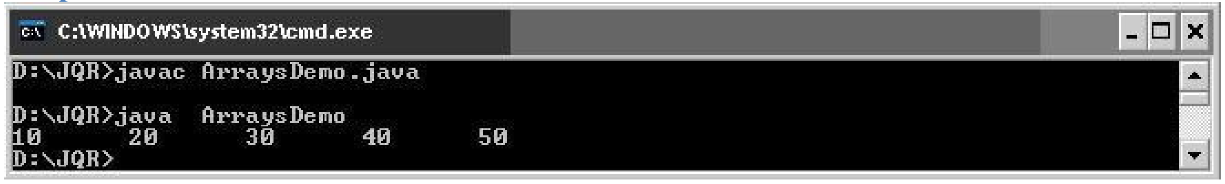
.

static int binarySearch (array, element)	This method searches for an element in the array and returns its position number If the element is not found in the array, it returns a negative value Note that this method acts only on an array which is sorted in ascending order This method internally uses BinarySearch algorithm
static boolean equals (array1, array2)	This method returns true if two arrays, that is array1 and array2 are equal, otherwise false
static array copyOf (source-array, int n)	This method copies n elements from the source-array into another array and returns the array
static void fill (array, value)	This method fills the array with the specified value It means that all the elements in the array will receive that value

Program 8: Write a program to sort given numbers using sort () method of Arrays Class import javautil*;

```
//Array
s Demo
class
Arrays
Demo
{public static void main(String args[])
        {
                int         x[]         =
                {40,50,10,30,20};
                Arrayssort( x );
                for                 (int
                        i=0;i<xlength;
                        i++)
                        System.out.pr
                        int(x[i]       +
                        "\t");
        }
}
```

Output:



```
D:\JQR>javac ArraysDemo.java

D:\JQR>java  ArraysDemo
10      20      30      40      50
D:\JQR>
```

StringTokenizer: The StringTokenizer class is useful to break a String into small pieces called tokens We can create an object to StringTokenizer as:

StringTokenizer st = new StringTokenizer (str, "delimeter");

StringTokenizer Class Methods:

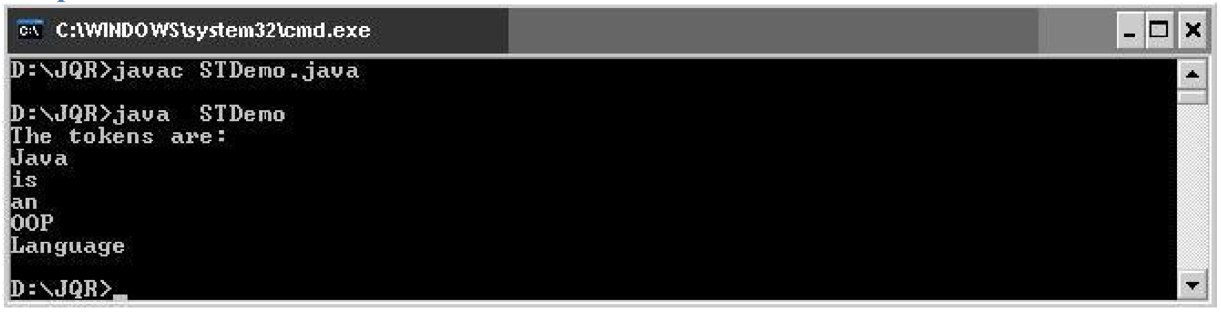
Method	Description
String nextToken()	Returns the next token from the StringTokenizer
boolean hasMoreTokens()	Returns true if token is available and returns false if not available
int countTokens()	Returns the number of tokens available

Program 9: Write a program that shows the use of StringTokenizer object

```
//cutting the String
into tokens import
javautil*;
class STDemo
{
public static void main(String args[])
        {//take a String
                String str = "Java is an OOP Language";
                //brake wherever a space is found
                StringTokenizer    st    =    new
                StringTokenizer (str," "); //retrieve
                tokens          and          display
                System.out.println   ("The   tokens
                are: ");

                while ( sthasMoreTokens () )
                {
                        String   s   =
                        stnextToken
                        ();
                        System.out.p
                        rintln (s
                        );
                }
        }
```

}

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\JQR>javac STDemo.java

D:\JQR>java  STDemo
The tokens are:
Java
is
an
OOP
Language

D:\JQR>_
```

Calendar: This class is useful to handle date and time We can create an object to Calendar class
as: Calendar cl = CalendargetInstance ();

Calendar Class Methods:

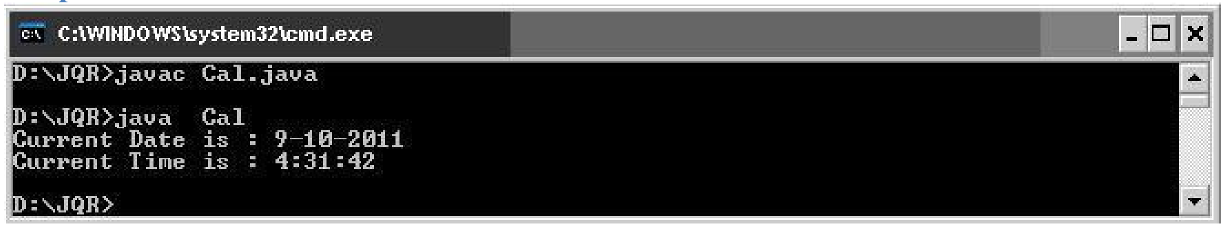
Method	Description
int get(Constant)	This method returns the value of the given Calendar constant Examples of Constants are CalendarDATE, CalendarMONTH, CalendarYEAR, CalendarMINUTE, CalendarSECOND, CalendarHour
void set(int field, int value)	This method sets the given field in Calendar Object to the given value For example, clset(CalendarDATE,15);
String toString()	This method returns the String representation of the Calendar object
boolean equals(Object obj)	This method compares the Calendar object with another object obj and returns true if they are same, otherwise false

Program 10: Write a program to display System time and date

```
//To display system time
and date import
javautil*;
class Cal
{public static void main(String args[])
	{Calendar cl = CalendargetInstance ();
```

```
                    //Retrieve Date
                    int dd = clget (CalendarDATE);
                    int mm = clget (CalendarMONTH);
                    ++mm;
                    int yy = clget (CalendarYEAR);
                    System.out.println ("Current Date is : " + dd + "-" +
                    mm + "-" + yy ); //Retrieve Time
                    int     hh     =     clget
                    (CalendarHOUR);    int
                    mi          =          clget
                    (CalendarMINUTE);
                    int     ss     =     clget
                    (CalendarSECOND);
System.out.println ("Current Time is : " + hh + ":" + mi + ":" +ss);
          }
  }
```

Output:



```
D:\JQR>javac Cal.java

D:\JQR>java  Cal
Current Date is : 9-10-2011
Current Time is : 4:31:42

D:\JQR>
```

Date Class: Date Class is also useful to handle date and time Once Date class object is created, it should be formatted using the following methods of DateFormat class of javatext packageWe
can create an object to Date class as: Date dd = new Date ();
Once Date class object is created, it should be formatted using the methods of DateFormat class of javatext package

DateFormat class Methods:

DateFormat fmt = DateFormatgetDateInstance(formatconst, region);
This method is useful to store format information for date value into DateFormat object fmt

DateFormat fmt = DateFormatgetTimeInstance(formatconst, region);
This method is useful to store format information for time value into DateFormat object fmt

DateFormat fmt = DateFormatgetDateTimeInstance(formatconst, formatconst, region); This method is useful to store format information for date value into DateFormat object fmt

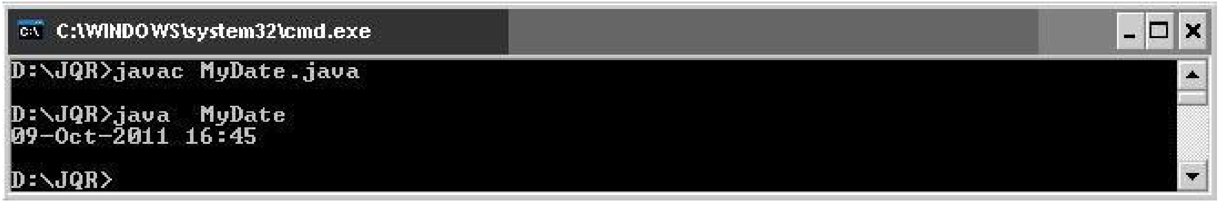
Formatconst	**Example (region=LocaleUK)**
DateFormatFULL	03 september 2007 19:43:14 O'Clock GMT + 05:30
DateFormatLONG	03 september 2007 19:43:14 GMT + 05:30

DateFormatMEDIUM	03-sep-07 19:43:14
DateFormatSHORT	03/09/07 19:43

Program 11: Write a program that shows the use of Date class

```
//Display System date and time using
Date class import javautil*;
impor
t
javate
xt*;
class
MyDa
te
{
        public static void main(String args[])

        {Date d = new Date ();
                DateFormat    fmt    =    DateFormatgetDateTimeInstance
            (DateFormatMEDIUM, DateFormatSHORT, LocaleUK);
                String   str   =
                fmtformat  (d);
                System.out.pri
                ntln (str);
        }
}
```

Output:



```
D:\JQR>javac MyDate.java

D:\JQR>java  MyDate
09-Oct-2011 16:45

D:\JQR>
```