

# Core Java

## Introduction:

In the olden days before introducing computer technology into the market electronic devices were highly consumed. We already know to operate the electronic devices we have to run the microprocessor which is available inside the circuit. At this particular requirement for running the microprocessor we have to implement the programming code. This requirement gave the life for evaluating the computer technologies into the market. We already know requirement is one of the important role playing in the software industry. So on the basis of above requirement they developed first language called machine language. On developing the applications(programs) on this language leads to following drawbacks.

- 1)code complexity
- 2)understanding the language is very difficult.

So by identifying the above drawbacks research development centre was proposed another kind of methodology called Procedure oriented methodology. The most of languages which has been developed on the basis of this kind of methodology. For Example like Fortran , cobol , pascal , c etc. What ever the languages which has been implemented on this kind of approach it leads to following drawbacks.....

- 1) Data and code that acts upon data was improperly organized means security was not be provided.
- 2) Complexity cannot be handled.

To overcome the above drawbacks the research development centre was identified and proposed another kind of approach called Object Oriented Programming Structure.

It has four features

1. Encapsulation
2. Abstraction
3. polymorphism
4. Inheritance

Encapsulation:-

Combining both data and code into single unit is called encapsulation.

Note:- It provides security

How can we provide a encapsulation in java programatically?

Sol:-In java encapsulation is provided by using a class keyword.

Abstraction :-

Presenting necessary things and hiding unnecessary things is called abstraction.

Note:- Abstraction is an output of encapsulation.

## Polymorphism:-

One object is behaving differently in different situations we say that particular object is exhibiting polymorphism.

In java there are two types of polymorphism

- 1) Static polymorphism
- 2) Dynamic polymorphism

The polymorphism which type is decided means either static or dynamic is totally depends upon binding.

Binding:-

Linking particular method call to particular method definition is called binding.

## Static polymorphism:-

Linking method call to method definition at compilation itself is called static polymorphism. It is also called early binding.

Dynamic polymorphism:-

Linking method call to method definition at run time itself is called dynamic polymorphism. It is also called late binding.

Inheritance:-

Aquiring superclass properties from subclass is called inheritance.

Note:- It provides code reusability.

After seeing the features of object oriented programming structure we discuss some of the important points of usage of java in real time environment.

1.Why java?

A) It is used in realtime environment for developing the projects due to the base on these factors.....

1) Developing the java on basis of object oriented programming structure it provides security and reducing the code complexity.

2) Platform independency

3) It supports and building for Dynamic webpage.

2. How many flavours of java?

A) There are 3 types

1) J2SE(JAVA 2 STANDARD EDITION)

2) J2EE(JAVA 2 ENTERPRISE EDITION)

3) J2ME(JAVA 2 MICRO EDITION)

3. What is the difference between platform dependency and platform independency?

a) Platform Dependency:-

The languages which has been implemented on the approach of procedure oriented methodology is called platform dependence applications.

Platform:- The environment which is provided for execution of a program is called platform.

Ex:- Cobol, Fortran, Pascal, c etc...

For example we take the c program assume that we have define 10 lines of code inside the c program. After compiling the program .object file is generated. After executing the program .exe file is generated. Suppose this program is executed on the dual core processor that kind of processor instructions was loaded on the .exe file. suppose we copy that file on the another operating system means another processor say some xeloron processor. So that .exe file was not executed there means that instructions what we have provided inside the .exe file does not match to this processor. So this is called platform dependency. This is the main reason our procedure oriented applications was not using in webapplication development.

b) Platform Independency:-

The languages which has been implemented on the basis of object oriented approach is called platform independence applications.

Ex:-java etc.....

Suppose we compile the java program with a java compiler it provides a class file. Class file means combination of byte code designed by ByteEngineeringLibrary(BEL). This class file we have to execute on java platform with the help of JVM(java virtual machine) provided by our jdk(java development kit) software released by sun networks.

## INTRODUCTION TO CLASS AND OBJECT:-

In the programming world we have so many data items to be represented and to be manipulated. All these data items cannot be flexible represent with the help of primitive data types.

In the c language with the help of structure we can create user defined data types.



In object oriented implementation by using class we can create user defined data type.

A structure in C cannot organize code and data into the one place. Therefore we say that it does not support encapsulation.

So finally class is the basis for encapsulation. But class is only a data type. By creating a data type we cannot store data. In business application development data processing in a secured manner is a major concern.

In order to process data we have to store it. Therefore we have to create a variable of type class. This variable is nothing but an object.

An object is an encapsulated unit.

A class is a basis for encapsulation whereas the object implements the encapsulation.

Class means collection of variables and methods.

Every object is associated with three things.....

1. Properties
2. Behavior
3. Identity

## Properties:-

Variables of an object are nothing but properties. Properties and current values together is known as state of the object.

## Behavior:-

The functional part of the object means methods is known as the behavior of objects.

## Identity:-

The name of the object which is uniquely identified is known as identity of the object.

## Introduction to loops:-

There are 3 loops

- 1) While loop
- 2) For loop
- 3) Do while loop

## While loop:-

It checks the condition first when ever condition becomes true it enters into the loop and the

loop will be repeated until the condition will be false. If the condition is false the controller comes out of the loop.

Syntax:

While(condition)

```
{  
}
```

Ex:-

For example i=1;

while(i<=10)

```
{
```

System.out.println(i);

i++; //value is post incremented every time.

```
}
```

Here output is 1....10 is printed.

For loop:-

It initializes the value first, after that it checks the condition whenever it is true it enters after that it increment the value again like that it processed.

Syntax:

```
for(initialization;condition;incrementation)
{
}
```

Ex:-

```
for(int i=1;i<=10;i++)
{
System.out.println(i);
}
```

doWhile:-

It enters into the loop with out checking the condition first after executing the statement one time then it checks the condition if it is true the loop will be repeated otherwise the controller comes out of the loop.

Ex:

```
int i=1;
do
{
```

```
System.out.println(i);
```

```
i++;
```

```
}while(i<=10);
```

## INTRODUCTION TO DATATYPES:-

There are two types:

- 1) Primitive data types
- 2) Advanced data types

Primitive data types:-

The primitive data type is a data type it stores single value.

Ex:- int , float etc

Advanced data types:-

The advanced data type is a data type it holds multiple values.

Ex:- String , boolean

How to write a first program in java?

Steps to write a program:

1. Create a folder with any name in any drive. This folder acts as a default package.
2. We have to write a program in a notepad or editplus and save the program with any name but extension should be .java only. This file should be stored in folder created in step 1. We have to save this below program.

Ex:-

```
class A----→identifiers
{
    public static void main(String args[])
    {
        System.out.println("welcome to java");
    }
}
```

Note:- In java identifiers are our own creations.

### 3. Compiling the above program

In order to compile the program open the command prompt and set our directory as current working directory means in which directory our java program is saved for example see this structure and follow the steps.....

## Step 1:

Click the start button and go to run option as shown below fig1.....



## Step 2:

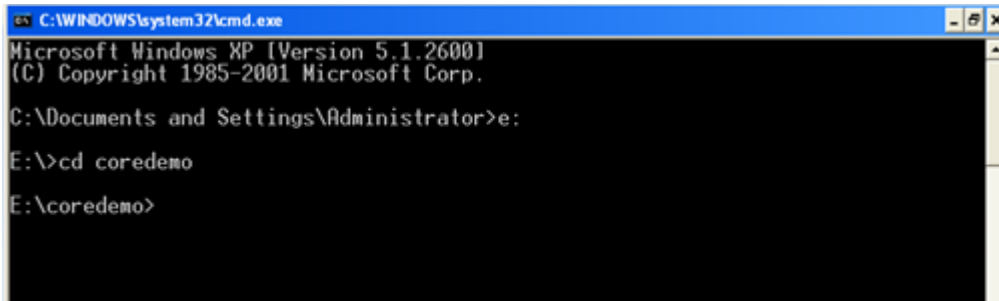
After clicking the run option a text box appears in that give cmd command as below fig2 follows.....



### Step 3:

After that a command prompt window is appeared in that our default directory is displayed means operating system directory is displayed. In that we have to change the existing directory our current working directory means in which drive our java program is saved see the below fig3.....



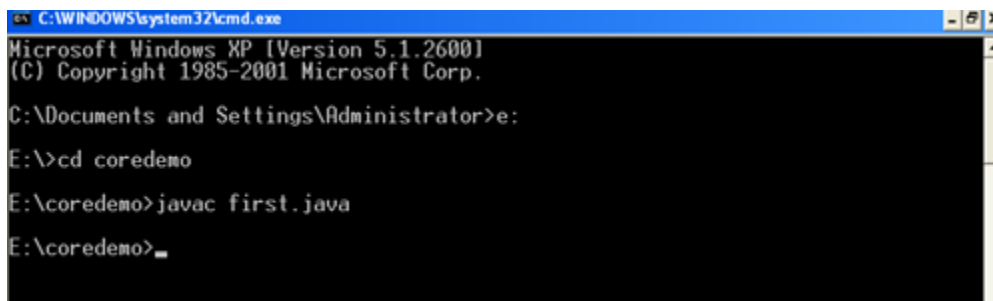


```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>e:
E:\>cd coredemo
E:\coredemo>
```

#### Step 4:-

After that we have to compile the java program by using java compiler as shown in the below fig4.....

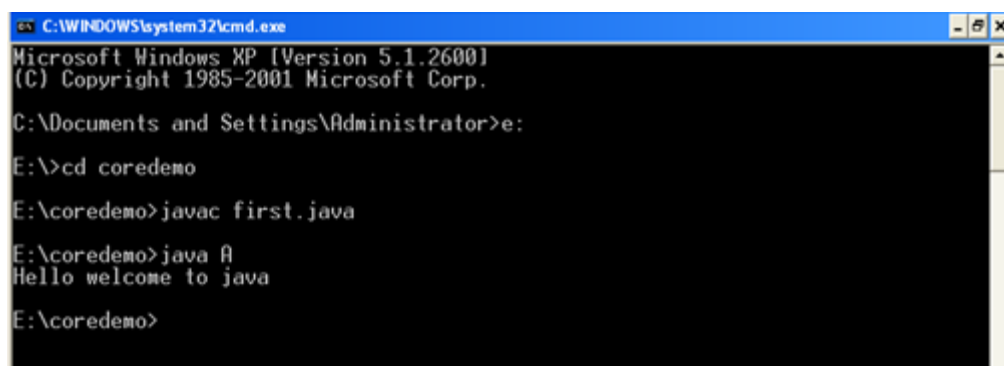


```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>e:
E:\>cd coredemo
E:\coredemo>javac first.java
E:\coredemo>
```

#### Step 5:

After compiling the program we have to execute it by using java virtual machine.....as shown in below fig5.....



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>e:
E:\>cd coredemo
E:\coredemo>javac first.java
E:\coredemo>java A
Hello welcome to java
E:\coredemo>
```

So by the above fig we get the output hello welcome to java.

Explanation of above program:-

1. In java any executable statement must be encapsulated inside the class ie variables and methods we call them as class members.
2. In the above program the class name A is defined to encapsulate the main method.
3. Here in above program the class name A is an identifier we can give the identifier names either in a small or upper case.

Note:-

1. main method is called by the jvm at the time of execution only. Jvm cannot called the main method if your are change the main method signature.
2. In java method signature is equal to method name plus parameter list.

Ex :- `main(String args[])`

Why public, static infront of main???

Public:- It is a keyword in java. We know what ever the code we define inside a class is encapsulated means

outer method call cannot be allowed inside a class. So public provides facility to the jvm without being the class member we can call main method directly.

Static: According to java principle, without object creation we cannot call any method directly. So static provides facility to jvm without object creation we can call main method directly.

Explanation of System.out.println statement????

- In order to print any thing on the command line in java we have to two library methods.
  1. System.out.print
  2. System.out.println

INTRODUCTION TO JAVA NAMING CONVENTIONS:-

We already know java is a case sensitive language means we have to follow strictly rules defined by the naming conventions whenever we are using predefined classes and methods.

1. In java predefined class names are started with in first word first letter is in uppercase and so on.....  
Ex: DataInputStream, String etc.....
2. In java predefined method names are started with in first word all letters are in small case and in second word first letter is in upper case and so on.....  
Ex: readLine(),println().....
3. In java predefined keywords all are in small case.  
Ex:- static, public etc.....

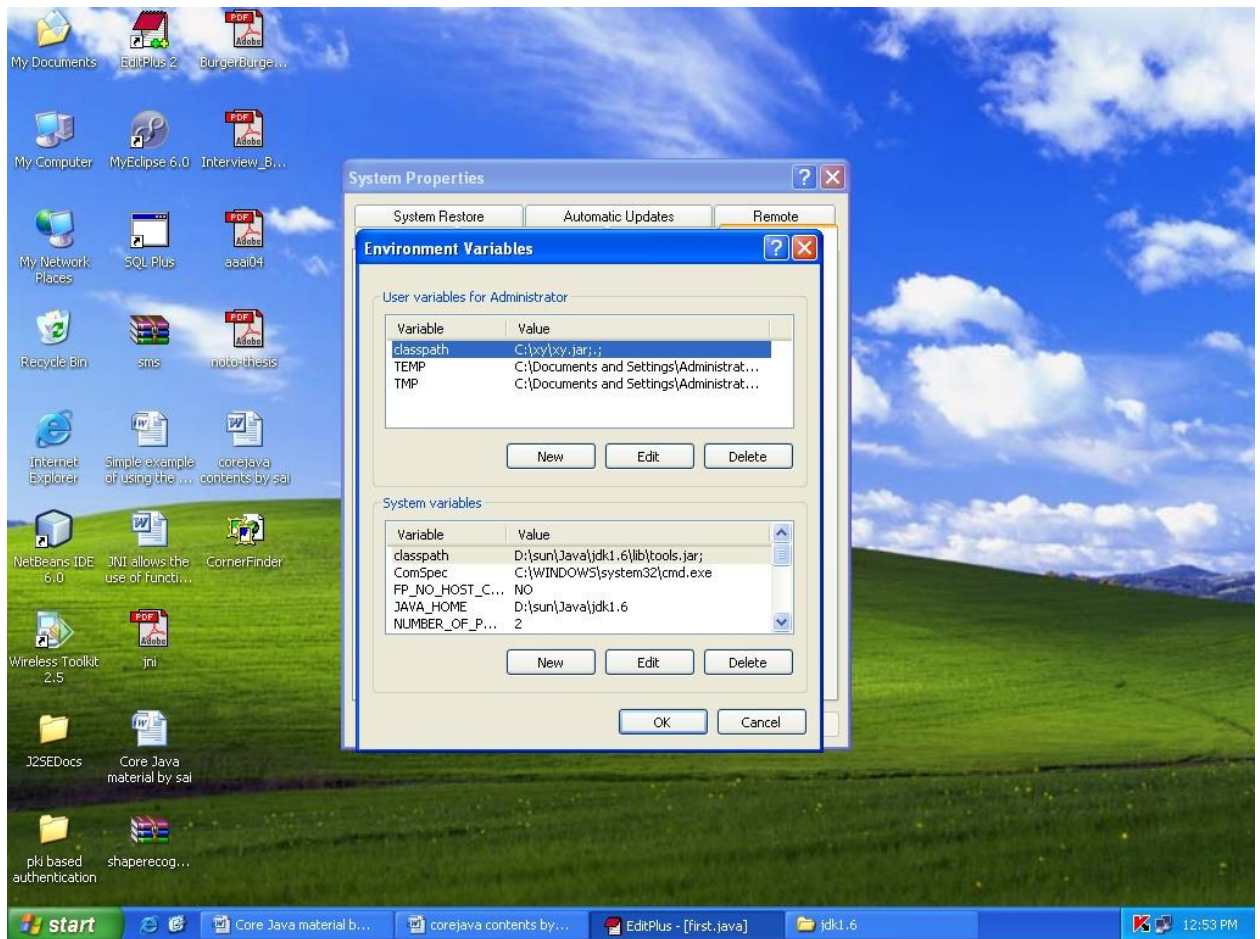
## EXPLANATION OF INSTALLING THE JAVA SOFTWARE:

1. We have to open the setup file of java.
2. We have to give the particular location name to the file is loaded when the installation is under progress.
3. After the installation is successfull we go to installation directory means where the set up file is loaded. for example D:\jdk1.6\bin  
Copy the above path

4. Minimize all the windows and go to mycomputer icon and left click on that and follow the direction as mentioned below.....

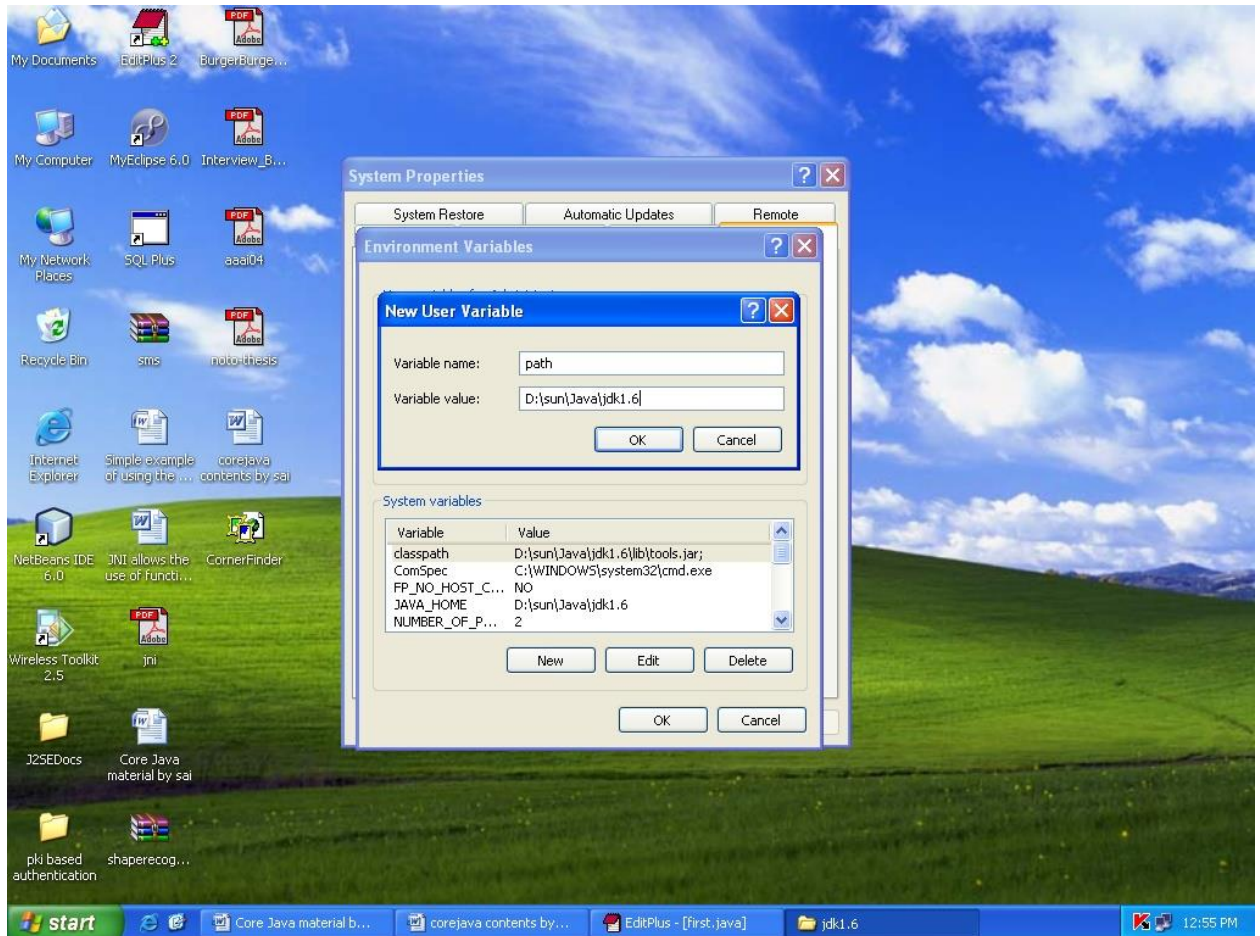
Properties->advanced->environmentalvariables

Click on that environmentalvariables tab we get a following below window.....



5. In that we go either for uservariables or systemvariables in that click on new button in that

variable name we give a path and variable value we have to paste the above directory into this a window is appears like as shown below.....

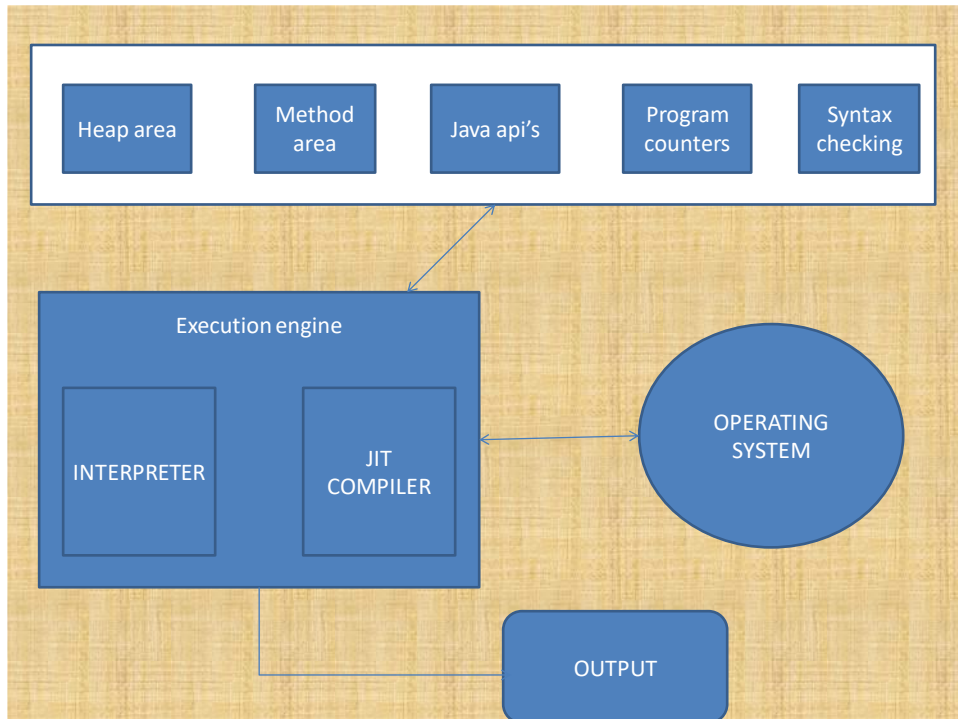


6. After that press double times ok and the path is successfully setted.

EXPLANATION ON JVM ARCHITECTURE:-

It has three parts

1. Class loader
2. Byte code verifier
3. Execution engine



Class loader:

It has two functions

- 1) It checks the given input class file available in the existing directory or not. If not found, it throws a runtime error, i.e., `NoClassDefoundError`. This error was raised on the following situations.....

1. Assume if the class file was saved on c drive and we were executed on another drive.
  2. Wrong path setting
- 2) It loads the class into the memory

ByteCodeVerifier:-

It verifies the code again whether you have provided correct syntax checking or not. If any error is provided in this case we call this error as runtime error. If it is successful it forwards the request to the execution engine.

Execution engine:-

It has two parts

- 1) Interpreter
- 2) JIT compiler

Interpreter:-

It acts as a both compiler and execution engine. The drawback is it checks line by line compilation and execution that's become more problem to the programmer. Another drawback is developing the applications on the interpreter increases the processor time this becomes programs slow.



To solve the above drawbacks JIT compiler was designed

JIT compiler:-

It is known as just in time compiler. It reduces the processor time and increases the programming speed. It is responsible to convert our byte code to machine code and viceversa. It invokes the interpreter when ever in your program mathematical logic is supplied.

Class variable and local variable:-

Class variable:

The variable which is not defined in inside a method is called class variable.

Local variable:-

The variable which is defined in inside a method is called local variable.

Example program of showing the difference between local variable and class variable.

```
class A
{
int a;//class variable
void f1()
    {
        int b;//local variable--→1
        System.out.println(a);
        System.out.println(b);
    }
}

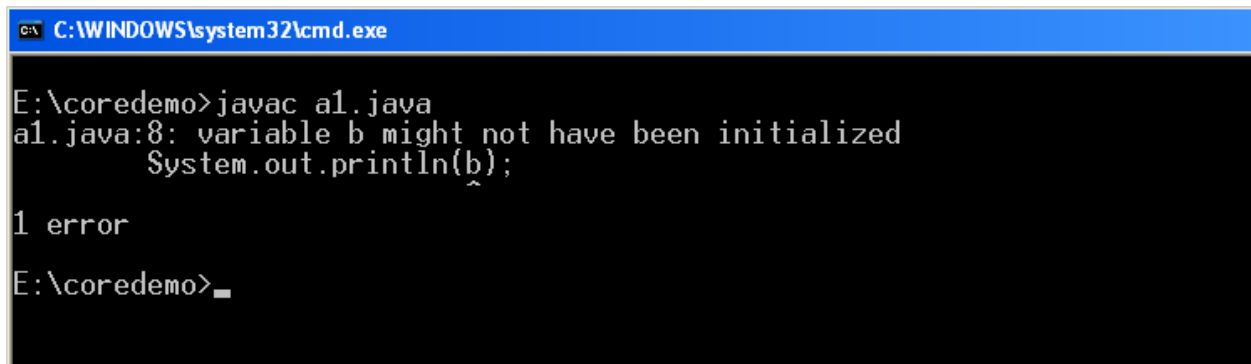
class B
{
public static void main(String args[])
    {
        A ob=new A();//object created
        ob.f1();//calling f1 method
    }
}
```

}

Explanation:

If you are trying to compile the above program it shows a compilation error saying that

Output:-

A screenshot of a Windows command prompt window. The title bar is blue and reads "C:\WINDOWS\system32\cmd.exe". The command prompt shows the following text:

```
E:\coredemo>javac a1.java
a1.java:8: variable b might not have been initialized
    System.out.println(b);
                       ^
1 error
E:\coredemo>_
```

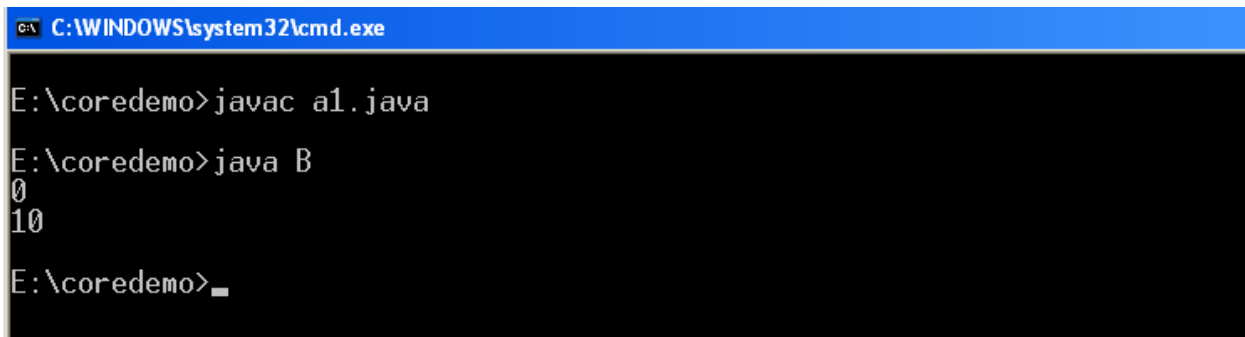
Note:-

- 1) Jvm supplies default values to a class variable.
- 2) Jvm does not supplies any default values to class variable.
- 3) Whenever local variable is declared it is mandatory to assign a value to that variable otherwise compiler gives an error.

In the above program modify at the 1 like as

int b=10 ;

The program is executed we get an output like.....

A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe". The command prompt shows the following sequence of commands and output:

```
E:\coredemo>javac a1.java
E:\coredemo>java B
0
10
E:\coredemo>_
```

Explanation on object creation process:-

In the above program inside the main method we are created the object with the syntax like

A ob=new A();

At this stage object creation process was involved into four steps.....

- 1) New operator allocating memory to variables.
- 2) JVM supplies default values to a class variables.
- 3) A special method constructor is executed.
- 4) New operator returns the object address.

Garbage collection:-

When ever object is no longer in use jvm calls implicitly finalize method to destroy the object due to the presence of memory leakage.

Note:

In java objects was not call methods directly in this case object references were created for every object. object reference is also called as reference variable.

What is the difference between reference variable and object??

Reference:- It is the address of object.

Object: It is instance of an class and instance means it is a representative of a class.

Argument passing:-

It is of two types

- 1) Pass by value
- 2) Pass by reference

Pass by value:

Supplying actual parameter value to the formal parameter is known as pass by value.

Formal parameter:-

The parameters which has been defined in the method signature.

Ex: void f1(int a,float b)



**formal parameters**

Actual parameters:

The parameters which has been defined inside the method calling with the object reference.

Ex : ob.f1(10,20);



**actual parameters**

We see the actual parameter and the formal parameter in the below program.....

```
class A
```

```
{
```

```
int result;
```

```
void f1(int a,int b)
```

```
{
```

```
    result=a+b;
```

```
    System.out.println("Sumis:"+result);
```

```
    }  
}  
  
class B  
{  
    public static void main(String args[])  
    {  
        A ob=new A();  
        ob.f1(10,20);  
    }  
}
```

Output:

E:\coredemo>javac a2.java

E:\coredemo>java B

Sumis:30

We see the pass by reference concept after the constructor concept.....



Method overloading:

Defining more than one method with the same name in a single class with different parameters we say that method is overloading

Program :

```
class A
{
    int a;
    String b;
    void f1()
    {
        System.out.println(a);
        System.out.println(b);
    }
    void f1(float a)
    {
```

```
        System.out.println(a);
    }
void f1(double a,char b)
    {
        System.out.println(a);
        System.out.println(b);
    }
}
class B
{
    public static void main(String args[])
    {
        A ob=new A();
        ob.f1();
    }
}
```

Output:

```
E:\coredemo>javac a3.java
```

```
E:\coredemo>java B
```

0

Null

Constructor:-

It is a special method in java which is used for object creation purpose.

There are two types of constructors in java.....

- 1) Implicit constructor
- 2) Explicit constructor

Implicit constructor:

If you are not providing any constructor in your program compiler supplies default constructor implicitly. Default constructor is also called no-argument constructor.

Explicit constructor:-

It is an explicit constructor provided by our programmer. At this case default constructor was not provided by our compiler why because we are providing constructor in your program explicitly.

Note:

The constructor should be matched to the given class name other wise compiler is doesnot treat as a constructor it gives an compile time error.

Program:

```
class A
{
A()
{
System.out.println("a's constructor");
}
```

```
}  
  
class B  
{  
  
public static void main(String args[])  
  
    {  
  
        A ob=new A();  
  
    }  
  
}
```

Output:

```
E:\coredemo>javac a4.java
```

```
E:\coredemo>java B
```

```
a's constructor
```

Constructor overloading:

Defining more than one constructor in a single class with different parameters we say that constructor is overloading.

Program:

```
class A
{
String a;
boolean b;

A()
{
    System.out.println(a);
    System.out.println(b);
}

A(int a)
{
    System.out.println(a);
}

A(float a,char b)
{
    System.out.println(a);
    System.out.println(b);
}
```

```
    }  
}  
  
class B  
{  
  
    public static void main(String args[])  
    {  
        A ob=new A(20.9f,'a');  
    }  
}
```

Output:

```
E:\coredemo>javac a5.java
```

```
E:\coredemo>java B
```

20.9

A

About this keyword:

It has two functions

- 1) It shows the difference between local variable and class variable to the compiler when they are of same name.
- 2) It calls one constructor to the another constructor.

Program on first function:-

```
class A
{
    int a;

    void f1(int a)
    {
        this.a=a;
    }

    void display()
    {
        System.out.println("Value of a is:"+a);
    }
}
```



```
class B
{
    public static void main(String args[])
    {
        A ob=new A();
        ob.f1(10);
        ob.display();
    }
}
```

Output:

```
E:\coredemo>javac a6.java
```

```
E:\coredemo>java B
```

Value of a is:10

Suppose if your not giving this keyword behind the variable the output is zero why because the compiler confused when the variable is of same name.

Program on this keyword second function:-

```
class A
```

```
{
```

```
    String a;
```

```
    float b;
```

```
A()
```

```
{
```

```
    this(10.9,'a');
```

```
    System.out.println(a);
```

```
    System.out.println(b);
```

```
}
```

```
A(int a)
```

```
{
```

```
    System.out.println(a);
```

```
}
```

```
A(double a,char b)
```

```
    {  
        this(10);  
        System.out.println(a);  
        System.out.println(b);  
    }  
}  
  
class B  
{  
    public static void main(String args[])  
    {  
        A ob=new A();  
    }  
}
```

Output:-

E:\coredemo>javac a7.java

```
E:\coredemo>java B
```

```
10
```

```
10.9
```

```
a
```

```
null
```

```
0.0
```

Introduction to inheritance:

Creating sub class from existing super class is called inheritance.

Method overriding:-

Redefining the functionality of method name with in the super class and in the sub class with same parameters we say that method is overriding.

Program:-

```
class A
```

```
{
```

```
void f1(int a)
```

```
{
```

```
        System.out.println("Super class:"+a);
    }
}

class B extends A
{
    void f1(int a)
    {
        System.out.println("Sub class:"+a);
    }
}

class C
{
    public static void main(String args[])
    {
        B ob=new B();
        ob.f1(10);
    }
}
```

```
}
```

Output:-

```
E:\coredemo>javac a8.java
```

```
E:\coredemo>java C
```

Sub class:10

Note:- In the above program class A is called super class i.e. overridden method and class B is called sub class i.e. overriding method.

b) cyclic inheritance:-

Note:- java does not support multiple inheritance directly means we cannot extends more than one class at a time. So this concept shows we can create number of subclasses to the any class by extending only one class at a time.

Program:-

```
class A
```

```
{
```

```
void f1()
{
    System.out.println("a's class");
}
}
```

```
class B extends A
```

```
{
void f2()
{
    System.out.println("b's class");
}
}
```

```
class C extends B
```

```
{
void f3()
{
    System.out.println("c's class");
}
```

```
    }  
}  
  
class cyclic  
{  
  
    public static void main(String args[])  
    {  
  
        C ob=new C();  
  
        ob.f1();  
  
        ob.f2();  
  
        ob.f3();  
  
    }  
  
}
```

Output:-

E:\coredemo>javac a9.java

E:\coredemo>java cyclic

a's class

b's class



c's class

use of super keyword:-

It has two functions.....

- 1) It calls super class methods from the sub class method.
- 2) It calls super class constructor from the sub class constructor.

Example program on super keyword first use:-

```
class A
{
void f1()
{
    System.out.println("super class");
}
}
```

```
class B extends A
{
void f1()
    {
        super.f1();
        System.out.println("sub class");
    }
}

class C
{
public static void main(String args[])
    {
        B ob=new B();
        ob.f1();
    }
}
```

Output:-

```
E:\coredemo>javac a10.java
```

```
E:\coredemo>java C
```

super class

sub class

Constructors in inheritance:-

Example program on super keyword second use:-

```
class A
```

```
{
```

```
A(int a)
```

```
{
```

```
    System.out.println("value of a is:"+a);
```

```
}
```

```
}
```

```
class B extends A
```

```
{
```

```
B()
```

```
{
```

```
        System.out.println("b's constructor");
    }
}

class C
{
    public static void main(String args[])
    {
        B ob=new B();
    }
}
```

Output:-

```
E:\coredemo>javac a11.java
a11.java:11: cannot find symbol
symbol   : constructor A()
location: class A
    {
```

^

1 error

Note:-

The above program shows a compilation error why because at the time of object creation super class default constructor and sub class default constructor should be available to the compiler other wise object creation fails. In the above program your not providing any default constructor in class A so the compiler throws an error default constructor is missing in class A. To solve the above problem we have two solutions.....

- 1) We have to provide the default constructor in class A
- 2) By the use of super keyword

Example program on first solution:-

```
class A
```

```
{
```

```
A()
```

```
{
```

```
        System.out.println("a's constructor");
    }
A(int a)
    {
        System.out.println("value of a is:"+a);
    }
}
class B extends A
{
    B()
    {
        System.out.println("b's constructor");
    }
}
class C
{
    public static void main(String args[])
```

```
{  
    B ob=new B();  
}  
}
```

Output:-

```
E:\coredemo>javac a11.java
```

```
E:\coredemo>java C
```

a's constructor

b's constructor

Example program on second solution:-

```
class A  
{  
    A(int a)  
    {  
        System.out.println("value of a is:"+a);  
    }  
}
```

```
class B extends A
{
    B()
    {
        super(10);
        System.out.println("b's constructor");
    }
}

class C
{
    public static void main(String args[])
    {
        B ob=new B();
    }
}
```

Output:-

```
E:\coredemo>javac a12.java
```



```
E:\coredemo>java C
```

```
value of a is:10
```

```
b's constructor
```

Dynamic method dispatch:-

Whenever method overriding is implemented super class reference is given to the sub class object this mechanism is called dynamic method dispatch.

Program:-

```
class A
```

```
{
```

```
void f1()
```

```
{
```

```
    System.out.println("a's class");
```

```
}
```

```
}
```

```
class B extends A
```

```
{
```

```
void f1()
{
    System.out.println("b's class");
}

class C
{
    public static void main(String args[])
    {
        //create the reference to the super class
        A ob;

        //pass the above ob reference to the sub class
        ob=new B();//dynamic dispatch
        ob.f1();
    }
}
```

Output:-

```
E:\coredemo>javac a13.java
```

```
E:\coredemo>java C
```

b's class

Access specifiers:-

There are four types.....

- 1) Public
- 2) Private
- 3) Protected
- 4) Default

Public:-

Whenever we declare class and methods as public we can access at anywhere in java environment.

Private :-

Whenever we declare variables and methods as private it can be accessible only in that same class only.

Protected:-

Whenever we declare variables and method as protected it can be accessible only for the subclasses and in the same package.

We see the examples of public and protected in the next concept package.

Example program on use of private:-

```
class A
{
    private int a=10;
    private void f1()
    {
        System.out.println("value of a is:"+a);
    }
}

class B
{
    public static void main(String args[])
    {
        A ob=new A();
```

```
        System.out.println(ob.a);  
        ob.f1();  
    }  
}
```

Output:

```
E:\coredemo>javac a14.java
```

```
a14.java:14: a has private access in A
```

```
        System.out.println(ob.a);
```

```
                ^
```

```
a14.java:15: f1() has private access in A
```

```
        ob.f1();
```

```
            ^
```

2 errors

Note:-

The above program shows a compilation error why because private members cannot be accessed from the outside class.

## Packages:-

It is one of the most important concept in java.

Def: collection of class files grouping into a single folder is called package.

There are two types of packages in java.....

- 1) Standard defined package
- 2) User defined package

### Standard defined package:-

Predefined packages are known as standard defined packages. In java these packages are used in our program with a keyword called import.

Ex:- `import java.lang.*;`

It is a default package which is provided by our compiler implicitly.

Where \* means set of class files.

List of some important standard defined packages:-

- 1) `import java.io.*;`

- 2) `import java.net.*;`
- 3) `import java.util.*;`
- 4) `import java.sql.*;`

User defined packages:-

Our own created packages are known as user defined packages.

There are four types in creation of user defined package.....

Type 1:-

Steps to create a package program:-

- 1) Packages are created in our program with a keyword called package.

Syntax: `package packagename;`

- 2) In the package program the class and methods should be declare as public.

- 3) We have to compile the package program with the special command as shown below

Syntax:- `javac -d directoryname filename.java`

Here `-d` → it indicates the compiler on which directory the class file has to be saved.

By using the above steps we have to create a package program as shown below.....

```
// package program
```

```
//save this file as A.java
```

```
package nit;
```

```
public class A
```

```
{
```

```
public void f1()
```

```
{
```

```
    System.out.println("Hi your are using the package");
```

```
}
```

```
}
```

Note:- whenever we give the public to the class the class name must match to the file name otherwise it gives a compilation error.



Steps to create a main program:-

- 1) We have to import our package program into our main program.
- 2) We have to set the class path to the class loader for the location of package program.

```
//main program
```

```
class usepack
```

```
{
```

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

```
{
```

```
    nit.A ob=new nit.A();
```

```
    ob.f1();
```

```
}
```

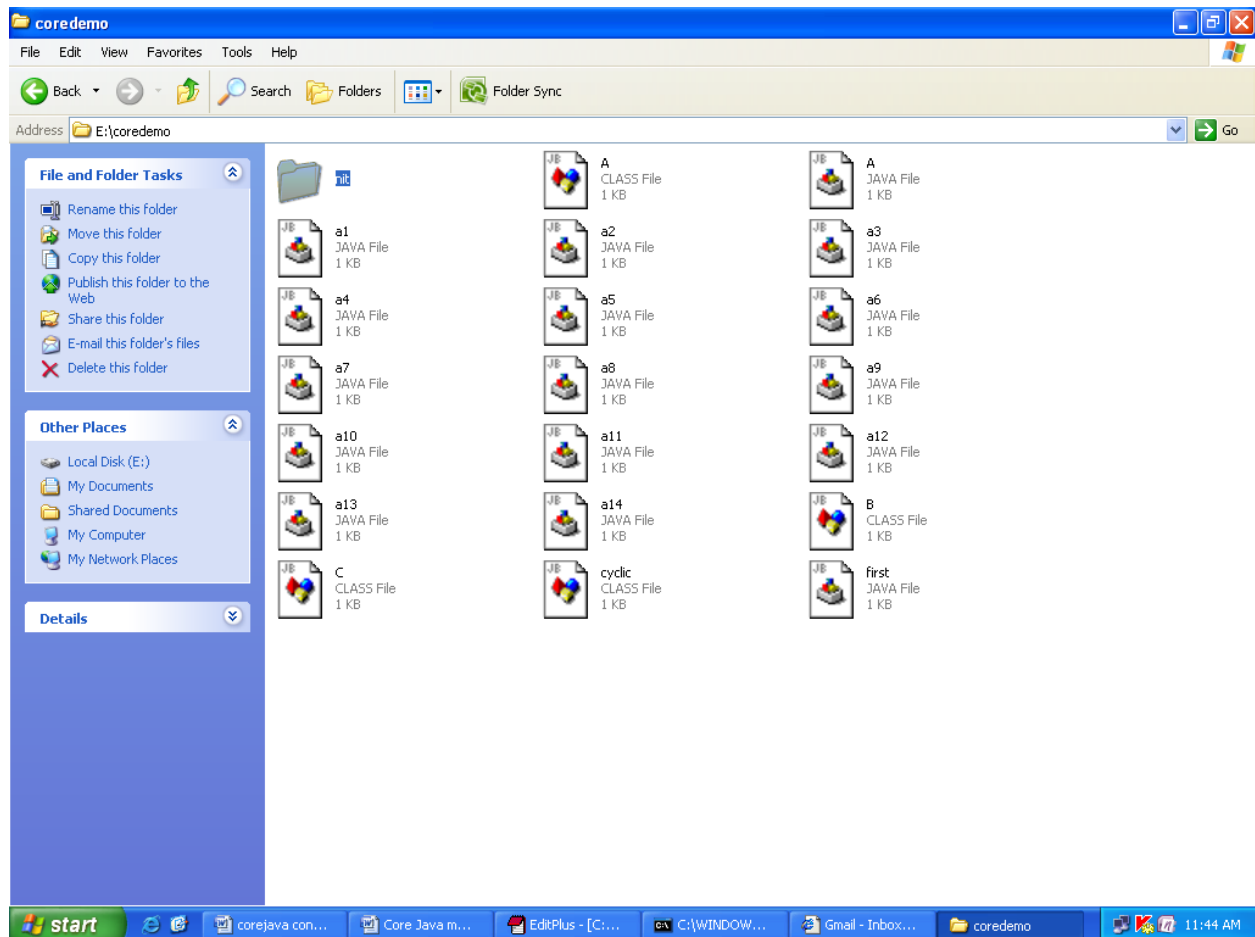
```
}
```

Compile the package program:-

```
E:\coredemo>javac -d . A.java
```

Here in above command “.” Means current working directory.

After compiling the package program the package nit folder is created inside the “coredemo” directory as shown below.....



Compile and execute the main program:-

Suppose this program is save in c drive so change the location to c drive as shown below.....

```
E:\coredemo>c:
C:\Documents and Settings\Administrator>cd\
C:\>set classpath=E:\coredemo;.
C:\>javac usepack.java
C:\>java usepack
Hi your are using the package
C:\>
```

Type 2:-

In this we are creating addition and subtraction package programs as shown below.....

//Addition.java

package nit.sai;

public class Addition

{

int result;

public void add(int a,int b)

{

result=a+b;

```
        System.out.println("Sumis:"+result);  
    }  
}
```

//Subtraction.java

```
package nit.sai;  
  
public class Subtraction  
{  
  
    int result;  
  
    public void sub(int a,int b)  
    {  
  
        result=a-b;  
  
        System.out.println("subis:"+result);  
  
    }  
  
}
```

After that create a main program by using the above two packages.....

```
import nit.sai.Addition;
```

```
import nit.sai.Subtraction;

class usepack
{
    public static void main(String args[])
    {
        Addition ob=new Addition();
        Subtraction ob1=new Subtraction();
        ob.add(10,20);
        ob1.sub(50,10);
    }
}
```

After that compile and execute the above programs as shown below.....

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

E:\coredemo>javac -d . Addition.java
E:\coredemo>javac -d . Subtraction.java
E:\coredemo>c:
C:\>javac usepack.java
C:\>java usepack
Sumis:30
subis:40
C:\>
```

Type 3:-

//Addition.java

package nit.sai;

public class Addition

{

int result;

public void add(int a,int b)

{

result=a+b;

System.out.println("Sumis:"+result);

}

}

```
//Subtraction.java  
  
package nit.sai;  
  
public class Subtraction  
{  
  
    int result;  
  
    public void sub(int a,int b)  
    {  
        result=a-b;  
        System.out.println("subis:"+result);  
    }  
}
```

Note:- Type 3 says that if you want to import the two class files which is available in the same package then we can give \* after defining the package command as shown in the below program.....

```
//usepack.java  
  
import nit.sai.*;  
  
class usepack
```

```
{  
public static void main(String args[])  
  
    {  
        Addition ob=new Addition();  
        Subtraction ob1=new Subtraction();  
        ob.add(10,20);  
        ob1.sub(50,10);  
    }  
}
```

Note:- Before compiling the usepack program make sure that the addition and subtraction java file should be compiled and after that the source code should not be available in coredemo directory. If do so it gives a compilation error.....so put this two source code files in other directory as shown below.....



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

E:\sai>tree/f
Folder PATH listing
Volume serial number is 0000CB68 F854:5F50
E:.
    Addition.java
    Subtraction.java

No subfolders exist

E:\sai>
```

After that compile and execute the usepack.java on above procedure.....as shown in below fig.....

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

E:\coredemo>set classpath=E:\coredemo;.
E:\coredemo>c:
C:\>javac usepack.java
C:\>java usepack
Sumis:30
subis:40
C:\>
```

Type 4:-

If you define any variable and method as protected we can access them by two ways

- 1) By inheritance mechanism

2) By defining the same package name to our main program

First way:-

```
//Addition.java
package nit.sai;
public class Addition
{
    int result;
    protected void add(int a,int b)
    {
        result=a+b;
        System.out.println("Sum is:"+result);
    }
}
```

```
//Subtraction.java
package nit.sai;
public class Subtraction
{
    int result;
    public void sub(int a,int b)
```

```
    {  
        result=a-b;  
        System.out.println("Sub is:"+result);  
    }  
}
```

```
//usepack.java  
import nit.sai.Addition;  
import nit.sai.Subtraction;  
class usepack extends Addition  
{  
    public static void main(String args[])  
    {  
        usepack ob=new usepack();  
        Subtraction ob1=new Subtraction();  
        ob.add(10,20);  
        ob1.sub(50,10);  
    }  
}
```

Compiling and executing the above programs as shown in below fig.....

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

E:\coredemo>javac -d . Addition.java
E:\coredemo>javac -d . Subtraction.java
E:\coredemo>c:
C:\>set classpath=E:\coredemo;.
C:\>javac usepack.java
C:\>java usepack
Sum is:30
Sub is:40
C:\>
```

Second way:-

In the second way iam giving the same package name to the main program i.e to usepack.java

//Addition.java

package nit.sai;

public class Addition

{

int result;

protected void add(int a,int b)

{

result=a+b;

System.out.println("Sum is:"+result);

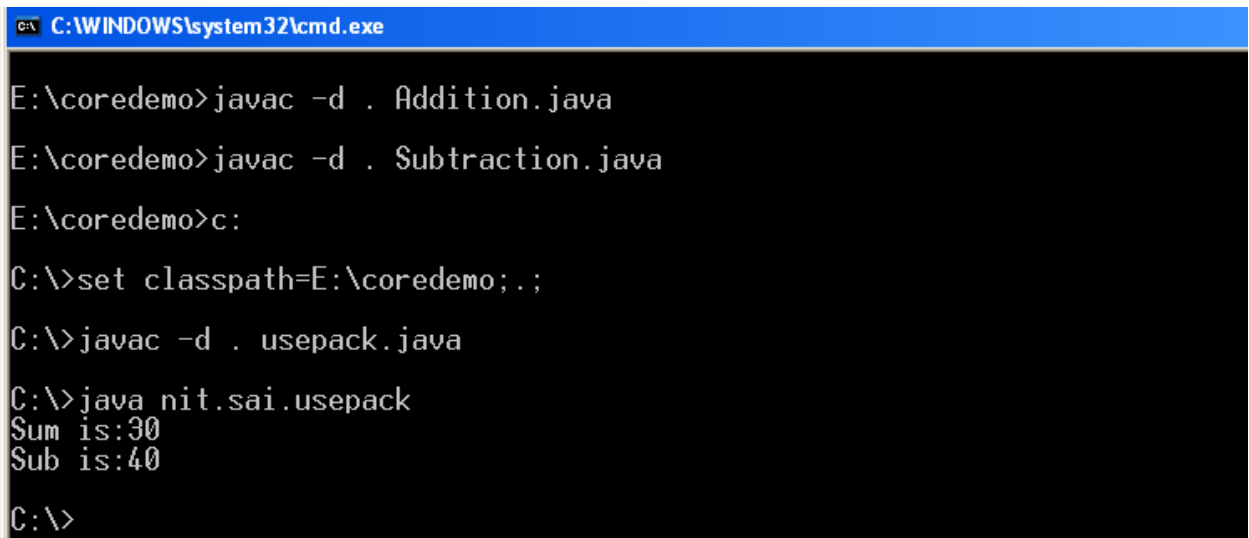
}

```
}  
//Subtraction.java  
package nit.sai;  
public class Subtraction  
{  
    int result;  
    public void sub(int a,int b)  
    {  
        result=a-b;  
        System.out.println("Sub is:"+result);  
    }  
}
```

```
//usepack.java  
package nit.sai;  
import nit.sai.Addition;  
import nit.sai.Subtraction;  
class usepack  
{  
    public static void main(String args[])  
    {  
        usepack ob=new usepack();  
        Subtraction ob1=new Subtraction();  
    }  
}
```

```
        ob.add(10,20);
        ob1.sub(50,10);
    }
}
```

Compiling and executing the above programs as shown in below fig.....



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

E:\coredemo>javac -d . Addition.java
E:\coredemo>javac -d . Subtraction.java
E:\coredemo>c:
C:\>set classpath=E:\coredemo;.
C:\>javac -d . usepack.java
C:\>java nit.sai.usepack
Sum is:30
Sub is:40
C:\>
```

1. In the above fig we are compiling the main program with package syntax why because we defined the main program is also package.
2. After that we have to execute the main program. In this case we have to give the exact class file location to the jvm. So that our usepack class file was located in c:\nit\sai directory. So we have to

give the above all location to the jvm as shown in above fig.....

Type casting:-

Casting means converting one value into another value.

In java there are two types of conversions.....

- 1) Widening conversion
- 2) Narrowing conversion

Widening conversion:-

It is an implicit conversion done by the compiler automatically. Converting lower value to higher value is known as widening conversion.

Narrowing conversion:-

It is an explicit conversion done by the programmer. Converting higher value to lower value is known as narrowing conversion.

Before performing the casting operations we have to know the data type order (smaller to higher) as shown below.....

- 1. Byte**
  - 2. short**
  - 3. Char**
  - 4. Int**
  - 5. Long**
  - 6. Float**
  - 7. double**
- 

Program 1:-

```
class A
{
//widening conversion
void f1(int a)
{
    float b=a;
    System.out.println(b);
}

//narrowing conversion
void f2(float a)
{
```



```
        int b=(int)a;
        System.out.println(b);
    }
}
class casting
{
    public static void main(String args[])
    {
        A ob=new A();
        ob.f1(10);
        ob.f2(100.9f);
    }
}
```

Output:-

E:\coredemo>javac a15.java

E:\coredemo>java casting

10.0

100

Note: The drawback in narrowing conversion is fraction part was truncated.

Program 2:-

```
class A
{
//widening conversion
void f1(char a)
    {
        int b=a;
        System.out.println(b);
    }
//narrowing conversion
void f2(int a)
    {
        char b=(char)a;
        System.out.println(b);
    }
}

class casting1
{
public static void main(String args[])
    {
        A ob=new A();
        ob.f1('A');
```

```
        ob.f2(68);  
    }  
}
```

Output:-

```
E:\coredemo>javac a16.java
```

```
E:\coredemo>java casting1
```

65

D

Note:-

In the above program when ever we are converting char to int “ASCII” code is provided for every character.

Wrapper classes:-

If you want to convert data type value into string value widening and narrowing conversions failed in this case, because the string and Boolean or advanced data types. So advanced types was not performed in casting because it hold multiple values for example in String we can pass

either numbers and characters also so that type of dual conversion was not allowed.

To solve this problem wrapper class was developed.

It has two functions

- 1) It converts string value into any data type value except character.
- 2) It converts data type value into wrapped object.

First function:-

In this below program we are converting String value into int value for this case Integer wrapper class is available. By using this we convert that string value into data type value and also see the below table for different wrapper classes available for different data types.....

Data types	wrapper classes
int	Integer
float	Float
char	Character
double	Double

Program:-

```
class A
```

```
{
```

```
void f1(String a)
```

```
{
```

```
int b=Integer.parseInt(a);
```

```
System.out.println(b);
```

```
}
```

```
}
```

```
class B
```

```
{
```

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

```
{
```

```
A ob=new A();
```

```
ob.f1("10");
```

```
}
```

```
}
```

Output:-

```
E:\coredemo>javac a17.java
```

```
E:\coredemo>java B
```

10

Second function:-

The second function was developed for the collection framework we discuss this concept later.....

In the collection framework there are some predefined collection classes was available suppose we want to pass the data type value to those collection class it was not accepted why because it take only objects in order to convert that data type value into object wrapper class introduced another mechanism called wrapping.

Wrapping: It is a process of converting data type value into wrapped object value.

Program:-

```
class A
```

```
{
```

```
void f1()
```

```
{  
    Integer i=new Integer(10);//wrapped the 10 value  
    into i object
```

```
    int j=i.intValue();
```

```
    /*intValue() is the predefined  
    static method which is available  
    inside the Integer wrapper class  
    it converts wrapped object value into  
    data type value*/
```

```
    System.out.println(j);
```

```
}
```

```
}
```

```
class B
```

```
{
```

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

```
{
```

```
    A ob=new A();
```

```
        ob.f1();  
    }  
}
```

Output:-

```
E:\coredemo>javac a18.java
```

```
E:\coredemo>java B
```

10

Command line arguments:-

Values supplied to the main method from the command line is known as command line arguments.

Program:-

```
class A  
{  
    int result;  
    void f1(int a,int b)  
    {  
        result=a+b;
```

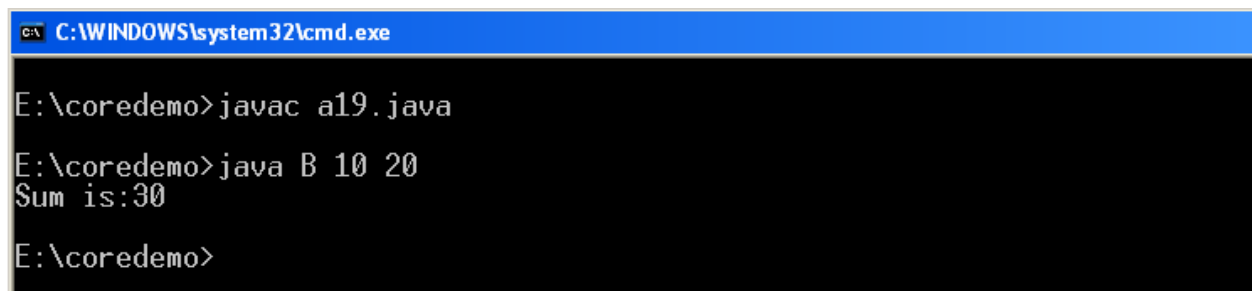


```
        System.out.println("Sum is:"+result);
    }
}

class B
{
    public static void main(String args[])
    {
        A ob=new A();

        int i=Integer.parseInt(args[0]);
        int j=Integer.parseInt(args[1]);
        ob.f1(i,j);
    }
}
```

Output:-



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

E:\coredemo>javac a19.java
E:\coredemo>java B 10 20
Sum is:30
E:\coredemo>
```

Note: If you want to get the values from the runtime itself we are using this kind of mechanism.

Arrays:-

Arrays is nothing but collection of homogenous(similar) elements referred by the same name is known as array.

Ex:- `int arr[]={10,20,30.....n};`

We can also supply n number of values to single variable as shown above.....

There are two types of arrays in java.....

- 1) Primitive defined array
- 2) User defined array

Primitive defined array:-

In this type array objects are created internally.

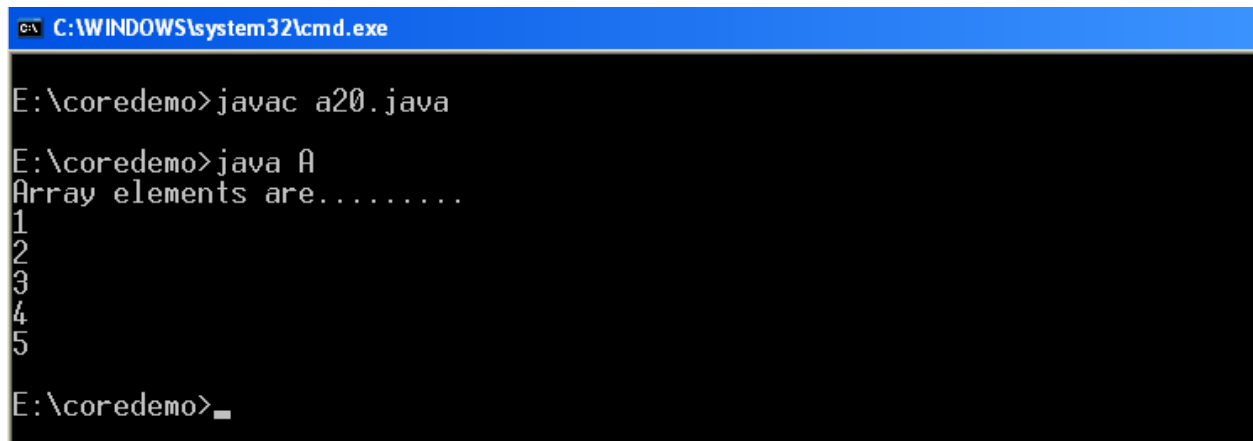
Program:-

class A

```
{  
public static void main(String args[])  
  
    {  
  
        int arr[]=new int[5];  
  
        /* upto 5 memory allocations are created  
        for array arr*/  
  
        //array index range was started from 0 onwards  
  
        /*length is the predefined property in java  
        to calculate the array length exactly */  
  
        System.out.println("Array elements are.....");  
  
        for(int i=0;i<arr.length;i++)  
  
            {  
  
                //for storing the values in array  
  
                arr[i]=i+1;  
  
                //we have to get the stored values  
  
                System.out.println(arr[i]);  
  
            }  
    }  
}
```

```
    }  
}
```

Output:-



```
C:\WINDOWS\system32\cmd.exe  
E:\coredemo>javac a20.java  
E:\coredemo>java A  
Array elements are.....  
1  
2  
3  
4  
5  
E:\coredemo>_
```

User defined array:-

In this type array references are created internally.

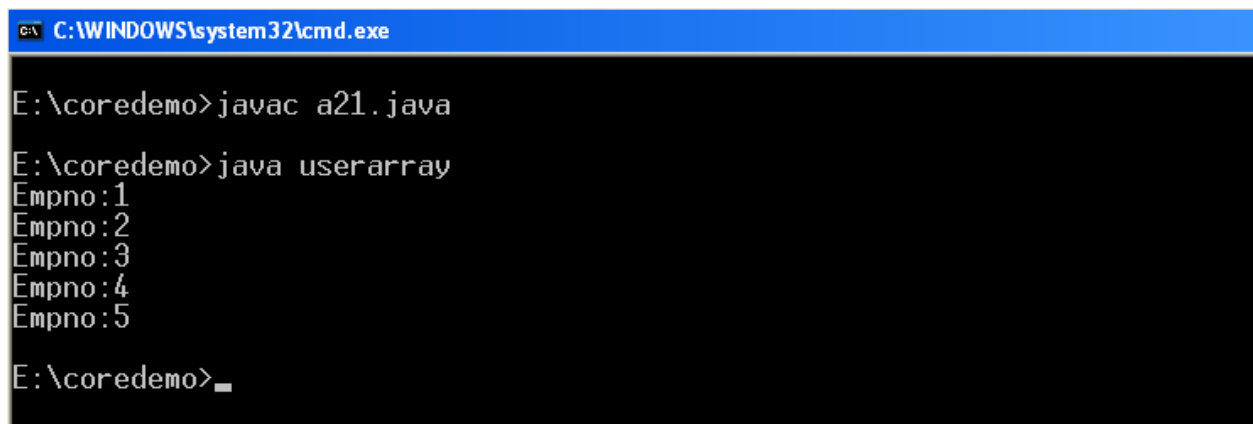
Program:-

```
class Employee  
{  
  
int empno;  
  
Employee(int empno)  
{
```

```
        this.empno=empno;
    }
void display()
    {
        System.out.println("Empno:"+empno);
    }
}
class userarray
{
public static void main(String args[])
    {
        Employee e[]=new Employee[5];
        for(int i=0;i<e.length;i++)
            {
                /* in the below line at a time we are
                invoking the constructor and loading
                the value in the array*/
```

```
        e[i]=new Employee(i+1);  
        /*To get the values call with the  
        array reference*/  
        e[i].display();  
    }  
}  
}
```

Output:-



```
C:\WINDOWS\system32\cmd.exe  
E:\coredemo>javac a21.java  
E:\coredemo>java userarray  
Empno:1  
Empno:2  
Empno:3  
Empno:4  
Empno:5  
E:\coredemo>_
```

Access modifiers:-

There are three types

- 1)Static
- 2)Final
- 3)Abstract

Static :-

Whenever we declare variable and method as static without object creation we can call directly with that class name.

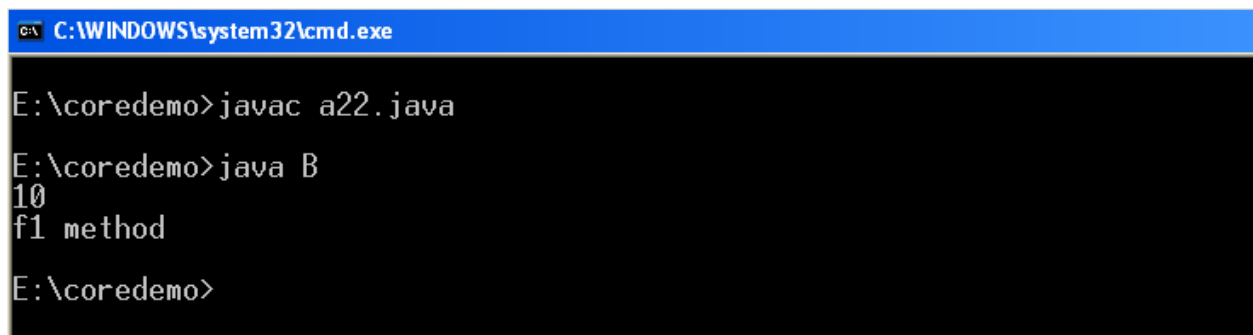
Program:-

```
class A
{
    static int a=10;
    static void f1()
    {
        System.out.println("f1 method");
    }
}

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

```
{  
    System.out.println(A.a);  
    A.f1();  
}  
}
```

Output:-

A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe". The command prompt shows the following sequence of commands and output:  
E:\coredemo>javac a22.java  
E:\coredemo>java B  
10  
f1 method  
E:\coredemo>

Note:-

Whenever class is loaded into memory whatever we declare with static one copy instance is created dynamically.

Q) can we execute our program without main method or not???

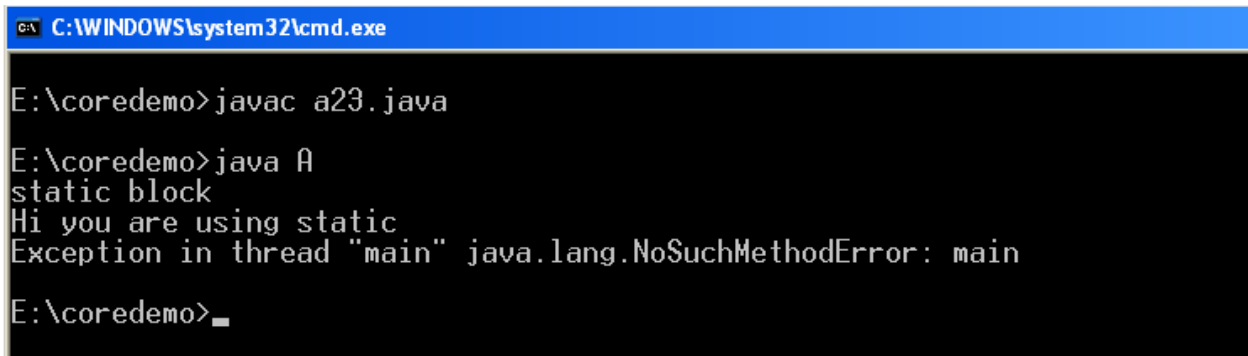
A) yes we can execute our program using static block.



Example program on use of static block:-

```
class A
{
    static void f1()
    {
        System.out.println("Hi you are using static");
    }
    static
    {
        System.out.println("static block");
        f1();
    }
}
```

Output:-

A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe". The command prompt shows the following sequence of commands and output:

```
E:\coredemo>javac a23.java
E:\coredemo>java A
static block
Hi you are using static
Exception in thread "main" java.lang.NoSuchMethodError: main
E:\coredemo>_
```

Final:-

Whenever we declare variable as final the value cannot be changed programmatically.

Program:-

```
class A
{
    final int a=10;

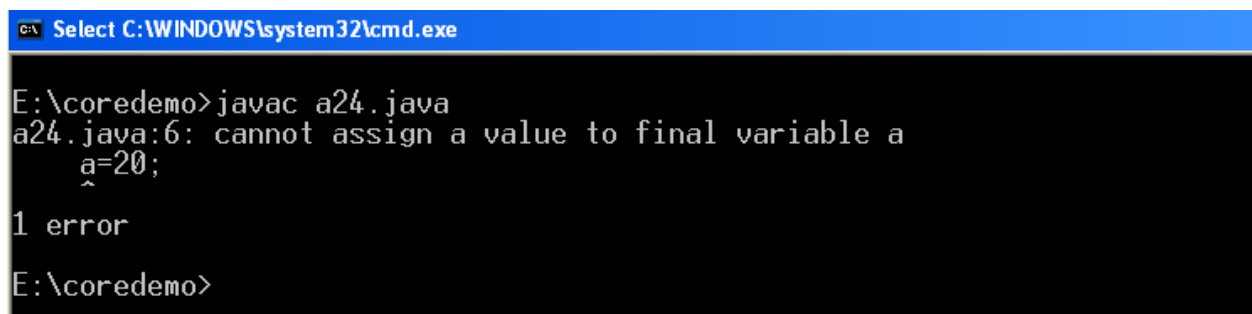
    void f1()
    {
        a=20;

        System.out.println("Value of a is:"+a);
    }
}
```

```
}  
  
class B  
{  
  
public static void main(String args[])  
  
    {  
  
        A ob=new A();  
  
        ob.f1();  
  
    }  
  
}
```

The above program shows a compilation error because we cannot reinitialize a value to a final variable as shown in below fig.....

Output:-



```
C:\ Select C:\WINDOWS\system32\cmd.exe  
  
E:\coredemo>javac a24.java  
a24.java:6: cannot assign a value to final variable a  
    a=20;  
    ^  
1 error  
  
E:\coredemo>
```

Abstract:-

A class represents a base class which another types to be a subtype of a class.

- 1) Whenever we declare method as abstract it has no implementation(body not be provided) just definition only.
- 2) Whenever we declare method as abstract it is mandatory to declare the class also a abstract.
- 3) Object was not created for abstract class.
- 4) In abstract class we can also provide concrete methods and constructors also.
- 5) In abstract class we can have zero or more abstract methods.

Q) what is the use of abstract class?

A) There is sometimes you will want to create a super class that only defines a generalized form that will be shared all of its sub classes, leaving it to each sub class to fill in details. Such a class determines the nature of methods that the sub classes must implement. One way

this situation occur is when a super class is unable to create a meaning full implementation. This implementation is filled up by each sub class in detail.....

Program:-

```
abstract class Figure
{
    abstract void area();
}

class Rectangle extends Figure
{
    int l;
    int w;
    int result;

    Rectangle(int l,int w)
    {
        this.l=l;
        this.w=w;
    }
}
```

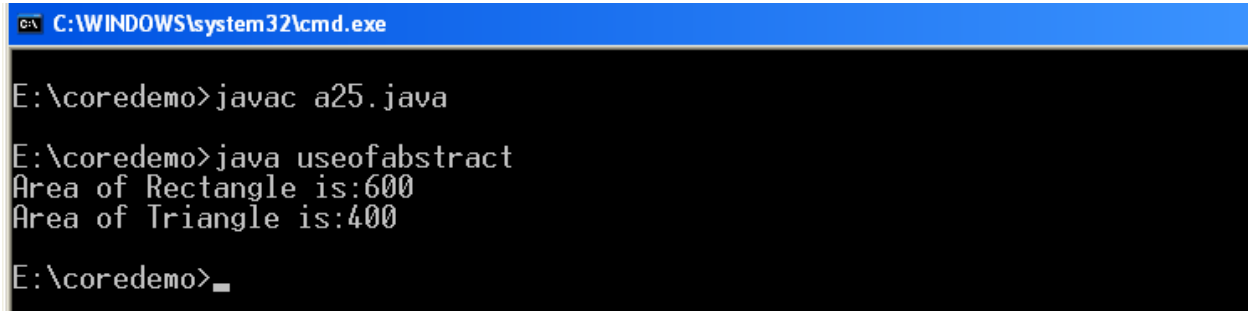
```
    }  
void area()  
{  
    result=l*w;  
    System.out.println("Area of Rectangle is:"+result);  
}  
}  
class Triangle extends Figure  
{  
    int b;  
    int h;  
    int result;  
    Triangle(int b,int h)  
    {  
        this.b=b;  
        this.h=h;  
    }  
}
```

```
void area()
{
    result=(b*h)/2;
    System.out.println("Area of Triangle is:"+result);
}

}

class useofabstract
{
    public static void main(String args[])
    {
        Rectangle rec=new Rectangle(20,30);
        Triangle tr=new Triangle(40,20);
        rec.area();
        tr.area();
    }
}
```

Output:-

A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe". The command prompt shows the following sequence of commands and output:

```
E:\coredemo>javac a25.java
E:\coredemo>java useofabstract
Area of Rectangle is:600
Area of Triangle is:400
E:\coredemo>_
```

Interface:-

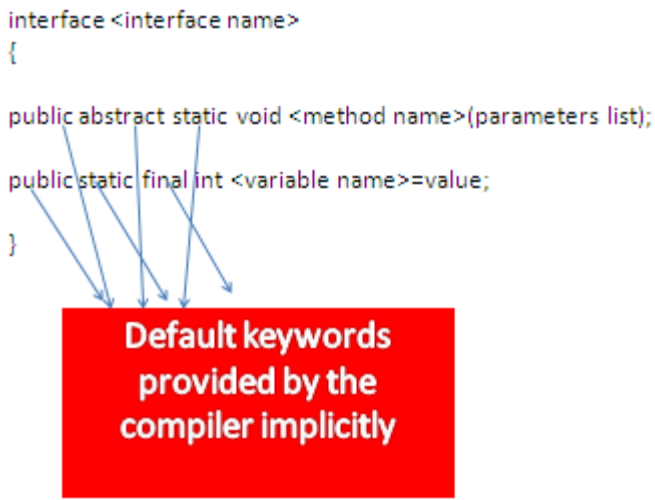
- 1) In the most common form, an interface is a group of related methods with empty bodies.
- 2) Implementing an interface allows a class to become more formal about the behavior it promises to provide.
- 3) Interfaces form a contract between the class and the outside world, and this contract is enforced at build time by the compiler.
- 4) If your class claims to implement an interface, all methods defined by that interface must appear in its source code before the class will successfully compile.
- 5) One interface can extend another interface also



6) Object was not created for interface only reference will be created.

Syntax of interface:-

```
interface <interface name>
{
    public abstract static void <method name>(parameters list);
    public static final int <variable name>=value;
}
```



Default keywords  
provided by the  
compiler implicitly

Program:-

```
interface CarDetails
{
    void setName(String a);
    String getName();
    void setMielage(int b);
    int getMielage();
    void setGears(int c);
}
```

```
int getGears();  
}  
  
class Alto implements CarDetails  
{  
    String a;  
    int b,c;  
    public void setName(String a)  
    {  
        this.a=a;  
    }  
    public String getName()  
    {  
        return a;  
    }  
    public void setMielage(int b)  
    {  
        this.b=b;
```

```
    }  
    public int getMielage()  
    {  
        return b;  
    }  
    public void setGears(int c)  
    {  
        this.c=c;  
    }  
    public int getGears()  
    {  
        return c;  
    }  
}  
class Santro implements CarDetails  
{  
    String a;
```

```
    int b,c;  
    public void setName(String a)  
    {  
        this.a=a;  
    }  
    public String getName()  
    {  
        return a;  
    }  
    public void setMielage(int b)  
    {  
        this.b=b;  
    }  
    public int getMielage()  
    {  
        return b;  
    }
```

```
public void setGears(int c)
```

```
{
```

```
    this.c=c;
```

```
}
```

```
public int getGears()
```

```
{
```

```
    return c;
```

```
}
```

```
}
```

```
class useofinterface
```

```
{
```

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

```
    {
```

```
        Alto al=new Alto();
```

```
        al.setName("Alto");
```

```
        al.setMielage(13);
```

```
        al.setGears(5);
```

```
        System.out.println("Car Details are.....");  
        System.out.println("Car name is:"+al.getName());  
        System.out.println("Car mielage  
is:"+al.getMielage());  
  
        System.out.println("Car gears is:"+al.getGears());  
        System.out.println("-----");  
        Santro st=new Santro();  
        st.setName("santro");  
        st.setMielage(15);  
        st.setGears(4);  
  
        System.out.println("Car Details are.....");  
        System.out.println("Car name is:"+st.getName());  
        System.out.println("Car mileage  
is:"+st.getMielage());  
  
        System.out.println("Car gears is:"+st.getGears());  
    }  
}
```

Output:-

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

E:\coredemo>javac a26.java

E:\coredemo>java useofinterface
Car Details are.....
Car name is:Alto
Car mielage is:13
Car gears is:5
-----
Car Details are.....
Car name is:santro
Car mielage is:15
Car gears is:4

E:\coredemo>_
```

Note:-

When you implement an interface method it must declared as public.

Nested interface:-

An interface can be declared a member of a class or another interface. Such an interface called a member interface or nested interface.

Program:-

```
class A
{
```

```
public interface NestedIf
{
    boolean isNotNegative(int x);
}

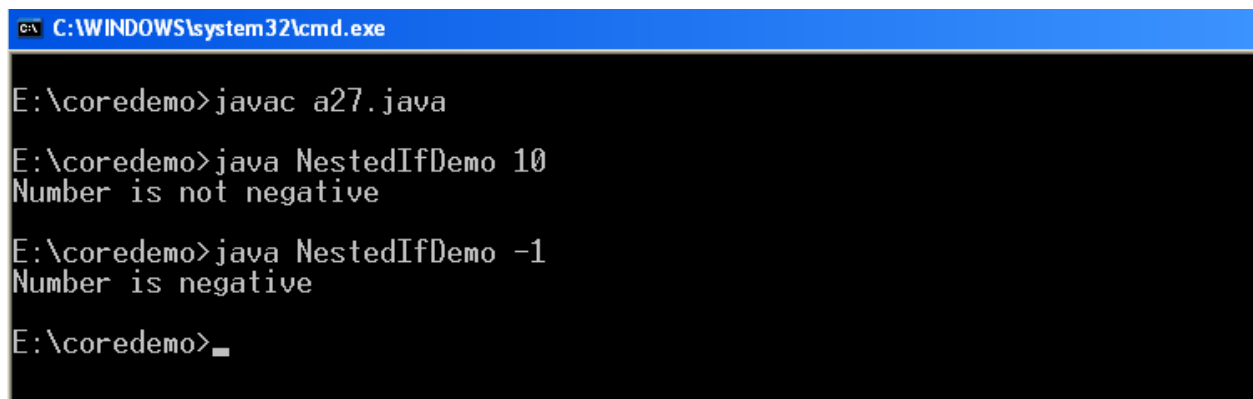
class B implements A.NestedIf
{
    public boolean isNotNegative(int x)
    {
        if(x<0)
            return false;
        else
            return true;
    }
}

class NestedIfDemo
{
```



```
public static void main(String args[])
{
    //use a nested interface reference
    A.NestedIf nif=new B();
    int no=Integer.parseInt(args[0]);
    if(nif.isNotNegative(no)==true)
        System.out.println("Number is not negative");
    else
        System.out.println("Number is negative");
}
}
```

Output:-



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

E:\coredemo>javac a27.java

E:\coredemo>java NestedIfDemo 10
Number is not negative

E:\coredemo>java NestedIfDemo -1
Number is negative

E:\coredemo>_
```

Interfaces can be extended:-

One interface can inherit another by use of the keyword extends. The syntax is same as for inheriting classes.

Program:-

```
interface A
```

```
{
```

```
void f1();
```

```
int a=10;
```

```
}
```

```
interface B extends A
```

```
{
```

```
void f2();
```

```
int b=20;
```

```
}
```

```
class C implements B
```

```
{
```

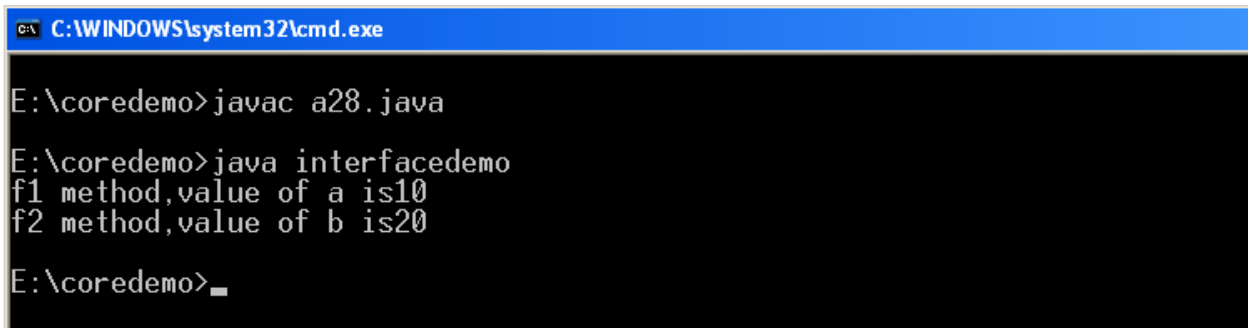
```
public void f1()
{
    System.out.println("f1 method,"+"value of a
is"+A.a);
}

public void f2()
{
    System.out.println("f2 method,"+"value of b
is"+B.b);
}
}

class interfacedemo
{
    public static void main(String args[])
    {
        C ob=new C();
        ob.f1();
    }
}
```

```
        ob.f2();  
    }  
}
```

Output:-



```
C:\WINDOWS\system32\cmd.exe  
E:\coredemo>javac a28.java  
E:\coredemo>java interfacedemo  
f1 method,value of a is10  
f2 method,value of b is20  
E:\coredemo>_
```

What is marker interface???

Sol:- The marker interface is a interface which has no members means variables and methods.

What is a concrete method???

Sol:- The concrete method is a method which has a body.  
Non abstract method is also a concrete method.

Nested classes:-

Defining one class in another class is called nested classes.

In java there are two types.....

- 1) Static nested class
- 2) Non static nested class(inner class)

Static nested class:-

A static nested class is one that has the static modifier is applied. Because it is static it must access the members of its enclosing class through the object. That is it cannot refers to members of its enclosing class directly. Because of this restriction static nested classes are rarely used.

Non static nested class:-

The most important type of nested class is inner class. A inner class is a non static nested class. It has access to all of the variables and methods of its outer class and may refer to them directly in the same way that other non static members of the outer class do.

Example program of non static nested class:-

```
class Outer  
{  
    int i=10;
```

```
void f1()
{
    Inner i=new Inner();
    i.f2();
    System.out.println("Outer class");
}

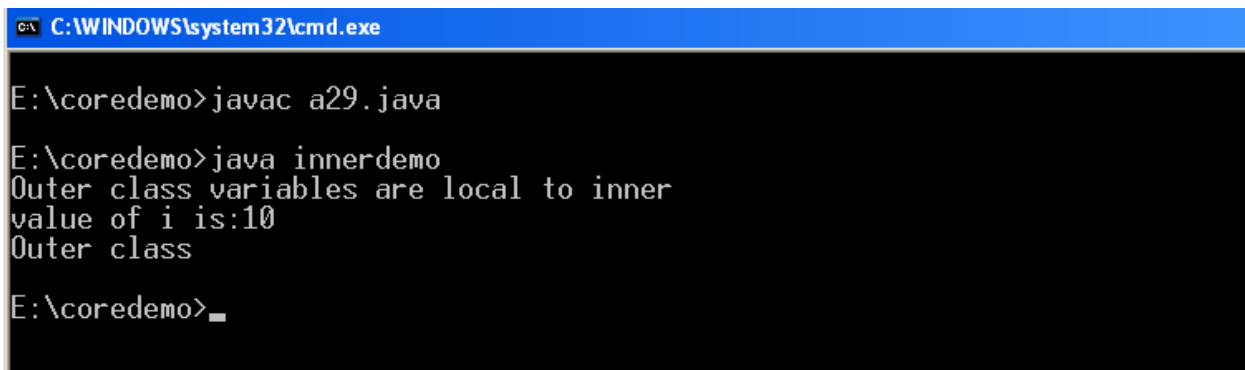
class Inner
{
    void f2()
    {
        System.out.println("Outer class variables are
local to inner");
        System.out.println("value of i is:"+i);
    }
}

}

class innerdemo
```

```
{  
  
public static void main(String args[])  
  
    {  
  
        Outer ot=new Outer();  
  
        ot.f1();  
  
    }  
  
}
```

Output:-

A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe". The command prompt shows the following sequence of commands and output:  
E:\coredemo>javac a29.java  
E:\coredemo>java innerdemo  
Outer class variables are local to inner  
value of i is:10  
Outer class  
E:\coredemo>\_

Exception handling:-

An exception is an abnormal event that occurs during the program execution.

There are two types of exceptions in java

- 1) Standard defined exception
- 2) User defined exception

Standard defined exception:-

Predefined exceptions are known as standard defined exception. In java hierarchy `java.lang.Throwable` is the top most class of all the exception classes.

It has two sub classes

- 1) Errors
- 2) Exception

Errors:-

It is a unreported exception that occurs during the compilation time itself. To handle this kind of exception java provides a keyword called `throws`.

Exception:-

It is of two types



- 1) Checked exception
- 2) Unchecked exception

Checked exception:-

It is a reported exception that occurs during the runtime itself. Application programmer can handle this kind of exception directly. Java provides three keywords to handle this kind of exception.

1. Try
2. Catch
3. Finally

Try:-

Whatever doubtful code we know in the program keep it inside the try block.

Catch:-

Whatever exception is generated inside the try block that exception object is hold by the catch block.

Finally:-

Whether the exception is generated or not whatever we defined inside the finally is compulsory to be executed.

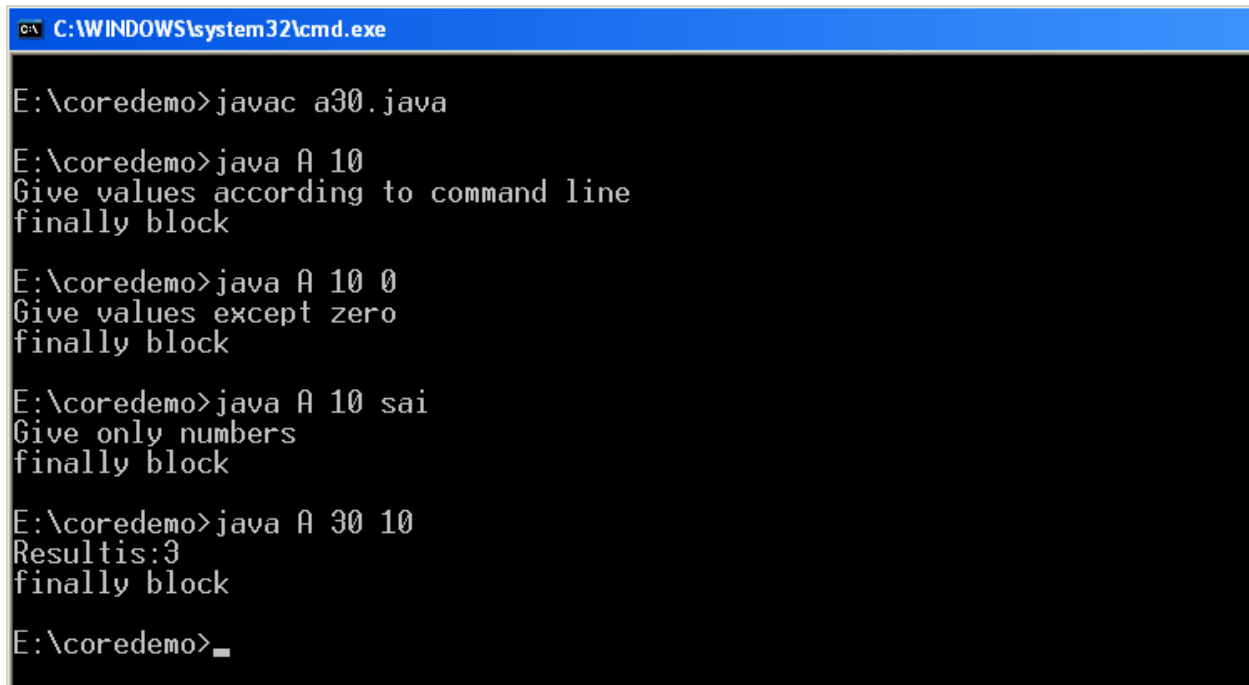
Example program on checked exception:-

```
class A
{
    public static void main(String args[])
    {
        try
        {
            int i=Integer.parseInt(args[0]);
            int j=Integer.parseInt(args[1]);
            int result=i/j;
            System.out.println("Resultis:"+result);
        }
    }
}
```

```
        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println("Give values according
to command line");
        }
        catch(ArithmeticException e)
        {
            System.out.println("Give values except
zero");
        }
        catch(NumberFormatException e)
        {
            System.out.println("Give only numbers");
        }
    finally
    {
        System.out.println("finally block");
    }
}
```

```
    }  
    }  
}
```

Output:-



```
C:\WINDOWS\system32\cmd.exe  
E:\coredemo>javac a30.java  
E:\coredemo>java A 10  
Give values according to command line  
finally block  
E:\coredemo>java A 10 0  
Give values except zero  
finally block  
E:\coredemo>java A 10 sai  
Give only numbers  
finally block  
E:\coredemo>java A 30 10  
Result is:3  
finally block  
E:\coredemo>_
```

user defined exception:-

Creating our own exception classes is known as user defined exception.

Steps to create a user defined exception:-

- 1) In order to create a user defined exception class we have to extend the super class exception class.
- 2) We have to create a user defined exception class object by using a throw keyword.

Program:-

```
class VotingException extends Exception
```

```
{
```

```
VotingException(String s)
```

```
{
```

```
    super(s);
```

```
}
```

```
}
```

```
class A
```

```
{
```

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

```
    {
```

```
        try
```

```
        {
```

```
int age=Integer.parseInt(args[0]);  
if(age<18)  
    throw new VotingException("Invalid age to  
vote");  
else  
    System.out.println("Valid age to vote");  
}  
catch(VotingException e)  
{  
    System.out.println(e.getMessage());  
}  
}  
}
```

Output:-

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

E:\coredemo>javac a31.java

E:\coredemo>java A 17
Invalid age to vote

E:\coredemo>java A 18
Valid age to vote

E:\coredemo>
```

IOProgramming:-

Stream:-

It is a predefined object in java. It takes input stream and converts into output stream and viceversa.

There are two types of streams in java

- 1) Byte oriented stream
- 2) Character oriented stream

There are some of predefined stream classes are available in both byte oriented and character oriented streams. We have to discuss the use of each stream classes in details by showing the related examples.

Java provides all these i/o stream classes in the predefined package called java.io.\*, We have to import this package explicitly.

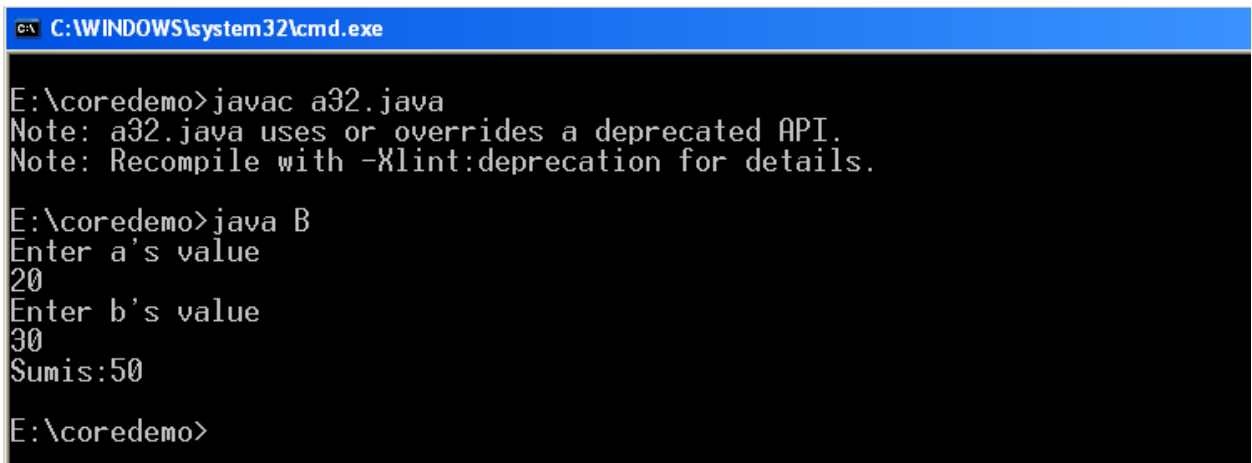
Example program on reading data from keyboard using byte oriented stream:-

```
import java.io.*;
class A
{
int result;
void f1(int a,int b)
    {
        result=a+b;
        System.out.println("Sumis:"+result);
    }
}
class B
{
public static void main(String args[])throws
Exception
    {
        DataInputStream dis=new
        DataInputStream(System.in);
```



```
    /* System.in its captures data from keyboard
    and sent to datainputstream object.*/
    System.out.println("Enter a's value");
    String x=dis.readLine();
    int i=Integer.parseInt(x);
    System.out.println("Enter b's value");
    String y=dis.readLine();
    int j=Integer.parseInt(y);
    A ob=new A();
    ob.f1(i,j);
    dis.close();
}
}
```

Output:-



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

E:\coredemo>javac a32.java
Note: a32.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

E:\coredemo>java B
Enter a's value
20
Enter b's value
30
Sumis:50

E:\coredemo>
```

Example program on reading data from key board using character oriented stream:-

```
import java.io.*;

class A

{

public static void main(String args[])throws Exception

    {

        BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));

        /* InputStreamReader it converts byte oriented
        message to charater oriented message*/

        System.out.println("Enter the name");

        String i=br.readLine();

        System.out.println("Enter the Rollno");

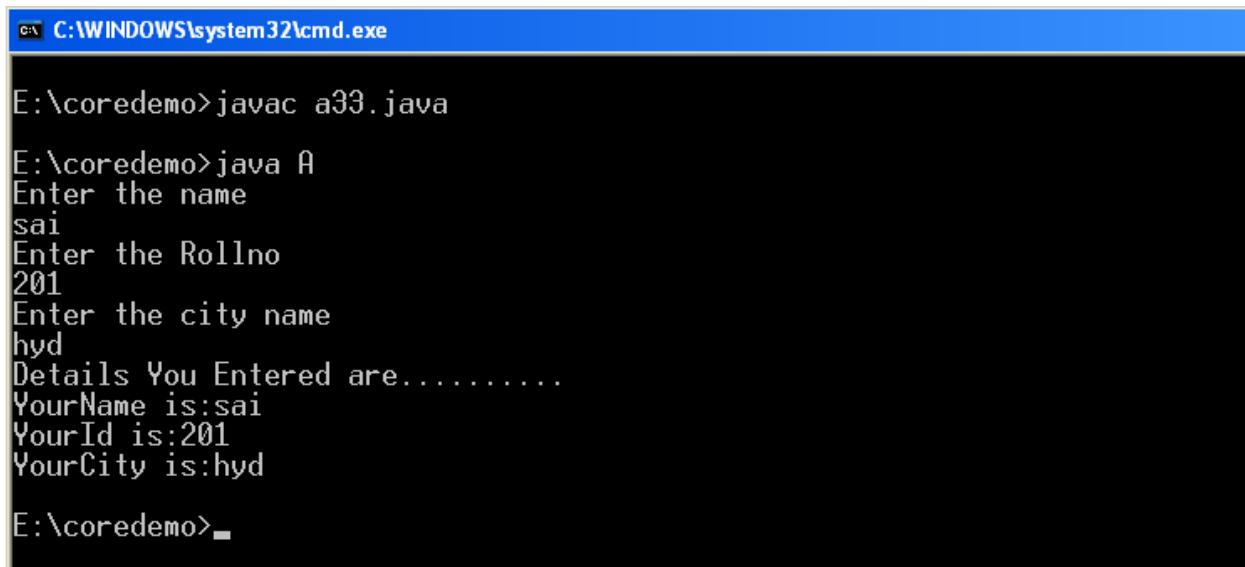
        int j=Integer.parseInt(br.readLine());

        System.out.println("Enter the city name");

        String k=br.readLine();
```

```
        System.out.println("Details You Entered are.....");  
        System.out.println("YourName is:"+i);  
        System.out.println("YourId is:"+j);  
        System.out.println("YourCity is:"+k);  
        br.close();  
    }  
}
```

Output:-



```
C:\WINDOWS\system32\cmd.exe  
E:\coredemo>javac a33.java  
E:\coredemo>java A  
Enter the name  
sai  
Enter the Rollno  
201  
Enter the city name  
hyd  
Details You Entered are.....  
YourName is:sai  
YourId is:201  
YourCity is:hyd  
E:\coredemo>_
```

Example program on reading data from file using byte oriented stream:-

```
import java.io.*;
```

```
class A
```

```
{
```

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

```
{
```

```
    DataInputStream dis=new  
    DataInputStream(System.in);
```

```
    System.out.println("Enter the file name");
```

```
    String filename=dis.readLine();
```

```
    FileInputStream fis=new FileInputStream(filename);
```

```
    int c=fis.read();
```

```
    /* read method throws 1 if char is available
```

```
    in file else it throws -1*/
```

```
    while(c!=-1)
```

```
    {
```

```

        System.out.print((char)c);

        c=fis.read();

    }

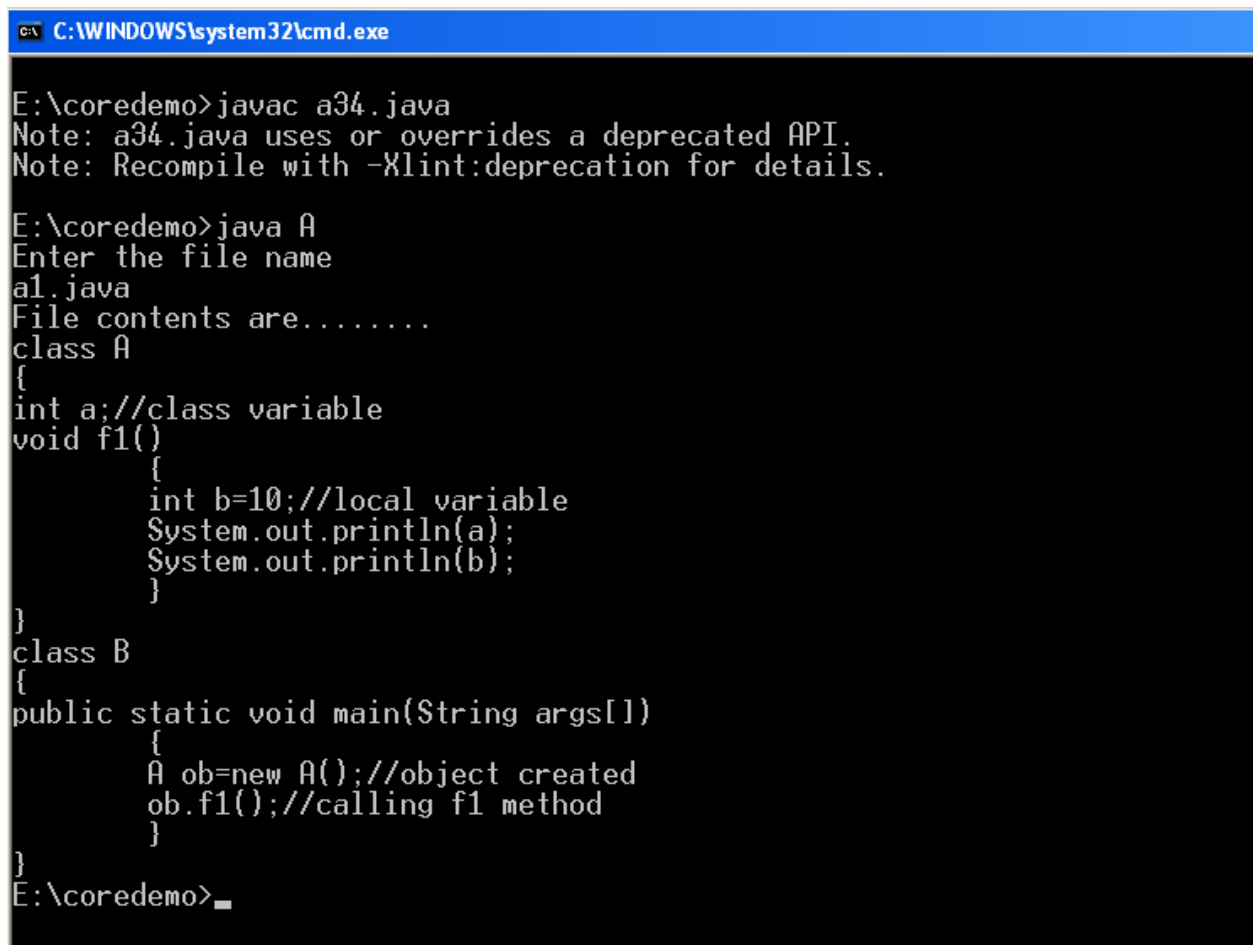
    fis.close();

}

}

```

Output:-



The screenshot shows a Windows command prompt window with the title bar "C:\WINDOWS\system32\cmd.exe". The command prompt is at the directory "E:\coredemo". The user has entered the command "javac a34.java", which compiled the file successfully. A note indicates that "a34.java" uses or overrides a deprecated API and suggests recompiling with "-Xlint:deprecation" for details. The user then entered "java A", which executed the program. The program prompts for a file name, and the user enters "a1.java". The program then displays the contents of "a1.java", which defines two classes: "A" and "B". Class "A" has a class variable "a" and a method "f1()" that prints the value of "a" and a local variable "b" (initialized to 10). Class "B" has a static method "main()" that creates an instance of "A" and calls its "f1()" method. The command prompt ends with a cursor on a new line.

```

C:\WINDOWS\system32\cmd.exe
E:\coredemo>javac a34.java
Note: a34.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

E:\coredemo>java A
Enter the file name
a1.java
File contents are.....
class A
{
int a;//class variable
void f1()
{
    int b=10;//local variable
    System.out.println(a);
    System.out.println(b);
}
}
class B
{
public static void main(String args[])
{
    A ob=new A();//object created
    ob.f1();//calling f1 method
}
}
E:\coredemo>_

```

Example program on reading data from file using character oriented stream:-

```
import java.io.*;

class A
{
    public static void main(String args[])throws Exception
    {
        BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));

        System.out.println("Enter the file name");

        String filename=br.readLine();

        FileReader fr=new FileReader(filename);

        int c=fr.read();

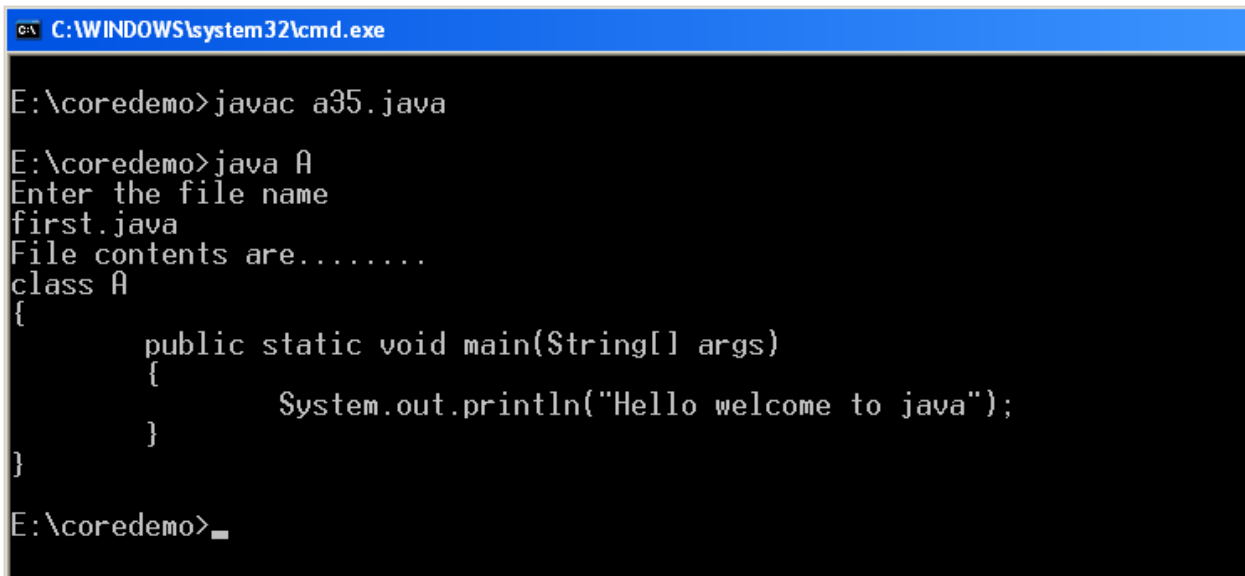
        /* read method throws 1 if char is available
        in file else it throws -1*/

        System.out.println("File contents are.....");
```

```
while(c!=-1)
{
    System.out.print((char)c);
    c=fr.read();
}

fr.close();
br.close();
}
```

Output:-



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

E:\coredemo>javac a35.java

E:\coredemo>java A
Enter the file name
first.java
File contents are.....
class A
{
    public static void main(String[] args)
    {
        System.out.println("Hello welcome to java");
    }
}

E:\coredemo>
```

Example program on writing data into file using byte oriented stream:-

Note:-

FileOutputStream class has two functions.....

- 1) Suppose the file you have entered is not available in the current directory then this stream class is automatically creates a file and write the data into the file.
- 2) Suppose the file you have entered is already available in the current directory then this stream class will overrides the new data into the existing data.

Program:-

```
import java.io.*;

class A

{

public static void main(String args[])throws Exception

{
```



```
DataInputStream dis=new
DataInputStream(System.in);

System.out.println("Enter the fileName");

String filename=dis.readLine();

FileOutputStream fos=new
FileOutputStream(filename);

String s="Hello welcome to io";

/* convert this string format into byte
format by calling the getBytes method
on the string reference*/

byte b[]=s.getBytes();

/* By calling getBytes() on the String reference
it converts the string into byte value and returns
in array form*/

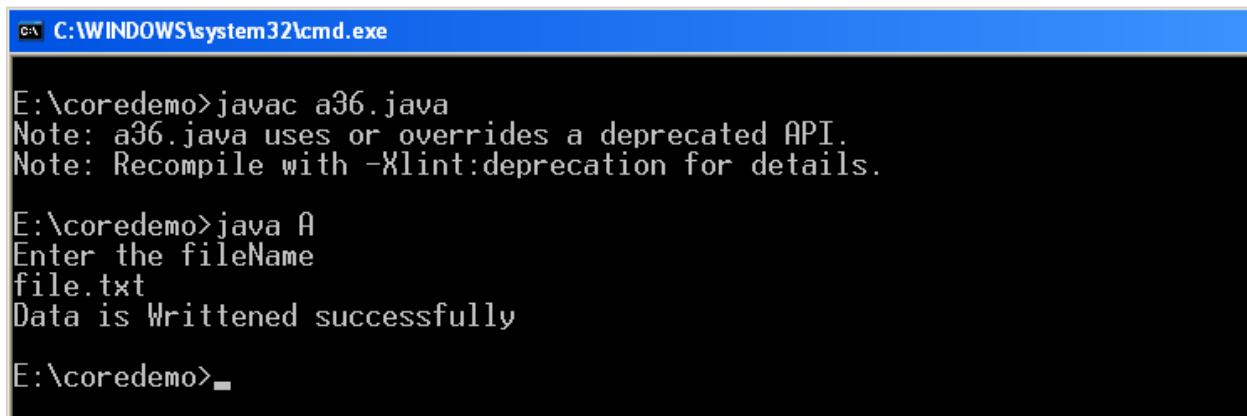
fos.write(b);

System.out.println("Data is Writtended successfully");

fos.close();
```

```
        dis.close();  
    }  
}
```

Output:-



```
C:\WINDOWS\system32\cmd.exe  
E:\coredemo>javac a36.java  
Note: a36.java uses or overrides a deprecated API.  
Note: Recompile with -Xlint:deprecation for details.  
E:\coredemo>java A  
Enter the fileName  
file.txt  
Data is Writtended successfully  
E:\coredemo>_
```

Example program on writing data into file using character oriented stream and the data should not be overridden.

```
import java.io.*;  
  
class A  
{  
  
    public static void main(String args[])throws Exception  
    {
```

```
        BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));

        System.out.println("Enter the file name");

        String filename=br.readLine();

        FileWriter fw=new FileWriter(filename,true);

        String s="hello welcome to io";

        fw.write(s);

        System.out.println("Data is Successfully writtended
with out overriding");

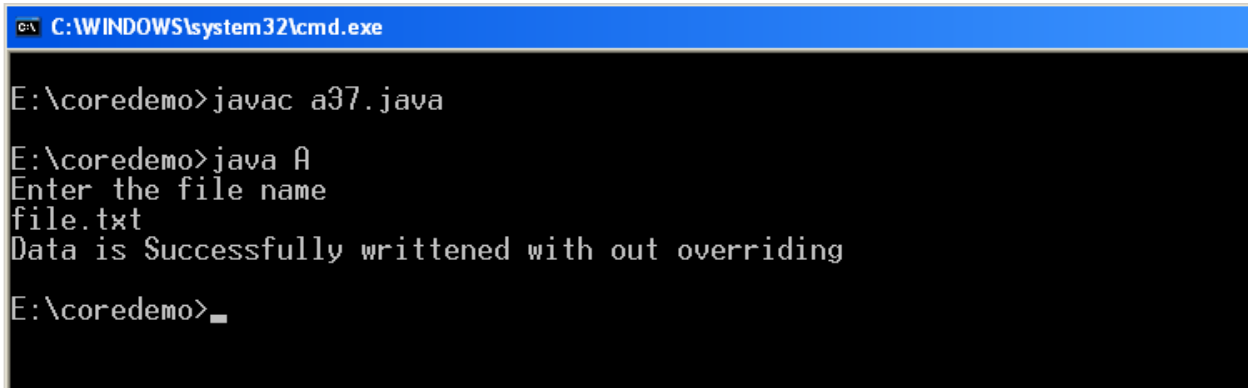
        fw.close();

        br.close();

    }

}
```

## Output:-

A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe". The command prompt shows the following sequence of commands and output:

```
E:\coredemo>javac a37.java
E:\coredemo>java A
Enter the file name
file.txt
Data is Successfully writtended with out overriding
E:\coredemo>_
```

## Object serialization:-

It is the process of converting stream code into object code( byte code)

### Program:-

```
import java.io.*;

class student implements Serializable
{
    int rollno;

    student(int rollno)
    {
        this.rollno=rollno;
    }
}
```

```

    }

void display()

    {

        System.out.println("Rollno:"+rollno);

    }

}

class objser

{

    public static void main(String args[])throws Exception

    {

        FileOutputStream fos=new
        FileOutputStream("file.txt");

        ObjectOutputStream oos=new
        ObjectOutputStream(fos);

        student s=new student(101);

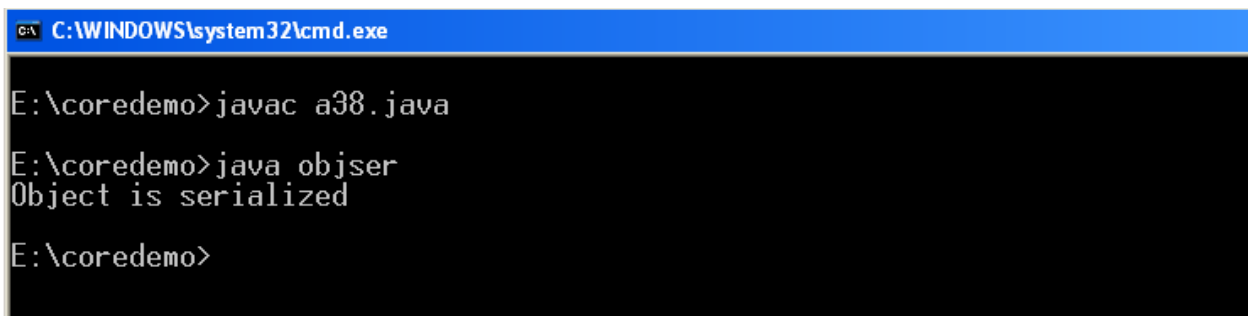
        oos.writeObject(s);

        System.out.println("Object is serialized");
    }
}

```

```
        oos.close();  
        fos.close();  
    }  
}
```

Output:-

A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe". The command prompt shows the following sequence of commands and output:  
E:\coredemo>javac a38.java  
E:\coredemo>java objser  
Object is serialized  
E:\coredemo>

Object deserialization:-

It is the process of converting object code into stream code.

Program:-

```
import java.io.*;  
  
class student implements Serializable  
{  
  
    int rollno;
```

```
student(int rollno)
```

```
{
```

```
    this.rollno=rollno;
```

```
}
```

```
void display()
```

```
{
```

```
    System.out.println("Rollno:"+rollno);
```

```
}
```

```
}
```

```
class objdser
```

```
{
```

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

```
{
```

```
    FileInputStream fis=new FileInputStream("file.txt");
```

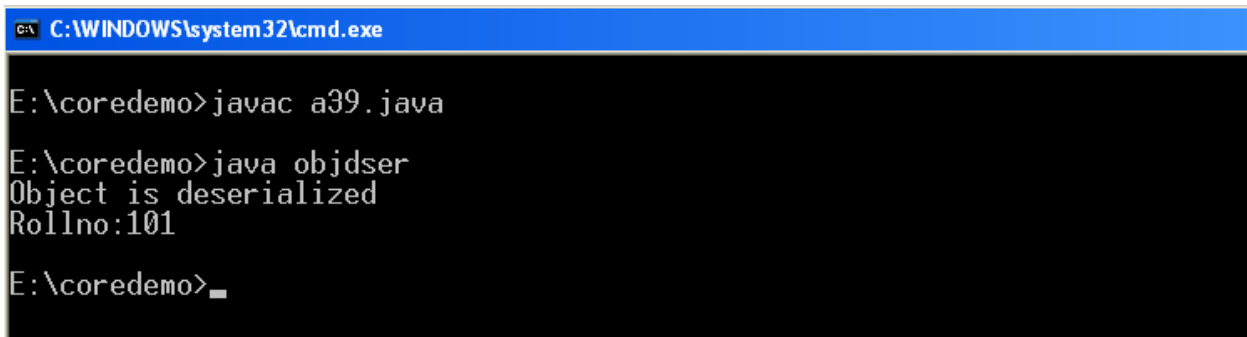
```
    ObjectInputStream ois=new ObjectInputStream(fis);
```

```
    student st=(student)ois.readObject();
```

```
    System.out.println("Object is deserialized");
```

```
        st.display();  
        ois.close();  
        fis.close();  
    }  
}
```

Output:-

A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe". The command prompt shows the following sequence of commands and output:  
E:\coredemo>javac a39.java  
E:\coredemo>java objdser  
Object is deserialized  
Rollno:101  
E:\coredemo>\_

Networking:-

Communication between two machines is called networking.

Socket:-

It is an end point communication between two systems.

In java communication is of two types.....



1) Single way communication

2) Two way communication

Single way communication:-

In this communication client interacts to the server only.

Program:

```
//Client.java
```

```
import java.net.*;
```

```
import java.io.*;
```

```
class Client
```

```
{
```

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

```
{
```

```
    Socket s=new Socket("localhost",9999);
```

```
    System.out.println("connection is established to  
server");
```

```
    OutputStream os=s.getOutputStream();
```

```
        PrintWriter pw=new PrintWriter(os);

        pw.println("Hello Server");

        pw.close();

        s.close();

    }

}

//Server.java

import java.net.*;

import java.io.*;

class Server

{

    public static void main(String args[])throws Exception

    {

        ServerSocket ss=new ServerSocket(9999);

        System.out.println("serversocket object is created");

        System.out.println("server is ready to interact with

client");
```

```
Socket s=ss.accept();

InputStream is=s.getInputStream();

BufferedReader br=new BufferedReader(new
InputStreamReader(is));

String fromclient=br.readLine();

System.out.println("FromClient:"+fromclient);

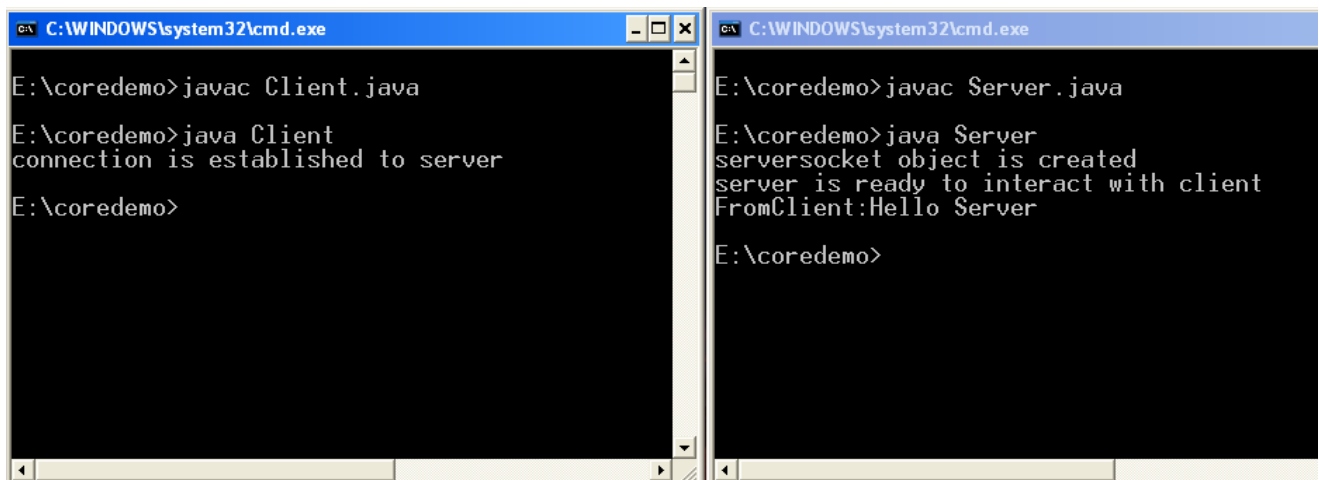
br.close();

ss.close();

}

}
```

Output:-



```
C:\WINDOWS\system32\cmd.exe
E:\coredemo>javac Client.java
E:\coredemo>java Client
connection is established to server
E:\coredemo>

C:\WINDOWS\system32\cmd.exe
E:\coredemo>javac Server.java
E:\coredemo>java Server
serversocket object is created
server is ready to interact with client
FromClient:Hello Server
E:\coredemo>
```

Two way communication:-

In this communication both client and server interacts with each other at the same time.

Program:-

```
//client.java
```

```
import java.net.*;
```

```
import java.io.*;
```

```
class Client
```

```
{
```

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

```
{
```

```
    Socket s=new Socket("localhost",9999);
```

```
    System.out.println("connection is established to  
server");
```

```
    OutputStream os=s.getOutputStream();
```

```
    InputStream is=s.getInputStream();
```

```
        BufferedReader br=new BufferedReader(new
InputStreamReader(is));

        BufferedReader br1=new BufferedReader(new
InputStreamReader(System.in));

        System.out.println("Enter the message to server");

        PrintWriter pw=new PrintWriter(os);

        pw.println(br1.readLine());

        System.out.println("FromServer:"+br.readLine());

        pw.close();

        br1.close();

        br.close();

        s.close();

    }

}
```

//Server.java

```
import java.net.*;
```

```
import java.io.*;
```

```
class Server
{
public static void main(String args[])throws Exception
    {
        ServerSocket ss=new ServerSocket(9999);

        System.out.println("serversocket object is created");

        System.out.println("server is ready to interact with
client");

        Socket s=ss.accept();

        InputStream is=s.getInputStream();

        OutputStream os=s.getOutputStream();

        PrintStream ps=new PrintStream(os);

        BufferedReader br1=new BufferedReader(new
InputStreamReader(System.in));

        System.out.println("Enter the message to client");

        ps.println(br1.readLine());

        System.out.println("Wait a moment client enters the
message");
```

```
        BufferedReader br=new BufferedReader(new
InputStreamReader(is));

        String fromclient=br.readLine();

        System.out.println("FromClient:"+fromclient);

        br1.close();

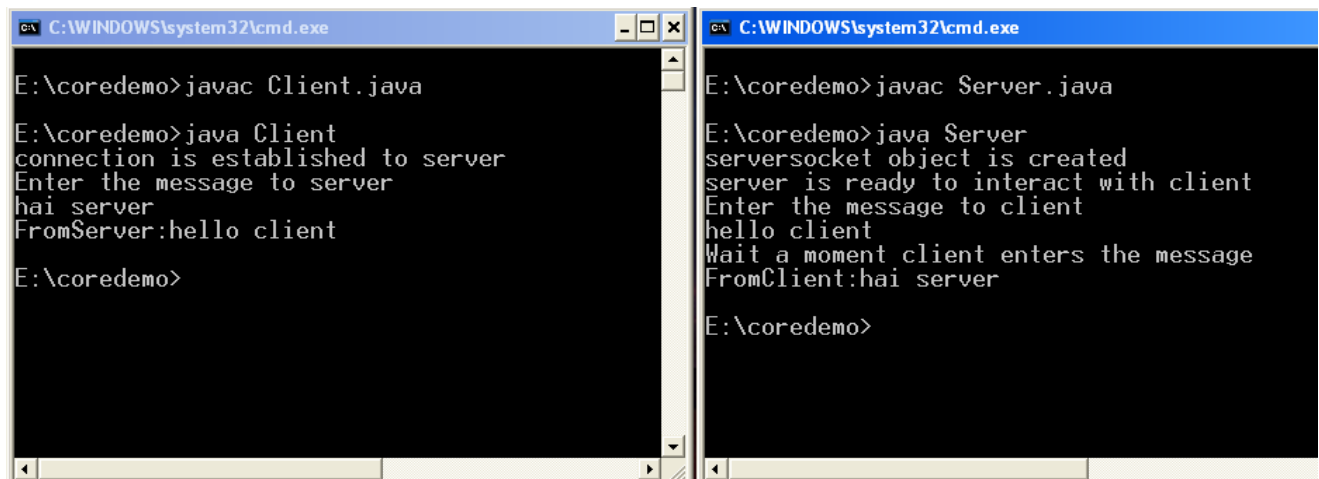
        br.close();

        ss.close();

    }

}
```

Output:-



```
C:\WINDOWS\system32\cmd.exe
E:\coredemo>javac Client.java
E:\coredemo>java Client
connection is established to server
Enter the message to server
hai server
FromServer:hello client
E:\coredemo>

C:\WINDOWS\system32\cmd.exe
E:\coredemo>javac Server.java
E:\coredemo>java Server
serversocket object is created
server is ready to interact with client
Enter the message to client
hello client
Wait a moment client enters the message
FromClient:hai server
E:\coredemo>
```

Multithreading:-

Introduction to multithreading:-

Java provides built in support for multithreaded programming. A multithreading program contains two or more parts that can run concurrently. Each part of a program is known as thread.

What is the multitasking ????

Multitasking:-

It is performing two or more tasks at the same time.

There are two multitasking techniques they are.....

1) Process based multitasking

2) Thread based multitasking

Process based multitasking:-

It is one to run two programs concurrently. Here program means a process. That's why it is also called as process based multitasking



Thread based multitasking:-

It is having a program perform to tasks at the same time. For example word processing program can check the spelling of words of document while you write a document this is called thread based multitasking.

Thread:-

A single sequential flow of process is known as thread.

There are two ways of creating a thread in java.....

- 1) Our class must be extends Thread class.
- 2) Our class must be implements Runnable interface.

Thread life cycle:-

It has four states.....

- 1) New state
- 2) Active state
- 3) Blocked state
- 4) Dead state

New state:-

Whenever we create an object of the subclass of the thread class the thread enters into the new state.

Active state:-

Whenever we call the start method on the subclass object the controller invokes the run method the thread enters into the active state.

Blocked state:-

Whenever we perform the suspending operations inside the run method the thread enters into the blocked state.

Dead state:-

Whenever the code is executed successfully inside the run method automatically jvm calls internally a predefined method called kill and thread becomes dead means the thread object is garbage collected by the garbage collector.

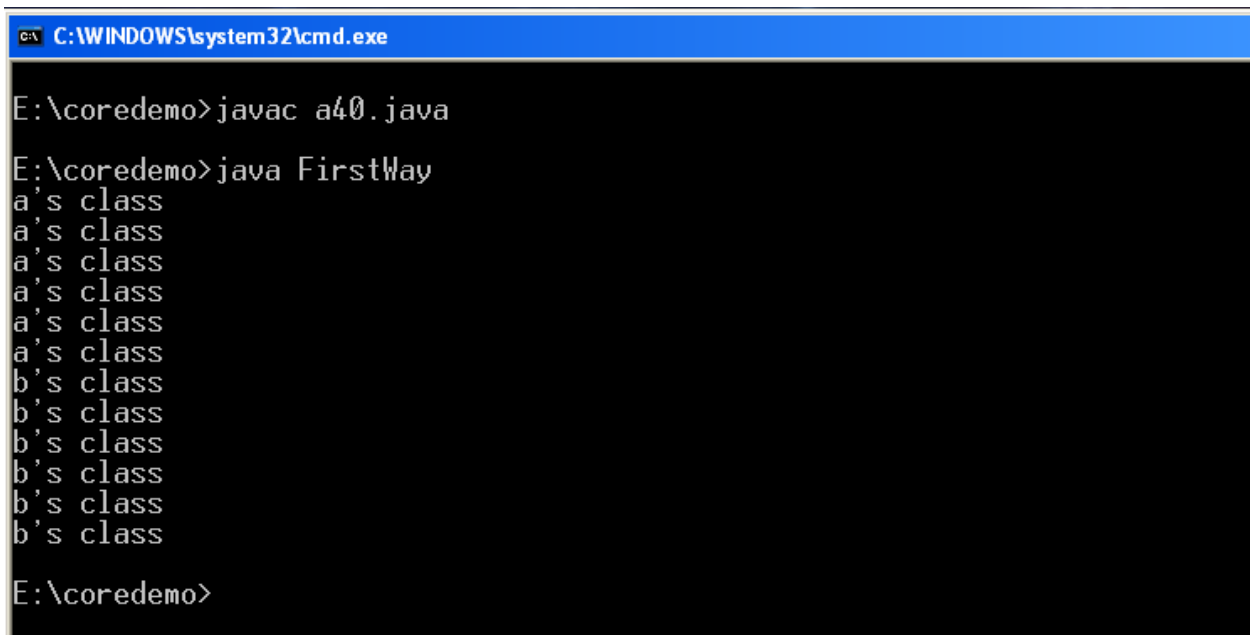
Example program of Implementing our class by first way:-

```
class A extends Thread
{
    public void run()
    {
        for(int i=0;i<=5;i++)
```

```
        {
            System.out.println("a's class");
        }
    }
}
class B extends Thread
{
    public void run()
    {
        for(int i=0;i<=5;i++)
        {
            System.out.println("b's class");
        }
    }
}
class FirstWay
{
    public static void main(String args[])
    {
        A ob=new A();
        B ob1=new B();
        ob.start();
        ob1.start();
    }
}
```

```
}  
}
```

Output:-



```
C:\WINDOWS\system32\cmd.exe  
E:\coredemo>javac a40.java  
E:\coredemo>java FirstWay  
a's class  
a's class  
a's class  
a's class  
a's class  
a's class  
b's class  
b's class  
b's class  
b's class  
b's class  
b's class  
E:\coredemo>
```

Example program of implementing the class by second way:-

```
class A implements Runnable  
{
```

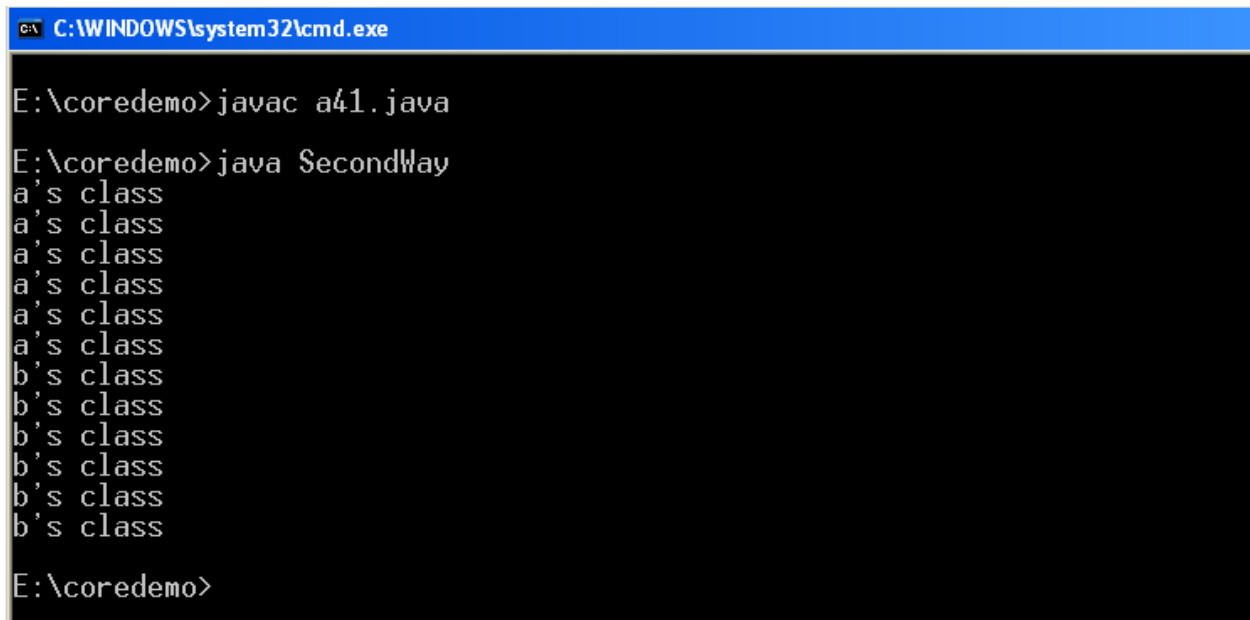
```
public void run()
{
    for(int i=0;i<=5;i++)
    {
        System.out.println("a's class");
    }
}

class B implements Runnable
{
    public void run()
    {
        for(int i=0;i<=5;i++)
        {
            System.out.println("b's class");
        }
    }
}

class SecondWay
{
    public static void main(String args[])
    {
        Thread ob=new Thread(new A());
```

```
        Thread ob1=new Thread(new B());  
        ob.start();  
        ob1.start();  
    }  
}
```

Output:-



```
C:\WINDOWS\system32\cmd.exe  
E:\coredemo>javac a41.java  
E:\coredemo>java SecondWay  
a's class  
a's class  
a's class  
a's class  
a's class  
a's class  
b's class  
b's class  
b's class  
b's class  
b's class  
b's class  
E:\coredemo>
```

Use of sleep method:-

If you want to suspend the thread for few minutes and execute the remaining threads also.

Program:-

```

class A extends Thread
{
    public void run()
    {
        for(int i=0;i<=5;i++)
        {
            if(i==0)
            {
                try
                {
                    System.out.println("Thread enters into
the waiting state of a's class");
                    Thread.sleep(5000);
                }
                catch(Exception e){}
            }
            else
                System.out.println("a's class");
        }
    }
}

class B extends Thread
{

```

```
public void run()
{
    for(int i=0;i<=5;i++)
    {
        System.out.println("b's class");
    }
}

class useofsleep
{
    public static void main(String args[])
    {
        A ob=new A();
        B ob1=new B();
        ob.start();
        ob1.start();
    }
}
```

Output:-



```
C:\WINDOWS\system32\cmd.exe
E:\coredemo>javac a42.java
E:\coredemo>java useofsleep
Thread enters into the waiting state of a's class
b's class
b's class
b's class
b's class
b's class
b's class
a's class
a's class
a's class
a's class
a's class
E:\coredemo>
```

Whenever if you want to share the same object values to different threads at this case we should not get the exact output what we have expected.

Let us discuss the example program of railway reservation system in this we have only one berth but the two passengers were trying to book the berth who is getting the berth see with the below output.....

Program:-

```
class reserve implements Runnable
{
    int available=1;
```

```

int wanted;
reserve(int i)
{
    wanted=i;
}
public void run()
{
    if(available>=wanted)
    {
        System.out.println("Available berths
is:"+available);
        String n=Thread.currentThread().getName();
        System.out.println(available+":berth is reserved
to passenger:"+n);
        try
        {
            System.out.println("Transaction is under
processing");
            Thread.sleep(5000);
            available=available-wanted;
        }
        catch(Exception e){}
    }
}

```

```
        else
            System.out.println("Sorry no berths");
        }
    }
class Unsafe
{
    public static void main(String args[])
    {
        reserve res=new reserve(1);
        Thread t1=new Thread(res);
        Thread t2=new Thread(res);
        t1.setName("sai");
        t2.setName("lakshman");
        t1.start();
        t2.start();
    }
}
```

Output:-

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

E:\coredemo>javac a43.java

E:\coredemo>java UnSafe
Available berths is:1
Available berths is:1
1:berth is reserved to passenger:sai
1:berth is reserved to passenger:laxhman
Transaction is under processing
Transaction is under processing

E:\coredemo>
```

So the two passengers was booked the same berth so the expected output was not presented means thread safe was not provided. To overcome this drawback java provides a mechanism called synchronization to make the thread applications as thread safe.

Synchronization:-

Allowing only one thread at a time and keeping remaining threads under waiting condition. This mechanism is called synchronization.

There are two types.....

- 1)Block level synchronization(object level)
- 2)Method level synchronization

Block level synchronization:-

In this type we can block the single line of code by blocking the current object.

Syntax:-

```
synchronized(objectname)
{
//code
}
```

Modifying the above Railway reservation program applying with block level synchronization.

Program:-

```
class reserve implements Runnable
{
int available=1;
int wanted;
reserve(int i)
{
wanted=i;
}
public void run()
{
synchronized(this)
//this means current object reference
{
```

```

        if(available>=wanted)
        {
            System.out.println("Available berths
is:"+available);
            String n=Thread.currentThread().getName();
            System.out.println(available+":berth is reserved
to passenger:"+n);
            try
            {
                System.out.println("Transaction is under
processing");
                Thread.sleep(5000);
                available=available-wanted;
            }
            catch(Exception e){}
        }
        else
            System.out.println("Sorry no berths");
    }
}
class Safe
{

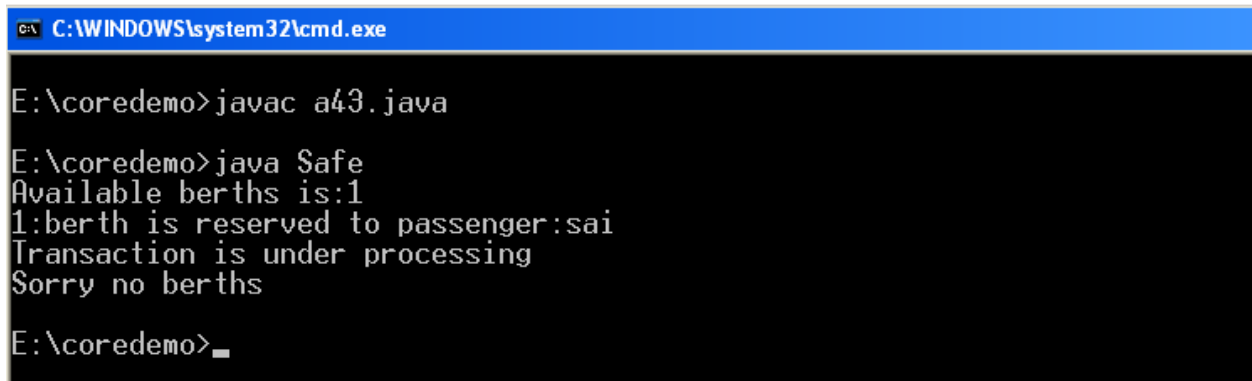
```

```

public static void main(String args[])
{
    reserve res=new reserve(1);
    Thread t1=new Thread(res);
    Thread t2=new Thread(res);
    t1.setName("sai");
    t2.setName("laxhsman");
    t1.start();
    t2.start();
}
}

```

Output:-



```

C:\WINDOWS\system32\cmd.exe
E:\coredemo>javac a43.java
E:\coredemo>java Safe
Available berths is:1
1:berth is reserved to passenger:sai
Transaction is under processing
Sorry no berths
E:\coredemo>_

```

Method level synchronization:-

In this type we can block the particular method by defining the keyword called synchronized beside the method.

Modifying the above railway reservation program by applying method level synchronization.

Program:-

```
class reserve implements Runnable
{
    int available=1;
    int wanted;
    reserve(int i)
    {
        wanted=i;
    }
    synchronized void check()
    {
        if(available>=wanted)
        {
            System.out.println("Available berths
is:"+available);
            String n=Thread.currentThread().getName();
            System.out.println(available+":berth is reserved
to passenger:"+n);
            try
```



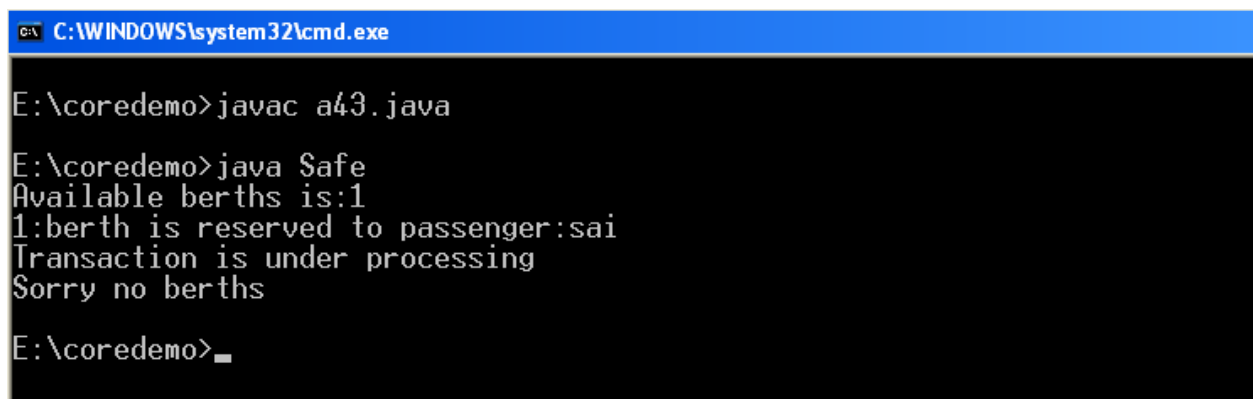
```
        {
            System.out.println("Transaction is under
processing");
            Thread.sleep(5000);
            available=available-wanted;
        }
        catch(Exception e){}
    }
    else
        System.out.println("Sorry no berths");
    }
}
```

```
public void run()
{
    check();
}
```

```
}
class Safe
{
    public static void main(String args[])
    {
        reserve res=new reserve(1);
```

```
        Thread t1=new Thread(res);
        Thread t2=new Thread(res);
        t1.setName("sai");
        t2.setName("laxhsman");
        t1.start();
        t2.start();
    }
}
```

Output:-

A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe". The command prompt shows the following sequence of commands and output:  
E:\coredemo>javac a43.java  
E:\coredemo>java Safe  
Available berths is:1  
1:berth is reserved to passenger:sai  
Transaction is under processing  
Sorry no berths  
E:\coredemo>\_

DeadLock:-

Deadlock describes a situation where two or more threads blocked forever and waiting for each other this situation is called deadlock.

Program:-

```
class BookTicket extends Thread
{
```

```
Object train,compt;
BookTicket(Object train,Object compt)
{
    this.train=train;
    this.compt=compt;
}
public void run()
{
    synchronized(train)
    {
        System.out.println("BookTicket is blocked
on train object");
        try
        {
            System.out.println("BookTicket is
waiting to block on compt object");
            Thread.sleep(100);
        }
        catch(Exception e){}
        synchronized(compt)
        {
            System.out.println("BookTicket is
blocked on compt object");
```

```

        }
    }
}

class CancelTicket extends Thread
{
    Object train,compt;
    CancelTicket(Object train,Object compt)
    {
        this.train=train;
        this.compt=compt;
    }
    public void run()
    {
        synchronized(compt)
        {
            System.out.println("CancelTicket is blocked
on compt object");
            try
            {
                System.out.println("CancelTicket is
waiting to block on train object");
                Thread.sleep(200);
            }
        }
    }
}

```

```

        }
        catch(Exception e){}
        synchronized(train)
        {
            System.out.println("CancelTicket is
blocked on train object");
        }
    }
}
class DeadLock
{
    public static void main(String args[])
    {
        Object train=new Object();
        Object compt=new Object();
        BookTicket bt=new BookTicket(train,compt);
        CancelTicket ct=new CancelTicket(train,compt);
        bt.start();
        ct.start();
    }
}

```

Output:-

```
C:\WINDOWS\system32\cmd.exe - java DeadLock

E:\coredemo>javac a44.java

E:\coredemo>java DeadLock
BookTicket is blocked on train object
CancelTicket is blocked on compt object
BookTicket is waiting to block on compt object
CancelTicket is waiting to block on train object
_
```

Inter thread communication:-

Java provides a very efficient way through which multiple threads communicate with each other. This way reduces the cpus idle time i.e A process where a thread is paused running in its critical region and another thread allowed to enter(or lock) in the same critical section to be executed. This technique is known as inter thread communication which is implemented by some methods as shown below.....

- 1)wait():- It indicates the calling thread to give up the monitor and go to sleep until some other threads enters the same monitor and calls method notify() or notifyAll().
- 2)notify():- It wakes up the first thread that called wait() on the same object.

3) notifyAll():- wakes up all the threads that called wait() on the same object. The important point is highest priority thread will run first.

Note:-

All these methods must be call in a try-catch block.

Example program shows the use of wait and notify methods:-

```
class DemoWait extends Thread
{
    int val=20;
    public static void main(String args[])
    {
        DemoWait d=new DemoWait();
        d.start();
        new Demo1(d);
    }
    public void run()
    {
        try
        {
            synchronized(this)
            {
```

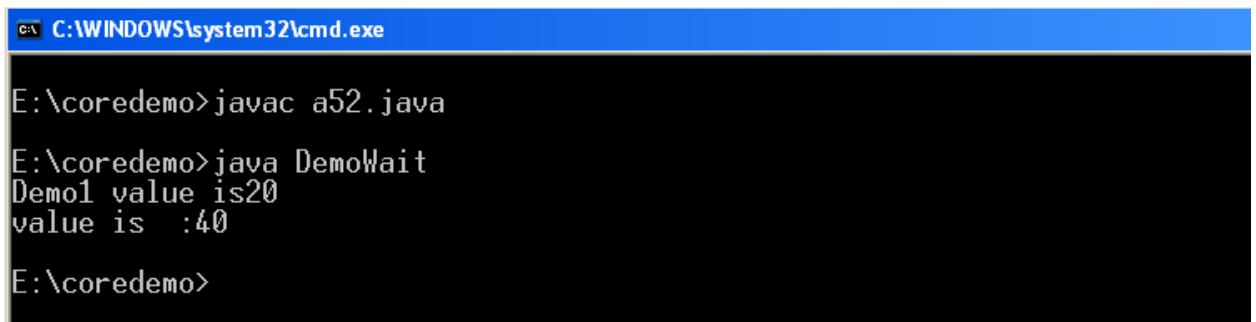
```
wait();  
System.out.println("value is :"+val);  
}  
}  
    catch(Exception e){}  
}
```

```
public void valchange(int val)  
    {  
    this.val=val;  
    try  
        {  
        synchronized(this)  
            {  
            notifyAll();  
            }  
        }  
        catch(Exception e){}  
  
    }  
}  
class Demo1 extends Thread  
    {
```



```
DemoWait d;  
Demo1(DemoWait d)  
    {  
this.d=d;  
start();  
}  
public void run()  
    {  
    try  
    {  
        System.out.println("Demo1 value is"+d.val);  
        d.valchange(40);  
    }  
    catch(Exception e){}  
    }  
}
```

Output:-



```
C:\WINDOWS\system32\cmd.exe  
E:\coredemo>javac a52.java  
E:\coredemo>java DemoWait  
Demo1 value is20  
value is :40  
E:\coredemo>
```

Collection framework:-

It standardizes the way in which groups of objects are handled by your programs. It was designed to meet several goals.

- 1) The framework had to be high performance and implementations of dynamic arrays are highly efficient.
- 2) It had to allow different types of collections to work in a similar manner and with a high degree of interoperability.

Java provides predefined collection classes and interfaces were binded in the package called `java.util` package.

Framework:- It is a set of well defined classes and interfaces.

List of some important predefined classes.....

- 1) Properties
- 2) Arrays
- 3) Vector
- 4) ArrayList
- 5) Hashtable
- 6) Hashmap

List of some important predefined interfaces.....

1) Enumeration

2) Iterator

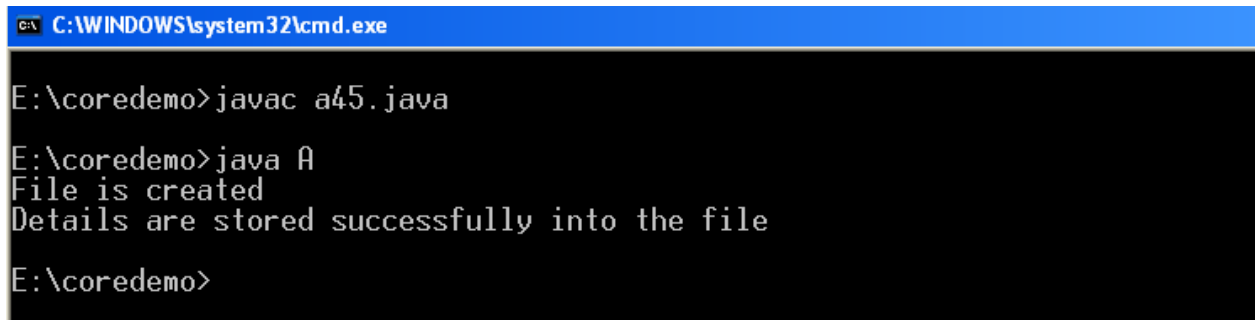
Let us see the each class in detail with example.....

Example program on storing some information into a file using properties.

```
import java.util.*;
import java.io.*;
class A
{
    public static void main(String args[])throws
    Exception
    {
        FileOutputStream fos=new
        FileOutputStream("file.txt");
        System.out.println("File is created");
        Properties p=new Properties();
        p.put("cityname","hyderabad");
        p.put("state","andhrapradesh");
        p.store(fos,"Details of city are.....");
    }
}
```

```
System.out.println("Details are stored successfully  
into the file");  
fos.close();  
}  
}
```

Output:-



```
C:\WINDOWS\system32\cmd.exe  
E:\coredemo>javac a45.java  
E:\coredemo>java A  
File is created  
Details are stored successfully into the file  
E:\coredemo>
```

Example program on getting an information from a file using properties.

```
import java.util.*;  
import java.io.*;  
class A  
{  
    public static void main(String args[])throws  
    Exception  
    {
```

```

        DataInputStream dis=new
DataInputStream(System.in);
        System.out.println("Enter the file name");
        String filename=dis.readLine();
        FileInputStream fis=new
FileInputStream(filename);
        try
        {
            System.out.println("File is reading wait.....");
            Thread.sleep(5000);
        }
        catch(Exception e){}
        Properties p=new Properties();
        //load the fis object into properties object
        p.load(fis);
        System.out.println("DETAILS ARE .....");
        System.out.println("Cityname:"+p.get("cityname
"));
        System.out.println("State :"+p.get("state"));
        fis.close();
    }
}

```

Output:-

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

E:\coredemo>javac a46.java
Note: a46.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

E:\coredemo>java A
Enter the file name
file.txt
File is reading wait.....
DETAILS ARE .....
Cityname:hyderabad
State :andhrapradesh

E:\coredemo>
```

Example program on finding a location of element using arrays:-

```
import java.util.*;
import java.io.*;
class A
{
public static void main(String args[])throws
Exception
{
int arr[]={1,45,67,4,56,78,90};
/*arrange the elements in ascending order
by using the predefined static method sort*/
Arrays.sort(arr);
/*check whether the elements arranged
in ascending order or not*/
```

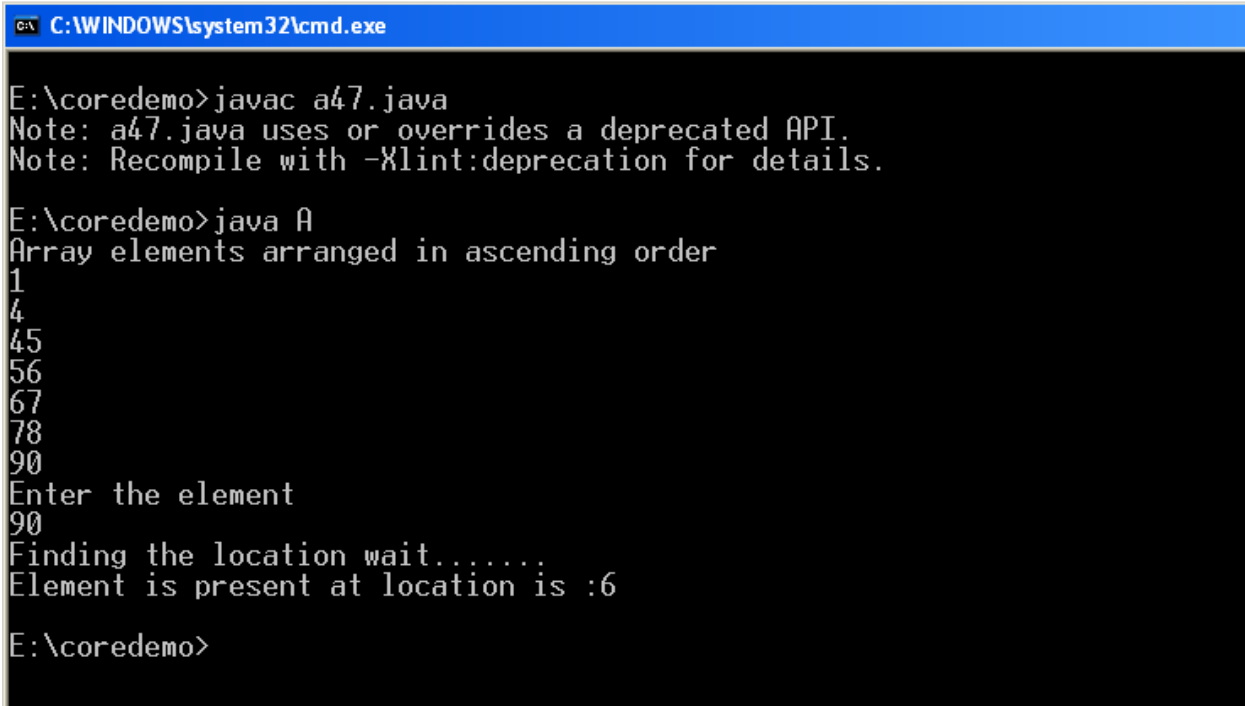
```
        System.out.println("Array elements arranged in
ascending order");
        for(int i=0;i<arr.length;i++)
        {
            System.out.println(arr[i]);
        }
        DataInputStream dis=new
DataInputStream(System.in);
        System.out.println("Enter the element");
        int element=Integer.parseInt(dis.readLine());
        try
        {
            System.out.println("Finding the location
wait.....");
            Thread.sleep(5000);
        }
        catch(Exception e){}
        int location=Arrays.binarySearch(arr,element);
        if(location<0)
            System.out.println("Element is not
present");
        else
```

```

        System.out.println("Element is present at
location is :"+location);
        dis.close();
    }
}

```

Output:-



```

C:\WINDOWS\system32\cmd.exe
E:\coredemo>javac a47.java
Note: a47.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

E:\coredemo>java A
Array elements arranged in ascending order
1
4
45
56
67
78
90
Enter the element
90
Finding the location wait.....
Element is present at location is :6

E:\coredemo>

```

Example program on use of vector class:-

```

import java.util.*;
class student
{
    int rollno;

```



```
student(int rollno)
{
    this.rollno=rollno;
}
void displayStudent()
{
    System.out.println("Rollno:"+rollno);
}
}
class employee
{
    int empno;
    employee(int empno)
    {
        this.empno=empno;
    }
    void displayEmployee()
    {
        System.out.println("Empno:"+empno);
    }
}
class useofvector
{
```

```

public static void main(String args[])
{
    Vector v=new Vector();
    System.out.println("InitialCapacity:"+v.capacity(
));
    System.out.println("InitialSize:"+v.size());
    student st=new student(101);
    employee emp=new employee(1001);
    v.addElement(st);
    v.addElement(emp);
    System.out.println("After adding elements
Size:"+v.size());
    Object o=v.elementAt(0);
    student st1=(student)o;
    st1.displayStudent();
    Enumeration en=v.elements();
    while(en.hasMoreElements()!=false)
    {
        Object o1=en.nextElement();
        if(o1 instanceof student)
        {
            student st2=(student)o1;
            st2.displayStudent();

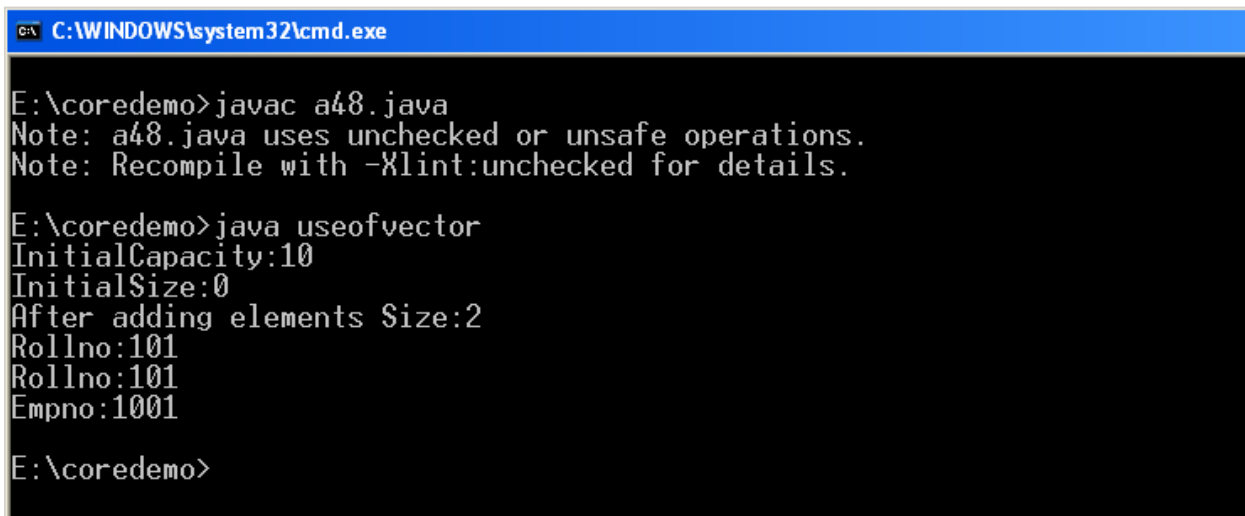
```

```

    }
    else if(o1 instanceof employee)
    {
        employee emp1=(employee)o1;
        emp1.displayEmployee();
    }
}
}
}

```

Output:-



```

C:\WINDOWS\system32\cmd.exe

E:\coredemo>javac a48.java
Note: a48.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

E:\coredemo>java useofvector
InitialCapacity:10
InitialSize:0
After adding elements Size:2
Rollno:101
Rollno:101
Empno:1001

E:\coredemo>

```

Example program on use of ArrayList:-

```

import java.util.*;
import java.io.*;
class A
{

```

```

static int check()throws Exception
{
    ArrayList al=new ArrayList();
    al.add("chennai");
    al.add("hyderabad");
    al.add("mumbai");
    System.out.println("After adding elements size
is:"+al.size());
    System.out.println("Elements in array list
are.....");
    System.out.println(al);
    DataInputStream dis=new
DataInputStream(System.in);
    System.out.println("Select the operation
below");
    System.out.println("1.adding
element,2.removing element,3.viewing elements
4.exit");
    int option=Integer.parseInt(dis.readLine());
    boolean repeat=true;
    while(repeat!=false)
    {
        switch(option)

```

```
{
    case 1:
        System.out.println("Enter the element
u want to add");
        String name=dis.readLine();
        al.add(name);
        System.out.println("Element added
successfully");
        System.out.println("To go back to main
menu type 5");
        option=Integer.parseInt(dis.readLine());
        break;
```

```
    case 2:
        System.out.println("Enter the location
for the element we want to remove");
        int n1=Integer.parseInt(dis.readLine());
        try
        {
            System.out.println("Element is removing
wait.....");
            Thread.sleep(5000);
```

```
        }
        catch(Exception e){}
        al.remove(n1);
        System.out.println("Element is removed
successfully");
        System.out.println("To go back to main menu
type 5");
        option=Integer.parseInt(dis.readLine());
        break;

    case 3:
        try
        {
            System.out.println("wait is under
processing of elements");
            Thread.sleep(5000);
        }
        catch(Exception e){}
        System.out.println("Elements of an
Arraylist are....."+al);
        System.out.println("To go back to main
menu type 5");
        option=Integer.parseInt(dis.readLine());
```

```
        break;

        case 4:
            System.out.println("successfully
exited");
            repeat=false;
            break;
        case 5:
            System.out.println("Select the
operation below");
            System.out.println("1.adding
element,2.removing element,3.viewing elements
4.exit");

            option=Integer.parseInt(dis.readLine());
            break;
        }

    }
    return option;
}
```

```

        public static void main(String args[])throws
        Exception
        {
            check();
        }
    }

```

Output:-

```

C:\WINDOWS\system32\cmd.exe
E:\coredemo>java A
After adding elements size is:3
Elements in array list are.....
[chennai, hyderabad, mumbail]
Select the operation below
1.adding element,2.removing element,3.viewing elements 4.exit
1
Enter the element u want to add
bhimavaram
Element added successfully
To go back to main menu type 5
5
Select the operation below
1.adding element,2.removing element,3.viewing elements 4.exit
2
Enter the location for the element we want to remove
1
Element is removing wait.....
Element is removed successfully

```

What is the difference between vector and arraylist???

- 1)Arraylist is faster and also it occupies less memory space where as vector takes more memory space.
- 2)Vector is a default synchronized object while arraylist is not iterator that are returned by both



classes are fail-fast, but the enumeration returned by vector are not.

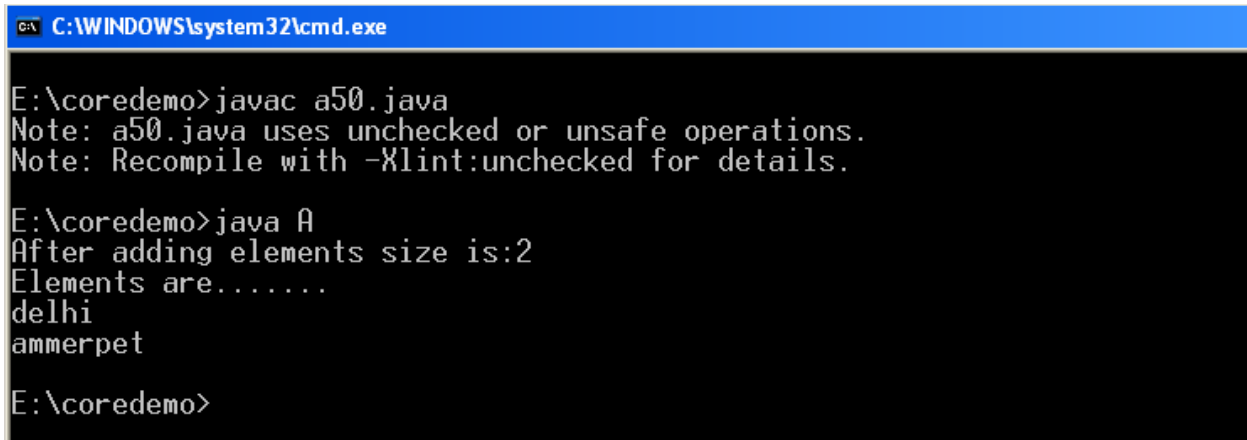
Example program on use of HashTable:-

```
import java.util.*;
class A
{
    public static void main(String args[])
    {
        Hashtable ht=new Hashtable();

        //System.out.println(ht.size());
        ht.put("city","hyd");
        ht.put("city","delhi");
        ht.put("addr","ammerpet");
        System.out.println("After adding elements size
is:"+ht.size());
        Enumeration en=ht.elements();
        System.out.println("Elements are.....");
        while(en.hasMoreElements()!=false)
        {
            Object o=en.nextElement();
            System.out.println(o);
        }
    }
}
```

```
    }  
}
```

Output:-



```
C:\WINDOWS\system32\cmd.exe  
E:\coredemo>javac a50.java  
Note: a50.java uses unchecked or unsafe operations.  
Note: Recompile with -Xlint:unchecked for details.  
E:\coredemo>java A  
After adding elements size is:2  
Elements are.....  
delhi  
ammerpet  
E:\coredemo>
```

Example program on use of HashMap:-

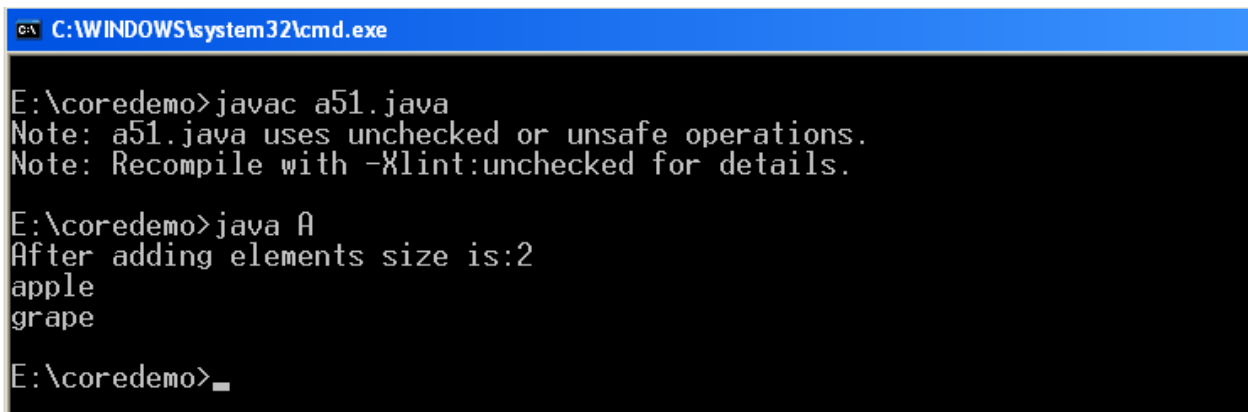
```
import java.util.*;  
class A  
{  
    public static void main(String args[])  
    {  
        HashMap hm=new HashMap();  
        hm.put("fruit","apple");  
        hm.put("fruitttype","grape");  
        System.out.println("After adding elements size  
is:"+hm.size());  
        Collection c=hm.values();
```

```

        Iterator it=c.iterator();
        while(it.hasNext()!=false)
        {
            Object o=it.next();
            System.out.println(o);
        }
    }
}

```

Output:-



```

C:\WINDOWS\system32\cmd.exe

E:\coredemo>javac a51.java
Note: a51.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

E:\coredemo>java A
After adding elements size is:2
apple
grape

E:\coredemo>_

```

What is the difference between hashtable and hashmap???

Sol:-

- 1) hashtable is by default synchronized where as hashmap is not.
- 2) Hashtable can't store null values where as hashmap can store null values.

Java 1.5 and 1.6 features:-

Note:-

Make sure that we have to install the java1.6 or 1.5 software to work out the below features

- 1)Autoboxing
- 2)For-each loop
- 3)Generics
- 4)Enum types
- 5)Varargs
- 6)Reflection api
- 7)Annotations
- 8)Static import

Auto boxing:-

Java added two important features

- 1)Auto boxing
- 2)Auto un boxing

Auto boxing:-

It is the process by which a primitive data type is automatically encapsulated(boxed) into its equivalent type wrapper whenever an object of that type is needed. There is no need to explicitly construct an object.

Auto un boxing:-

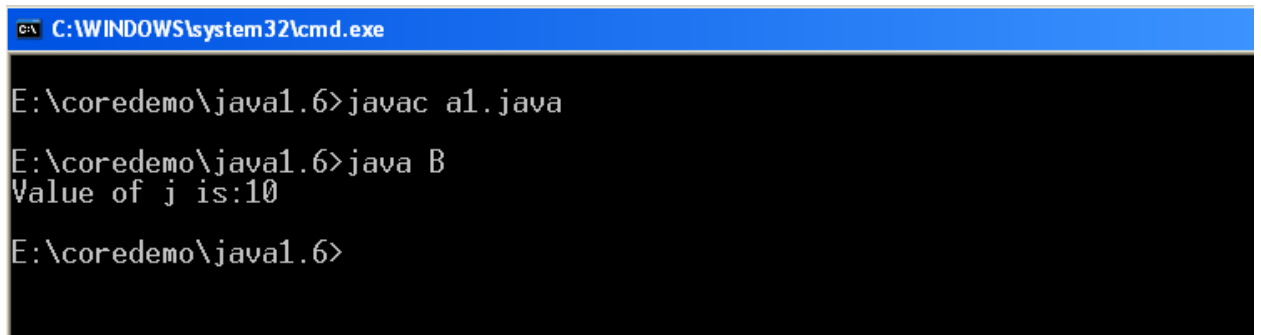
It is the process by which the value of a boxed object is automatically extracted(unboxed) from a type wrapper when its value is needed. So there is no need to call explicitly such as intValue() or doubleValue().

Program:-

```
class A
{
void f1()
{
    Integer i=10;//auto in boxing
    int j=i;//auto out boxing
    System.out.println("Value of j is:"+j);
}
}
```

```
class B
{
    public static void main(String args[])
    {
        A ob=new A();
        ob.f1();
    }
}
```

Output:-

A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\WINDOWS\system32\cmd.exe". The command prompt has a black background with white text. The text shows the following sequence of commands and output:  
E:\coredemo\java1.6>javac a1.java  
E:\coredemo\java1.6>java B  
Value of j is:10  
E:\coredemo\java1.6>

For-each loop:-

It is designed to cycle through a collection of objects such as an array, in strictly sequential order from start to finish. In c# that implements a for-each loop by using the keyword foreach. Java adds

the for-each capability by enhancing the for statement

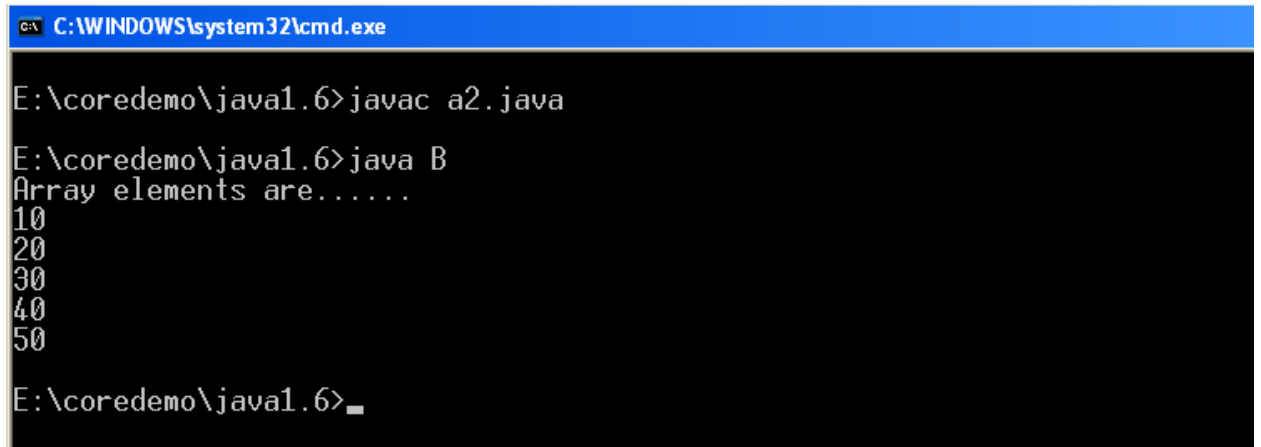
Program:-

```
class A
{
int arr[]={10,20,30,40,50};
void f1()
    {
        System.out.println("Array elements are.....");
        for(int a:arr)
            {
                System.out.println(a);
            }
    }
}

class B
{
public static void main(String args[])
    {
        A ob=new A();
        ob.f1();
    }
}
```

```
}
```

Output:-

A screenshot of a Windows command prompt window. The title bar is blue and reads "C:\WINDOWS\system32\cmd.exe". The command prompt shows the following sequence of commands and output:

```
E:\coredemo\java1.6>javac a2.java
E:\coredemo\java1.6>java B
Array elements are.....
10
20
30
40
50
E:\coredemo\java1.6>_
```

Generics:-

The term generics means parameterized types.

Parameterized types are important because they allow you to create classes , interfaces , and methods in which the type of data upon which they operate as a specified parameter. By using generics it is possible to create a single class that automatically works for different data types of data.

Program:-

```
class A<t>
{
    t a;
    A(t a)
```



```

        {
            this.a=a;
        }
    t f1()
    {
        return a;
    }
}
class UseOfGenerics
{
    public static void main(String args[])
    {
        Integer i=10;//auto boxing
        A<Integer> ob=new A<Integer>(i);
        System.out.println("Hi ur using generic");
        System.out.println("Value of a is:"+ob.f1());
    }
}

```

Output:-

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

E:\coredemo\java1.6>javac a3.java

E:\coredemo\java1.6>java UseOfGenerics
Hi ur using generic
Value of a is:10

E:\coredemo\java1.6>_
```

Note:-

1. In the above program where t is the name of type parameter.
2. Generics work only with the objects for example you cannot pass like this if do so it gives a compilation error.  
Ex:- `A<int> ob=new A<int>(53);`//compilation error.

Handling generic class with two type parameters:-

In generic class you can declare more than two type parameter in a generic type by simply using a comma-separated list.

Program:-

```
class A<t,v>
{
    t a1;
    int result;
    v a2;
```

```

A(t a1,v a2)
{
    this.a1=a1;
    this.a2=a2;
}
void display()
{
    String a=a1.toString();
    /*toString() is used to convert the
    object type into string type*/
    String b=a2.toString();
    Integer i=new Integer(a);
    /*Integer wrapper class is used to convert
    string value into wrapped object value*/
    Integer j=new Integer(b);
    int k=i;//auto unboxing
    int l=j;//auto unboxing
    result=k+l;
    System.out.println("Sumis:"+result);
}
}
class GenericDemo
{

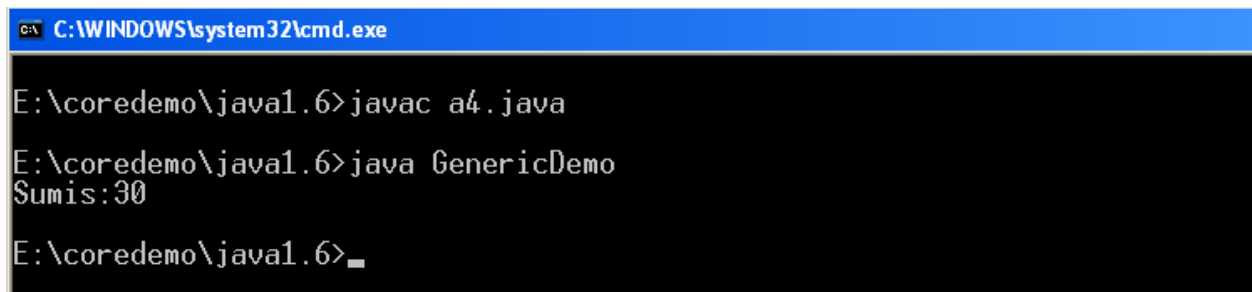
```

```

    public static void main(String args[])
    {
        Integer i=10;
        Integer j=20;
        A<Integer,Integer> ob=new
        A<Integer,Integer>(i,j);
        ob.display();
    }
}

```

Output:-



```

C:\WINDOWS\system32\cmd.exe

E:\coredemo\java1.6>javac a4.java

E:\coredemo\java1.6>java GenericDemo
Sumis:30

E:\coredemo\java1.6>_

```

Enumeration:-

A enumeration is a list of named constants. In java to declare a variable as constant it provides keyword called final. Java enumerations are appear similar to enumerations in other languages. In languages such as c++, enumerations are simply lists of named integer constants. In java an enumerations defines a class type.

In java an enumeration can have constructors, methods and instance variables.

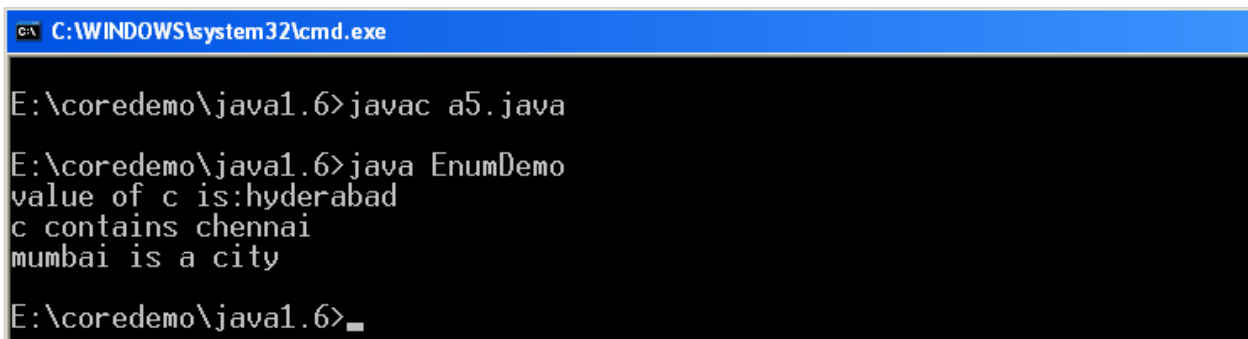
An enumeration is created in java using the keyword enum.

Program:-

```
enum city
{
    hyderabad,chennai,mumbai;
}
class EnumDemo
{
    public static void main(String args[])
    {
        city c;
        c=city.hyderabad;
        System.out.println("value of c is:"+c);
        //enum provides to compare two values also
        c=city.chennai;
        if(c==city.chennai)
            System.out.println("c contains chennai");
        //enum supports to control a switch statement
        c=city.mumbai;
```

```
switch(c)
{
case hyderabad:
    System.out.println("hyderabad is a
city");
    break;
case chennai:
    System.out.println("chennai is a city");
    break;
case mumbai:
    System.out.println("mumbai is a city");
    break;
}
}
```

Output:-



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

E:\coredemo\java1.6>javac a5.java

E:\coredemo\java1.6>java EnumDemo
value of c is:hyderabad
c contains chennai
mumbai is a city

E:\coredemo\java1.6>_
```

Note:-

In the above program Hyderabad, Mumbai and so on are called enumeration constants. Each one is implicitly declared as public , static , final member of city.

Use of values() and valueOf() methods in enumeration:-

All enumerations automatically contains two predefined methods values() and valueOf().

values():-

This method returns an enum array type that contains a list of the enumeration constants.

valueOf():-

This method returns the enumeration constant whose value corresponds to the string passed in argument.

Example program shows how to use the values and valueOf methods:-

```
enum city
{
    hyderabad,mumbai,chennai;
}
```

```

class EnumDemo1
{
    public static void main(String args[])
    {
        city c;
        //use values()
        city c1[]=city.values();
        System.out.println("city constants are.....");
        for(city c2:c1)
        {
            System.out.println(c2);
        }
        System.out.println("-----");
        //use valueOf()
        c=city.valueOf("mumbai");
        System.out.println("C contains:"+c);
    }
}

```

Output:-



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

E:\coredemo\java1.6>javac a6.java

E:\coredemo\java1.6>java EnumDemo1
city constants are.....
hyderabad
mumbai
chennai
-----
C contains:mumbai

E:\coredemo\java1.6>_
```

Can we define java enumerations as a class types???

Sol:- yes we can define java enumerations as a class type.

Example program shows how to define java enumerations as a class type and enum constructor:-

```
enum fruits
{
    Apple(50),orange(10),Grape(30);
    private int price;
    fruits(int p)
    {
        price=p;
    }
    int getPrice()
    {
        return price;
    }
}
```

```

        }
    }
    class EnumDemo2
    {
    public static void main(String args[])
    {

        //display price of apple
        System.out.println("Apple cost
is:"+fruits.Apple.getPrice());
        //display all prices of fruits
        System.out.println("All fruits prices
are.....");
        for(fruits fr:fruits.values())
        {
            System.out.println(fr + "cost is:" +
fr.getPrice());
        }
    }
}

```

Output:-

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

E:\coredemo\java1.6>javac a7.java

E:\coredemo\java1.6>java EnumDemo2
Apple cost is:50
All fruits prices are.....
Applecost is:50
orangecost is:10
Grapecost is:30

E:\coredemo\java1.6>
```

Note:-

- 1) In the above program each enumeration constant is an object of its enumeration type.
- 2) When you define a constructor for an enum, the constructor is called when each enumeration constant is created.
- 3) Each enumeration constant has its own copy of any instance variables defined by enumeration.

Varargs:-

It simplifies the creation of methods that need to take a variable number of arguments. This feature is called varargs and its short for variable number of arguments.

What is the use of varargs???

Sol:- for example a method that opens a internet connection might take a username, password and so on, but supply defaults if some of this information is not provided. At this situation this concept helps to pass only the arguments to which the defaults did not apply.

Note :-

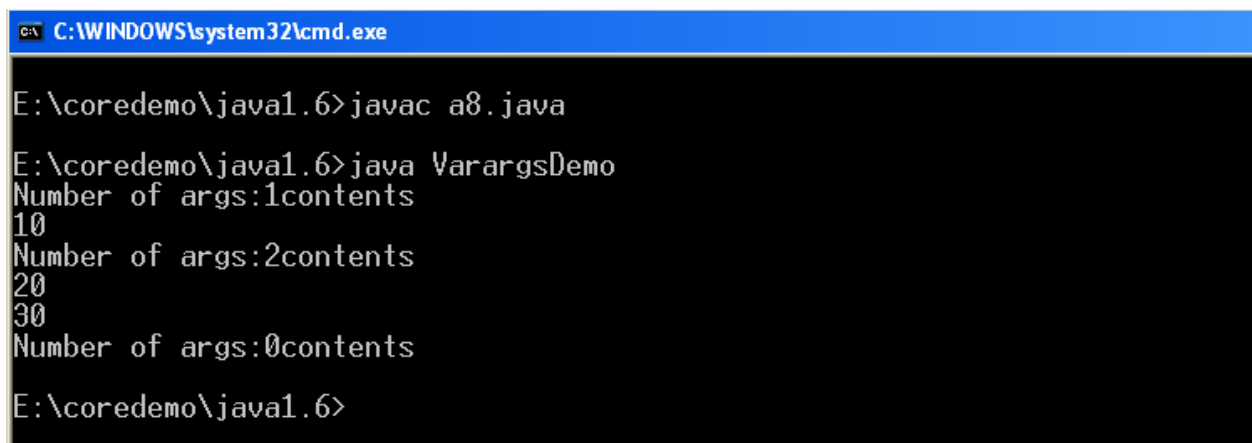
A variable length argument is specified by three periods(...).

Program:-

```
class A
{
void f1(int...v)
{
    System.out.println("Number of
args:"+v.length+"contents");
    for(int x:v)
        System.out.println(x+"");
}
}
class VarargsDemo
{
```

```
public static void main(String args[])
{
    A ob=new A();
    ob.f1(10);
    ob.f1(20,30);
    ob.f1();
}
}
```

Output:-



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

E:\coredemo\java1.6>javac a8.java

E:\coredemo\java1.6>java VarargsDemo
Number of args:1contents
10
Number of args:2contents
20
30
Number of args:0contents

E:\coredemo\java1.6>
```

Note:-

- 1) In the above program v is operated as an array
- 2) The syntax ... simply tells to the compiler that a variable number of arguments will be used and these arguments will be stored in the array referred to by v.

3) In the case of no arguments the length of an array is zero.

Q) can we overload a vararg methods or not???

a) yes we can overload a vararg methods

Example program shows on overloading varargs methods:-

```
class A
{
void f1(int...i)
    {
        System.out.println("Number of
args:"+i.length+"contents");
        for(int x:i)
            System.out.println(x+"");
    }
void f1(boolean...i)
    {
        System.out.println("Number of
args:"+i.length+"contents");
        for(boolean x:i)
            System.out.println(x+"");
    }
}
```

```

    }
void f1(String msg,float...i)
{
    System.out.println("Number of
args:"+i.length+"contents");
    for(float x:i)
        System.out.println(x+"");
}
}
class VarargsDemo1
{
public static void main(String args[])
{
    A ob=new A();
    ob.f1(1,2,3);
    ob.f1(true,false,true);
    ob.f1("saikrishna",10.89f,70.89f);
}
}

```

Output:-

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

E:\coredemo\java1.6>javac a9.java

E:\coredemo\java1.6>java VarargsDemo1
Number of args:3contents
1
2
3
Number of args:3contents
true
false
true
Number of args:2contents
10.89
70.89

E:\coredemo\java1.6>_
```

Reflection api:-

Reflection is commonly used by the programs which require the ability to examine or modify the runtime behavior of applications running in the java virtual machine.

Example program on getting the details of methods in other class

```
import java.lang.reflect.*;
class A
{
void f1(int a,int b)
{
    int result=a+b;
```



```

        System.out.println("Sumis:"+result);
    }
}
class B
{
    public static void main(String args[])
    {
        A ob=new A();
        Class c=ob.getClass();
        Method m[]=c.getDeclaredMethods();
        for(Method m1:m)
        {
            System.out.println("MethodName
is:"+m1.getName());
            Class c1[]=m1.getParameterTypes();
            Class c2=m1.getReturnType();
            System.out.println("Method Parameter
name is:"+c1[0].getName());
            System.out.println("Method Return
Type:"+c2.getName());
        }
    }
}

```

Output:-

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

E:\coredemo\java1.6>javac a10.java

E:\coredemo\java1.6>java B
MethodName is:f1
Method Parameter name is:int
Method Return Type:void

E:\coredemo\java1.6>
```

Example program on getting the details of constructors in other class.

```
import java.lang.reflect.*;

class A
{
    A(int a)
    {
        System.out.println("value of a is:"+a);
    }
}

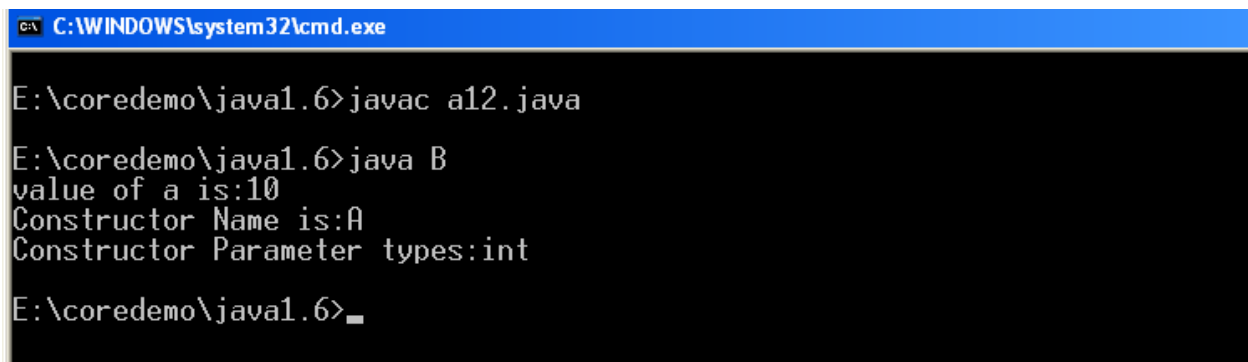
class B
{
    public static void main(String args[])
    {
        A ob=new A(10);
```

```

        Class c=ob.getClass();
        Constructor cn[]=c.getDeclaredConstructors();
        for(int i=0;i<cn.length;i++)
        {
            Class c1[]=cn[i].getParameterTypes();
            System.out.println("Constructor Name
is:"+cn[i].getName());
            System.out.println("Constructor Parameter
types:"+c1[i].getName());
        }
    }
}

```

Output:-



```

C:\WINDOWS\system32\cmd.exe
E:\coredemo\java1.6>javac a12.java
E:\coredemo\java1.6>java B
value of a is:10
Constructor Name is:A
Constructor Parameter types:int
E:\coredemo\java1.6>_

```

Annotations:-

It is a new facility added to a java that enables you to embed supplemental information into a source file. This information is called an annotation. It does not change

the actions of a program. This information is used by various tools during both development and deployment.

An annotation is created through a mechanism based on the interface

Example program on annotation:-

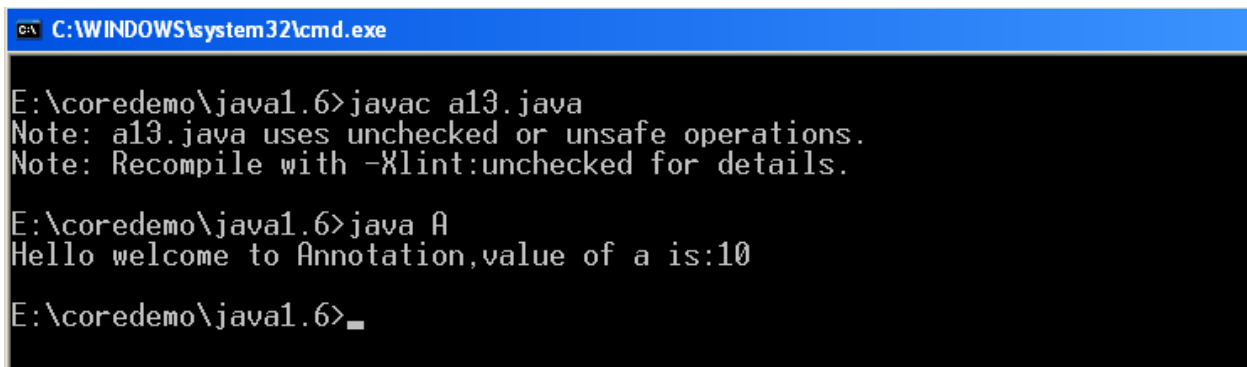
```
import java.lang.reflect.*;
import java.lang.annotation.*;
@Retention(RetentionPolicy.RUNTIME)
@interface i1
{
    String f1();
    int f2();
}
class A
{
    @i1(f1="Hello welcome to Annotation",f2=10)
    public static void f3()throws Exception
    {
        A ob=new A();
        Class c=obj.getClass();
        Method m1=c.getMethod("f3");
```

```

        i1 i=m1.getAnnotation(i1.class);
        System.out.println(i.f1()+" ,value of a is:"+i.f2());
    }
    public static void main(String args[])throws
    Exception
    {
        f3();
    }
}

```

Output:-



```

C:\WINDOWS\system32\cmd.exe

E:\coredemo\java1.6>javac a13.java
Note: a13.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

E:\coredemo\java1.6>java A
Hello welcome to Annotation,value of a is:10

E:\coredemo\java1.6>_

```

RetentionPolicy:-

A retention policy determines at what point an annotation is not needed. Java defined three such policies which are encapsulated with in the java.lang.annotation.RetentionPolicy enumeration. They are SOURCE, CLASS and RUNTIME.

- 1) An annotation with a retention policy of SOURCE is retained only in the source file and is not needed during compilation.
- 2) An annotation with a retention policy of CLASS is stored in .class file during compilation. It is not available to jvm during run time.
- 3) An annotation with a retention policy of RUNTIME is stored in the .class file during compilation and is available through the jvm during run time. This RUNTIME retention offers the greatest annotation persistence.

A retention policy is specified for an annotation by using one of java's built in annotations @Retention.

Note:-

If no retention policy is specified in the annotation, then the default policy of CLASS is used.

Marker annotations:-

A marker annotation is a special kind of annotation which has no members.

Ex:-

```
@interface marker{}
```