**Introduction to Object Oriented Programming using JAVA**

Java is a high-level programming language originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. James Gosling initiated the Java language project in June 1991 for use in one of his many set-top box projects. The language, initially called Oak after an oak tree that stood outside Gosling's office, also went by the name Green and ended up later being renamed as Java, from a list of random words.

**Overview of Java**

Java Is Important to the Internet, The Internet helped catapult Java to the forefront of programming, and Java, in turn, has had a profound effect on the Internet. The reason for this is quite simple: Java expands the universe of objects that can move about freely in cyberspace. In a network, two very broad categories of objects are transmitted between the server and our personal computer: passive information and dynamic, active programs.

Java can be used to create two types of programs: applications and applets. An application is a program that runs on your computer, under the operating system of that computer. An applet is an application designed to be transmitted over the Internet and executed by a Java compatible Web browser.

**Features of JAVA**

* Simple
* Secure
* Portable
* Object-oriented
* Robust
* Multithreaded
* Architecture-neutral
* Interpreted
* High performance
* Distributed
* Dynamic

**JDK**

The Java Development Kit (JDK) is an implementation of either one of the Java SE, Java EE or Java ME platforms. The JDK includes a private JVM and a few other resources to finish the development of a Java Application.

**Data Types used in JDK**

Each row contains the data type and size and range of the data type. The list of available data types in Java is shown in table below

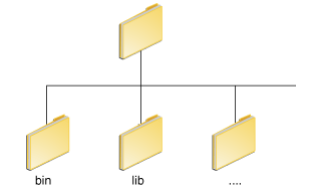
|  |  |  |
| --- | --- | --- |
| **Name** | **Width** | **Range** |
| byte | 8 | –128 to 127 |
| short | 16 | –32,768 to 32,767 |
| int | 32 | –2,147,483,648 to 2,147,483,647 |
| long | 64 | –2,147,483,648 to 2,147,483,647 |
| float | 32 | –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| double | 64 | 32 1.4e−045 to 3.4e+038 |
| Char | 2 | 4.9e–324 to 1.8e+308 |
| Boolean | 1 | True or false |

**PATH and CLASSPATH**

Setting PATH and CLASSPATH environment variables on Microsoft Windows, Solaris, and Linux are as follows.

Install the Java Development Kit (JDK) software.

After installing the software, the JDK directory will have the structure shown below

****

The bin directory contains both the compiler and the launcher.

**Update the PATH Environment Variable (Microsoft Windows)**

We can run Java applications just fine without setting the PATH environment variable. Or, we can optionally set it as a convenience.

Set the PATH environment variable if we want to be able to conveniently run the executables (javac.exe, java.exe, javadoc.exe, and so on) from any directory without having to type the full path of the command. If we do not set the PATH variable, we need to specify the full path to the executable every time we run it, such as:

C:\Java\jdk1.7.0\bin\javac MyClass.java

The PATH environment variable is a series of directories separated by semicolons (;). Microsoft Windows looks for programs in the PATH directories in order, from left to right. We should have only one bin directory for the JDK in the path at a time (those following the first are ignored), so if one is already present, we can update that particular entry.

**The following is an example of a PATH environment variable:**

C:\Java\jdk1.7.0\bin;C:\Windows\System32\;C:\Windows\;C:\Windows\System32\Wbem It is useful to set the PATH environment variable permanently so it will persist after rebooting. To make a permanent change to the PATH variable, use the System icon in the Control Panel. The precise procedure varies depending on the version of Windows:

**Windows XP**

1. Select Start, select Control Panel. double click System, and select the Advanced tab.
2. Click Environment Variables. In the section System Variables, find the PATH environment variable and select it. Click Edit. If the PATH environment variable does not exist, click New.
3. In the Edit System Variable (or New System Variable) window, specify the value of the PATH environment variable. Click OK. Close all remaining windows by clicking OK.

**NetBeans**

**NetBeans** is an [integrated development environment](https://en.wikipedia.org/wiki/Integrated_development_environment) (IDE) for [Java](https://en.wikipedia.org/wiki/Java_(programming_language)). NetBeans allows applications to be developed from a set of modular [software components](https://en.wikipedia.org/wiki/Software_component) called *modules*. NetBeans runs on [Windows](https://en.wikipedia.org/wiki/Microsoft_Windows), [macOS](https://en.wikipedia.org/wiki/MacOS), [Linux](https://en.wikipedia.org/wiki/Linux) and [Solaris](https://en.wikipedia.org/wiki/Solaris_(operating_system)). In addition to Java development, it has extensions for other languages like [PHP](https://en.wikipedia.org/wiki/PHP), [C](https://en.wikipedia.org/wiki/C_(programming_language)), [C++](https://en.wikipedia.org/wiki/C%2B%2B), [HTML5](https://en.wikipedia.org/wiki/HTML5) and [JavaScript](https://en.wikipedia.org/wiki/JavaScript). Applications based on NetBeans, including the NetBeans IDE, can be extended by [third party developers](https://en.wikipedia.org/wiki/Third_party_developer).

**NetBeans IDE** is an [open-source](https://en.wikipedia.org/wiki/Open-source_software) integrated development environment. NetBeans IDE supports development of all Java application types ([Java SE](https://en.wikipedia.org/wiki/Java_Platform,_Standard_Edition) (including [JavaFX](https://en.wikipedia.org/wiki/JavaFX)), [Java ME](https://en.wikipedia.org/wiki/Java_Platform,_Micro_Edition), [web](https://en.wikipedia.org/wiki/Web_application), [EJB](https://en.wikipedia.org/wiki/EJB) and [mobile](https://en.wikipedia.org/wiki/MIDlet) applications) out of the box.

The **NetBeans Platform** is a [framework](https://en.wikipedia.org/wiki/Software_framework) for simplifying the development of [Java Swing](https://en.wikipedia.org/wiki/Java_Swing) desktop applications. The NetBeans IDE bundle for Java SE contains what is needed to start developing NetBeans plugins and NetBeans Platform based applications; no additional SDK is required.

Applications can install modules dynamically. Any application can include the Update Center module to allow users of the application to download [digitally signed](https://en.wikipedia.org/wiki/Digital_signature) upgrades and new features directly into the running application. Reinstalling an upgrade or a new release does not force users to download the entire application again.

The platform offers reusable services common to desktop applications, allowing developers to focus on the logic specific to their application.

Among the features of the platform are:

* User interface management (e.g. menus and toolbars)
* User settings management
* Storage management (carries out efficient storage)
* Window management
* Wizard framework (supports step-by-step dialogs)
* NetBeans Visual Library
* Integrated development tools

**IntelliJ IDEA**

IntelliJ IDEA is a special programming environment or integrated development environment (IDE) largely meant for Java. This environment is used especially for the development of programs. It is developed by a company called JetBrains, which was formally called IntelliJ. It is available in two editions: the Community Edition which is licensed by Apache 2.0, and a commercial edition known as the Ultimate Edition. Both of them can be used for creating software which can be sold. What makes IntelliJ IDEA so different from its counterparts is its ease of use, flexibility and its solid design.

**IntelliJ Idea tool Features:**

Smart completion and Chain completion, Static members completion, Data flow analysis, Language injection, Cross-language refactorings, Detecting duplicates, Inspections and quick-fixes, Editor-centric environment, Shortcuts for everything, Ergonomic user interface, Inline debugger, Version control, Build tools, Test runner and coverage, Decompiler, Terminal, Database tools, Application servers, Docker, First-class support for top frameworks, Many languages–one IDE

**Prerequisite:** Basics of computer language programming.

**Course Objectives:**

Upon successful completion of this Lab the student will be able to:

* Understand the concept of OOP as well as the purpose and usage principles of inheritance, polymorphism, encapsulation and method overloading.
* Understand fundamentals of programming such as variables, conditional and iterative execution, methods, etc.
* Identify classes, objects, members of a class and the relationships among them needed for a specific problem.
* Understand fundamentals of object-oriented programming in Java, including defining classes, invoking methods, using class libraries, etc.
* Create Java application programs using sound OOP practices (e.g., interfaces and APIs) and proper program structuring (e.g., by using access control identifies, automatic documentation through comments, error exception handling)
* Have the ability to write a computer program to solve specified problems.
* Develop programs using the Java Collection API as well as the Java standard class library.
* Use the Java SDK environment to create, debug and run simple Java programs.

**List of Experiments:**

1. Demonstrate Constructor Overloading and Method Overloading in JAVA.
2. Implement Inner Classes and demonstrate its access protection.
3. Implement the following:
   1. An abstract class "Shape" with the following properties: an instance variable shapeName of type String, an abstract method area (), a toString() method that returns the name of the shape.
   2. Create a subclass named "Sphere" which has radius and its area given by the formula 4\*PI\*r^2.
   3. Create a subclass named "Rectangle" which has length and width and its area is length times width.
   4. Create a subclass named "Triangle" which has base and height and its area is ½\*base\*height.
   5. Create another class which displays the calculated area.
4. Perform the following operations:
   1. Check the length and capacity of String and StringBuffer objects
   2. Reverse the contents of a string given on console and convert the resultant string in Upper Case.
   3. Input a string from the console and append it to above resultant string.
   4. Extract the substring from resultant string.
5. Create the following:
   1. A class "Account" with minimum balance 1000rs, deposit() method to deposit amount, withdraw() method to withdraw amount and also throws LessBalanceException if an account holder tries to withdraw money which makes the balance less than 1000rs.
   2. A class "LessBalanceException" which returns the statement that says "withdraw amount ( \_\_ rs) is not valid".
   3. A class which creates 2 accounts through which both deposit and withdraw operations are performed. Appropriate action has to be taken for LessBalanceException.
6. Implement Linear Queue using user defined exception handling (also use 'throw' and ‘throws’ keyword)
7. Implement the concept of Producer Consumer using synchronized threads.
8. Create the following:
   1. Create an Interface for ‘Stack’ operations.
   2. A class that implements the Stack interface and create a fixed length stack.
   3. A class that implements the Stack interface and create a dynamic length stack.
   4. A class that uses the above stacks through interface reference and does the stack operations that demonstrates the runtime binding
9. Develop the following:
   1. Create a package named "Calculator".
   2. Create some classes in the package representing some common operations like addition, subtraction, multiplication and division.
   3. Import and compile these classes in other program.
10. Create an enumeration DayofWeek with seven values SUNDAY through SATURDAY. Add a method isWorkday() to the DayofWeek class that returns true if the value on which it is called is MONDAY through FRIDAY.
11. Using File I/O streams, write a program to demonstrate file operations.
12. Write a Swing Application which uses:
    1. JTabbed Pane
    2. Each Tab should use JPanel, which includes any one component given below in each Panel
       1. ComboBox / List / Tree / Radiobutton

Experiment No: 1

**Aim:** To demonstrate Constructor overloading and method overloading in Java

**Theory:** When more than a single constructor is defined in a class, it is known as constructor overloading. Similarly, when more than one method with the same name is defined in the same class, it is known as method overloading. However, there is a restriction on such overloading. Constructors or methods can be overloaded only in a way that would allow calls to these constructors or methods to be resolved i.e. when an overloaded method is invoked, the compiler should be able to decide which of the overloaded methods is to be called by checking the arguments. For example consider an overloaded method print() having two versions. The first of these does not take any arguments while the second one takes a single argument. A statement like print(“Example”) is a call to definitely the second version of the overloaded method which requires a single String argument.

However, if both the versions of the overloaded methods take a single String argument, then the call is non resolvable. A statement like the above could be a call to either the first or to the second version of the method. Therefore, the overloaded methods here would give compilation errors. In short, it can be stated that overloading of methods or constructors can be done only if the parameter list varies in at least one of the following things: number of arguments, types of arguments or order of arguments. The return types of the overloaded method doesn't matter.

Methods calls to overloaded versions are resolved by looking at the above stated three parameters. Following statements shows a few sets of overloaded methods and also specify if that particular set is allowed in a single class. The reason is also stated in each case.

Set 1:

public void print()   
public void print ( String str)

This set is allowed since the number of arguments differ in the two cases.

Set 2:

public void print ( int a)   
public void print ( String s)

This set is also allowed since the type of parameters are different even though the number of arguments are the same. One of the overloaded versions of print accepts an integer argument while the other accepts a String argument.

Set 3:

public void print ( int x )   
public void print ( int y)

This set may appear as acceptable at the first look but it is not. These versions do not differ in either the number, order or type of arguments. The variable name in the parameter list doesn't really matter. For example, consider the following call:

print(34);

This might be a call to either the first method or the second method. Hence this set is not allowed. The ultimate rule to check the validity of overloaded methods is to see if a method call is resolvable i.e. there should be a one to one correspondence between a method call and a method. A method call should always point to a single method and there should be no doubt as to which of the methods would be called.

Set 4:

public void print ( int a, String s)   
public void print ( String s, int a )

This set is an acceptable set for overloading the methods since the parameters differ in the order in which they are specified even though the number and type of parameters is the same.

Set 5:

public void print ( int a)   
public int print ( int a)

The above sets of methods cannot be declared in the same class. As already said, the return type doesn't matter when deciding the mutual co-existence of overloaded methods. One might argue that the call can be resolved depending on whether the call requires a value to be returned. But such an argument isn't valid because, it isn't necessary that a returned value should always be used in one or the other way. Consider the following method call.

print(34);

Looking at this call, one can't say that the print() method doesn't return a value. If we had only the second version of print (the one that returns an int) out of the two stated versions above, that version would be the one to be called. The returned int would be simply ignored. In short, values returned by a method need not always be put to use and therefore the return type isn't checked to verify the validity of overloaded methods. In a similar way, we can check the extent to which we can overload constructors for a class.

**Program:** Method Overloading

package one;

class Overload

{

int a=0,b=0;

void add()

{

System.out.println("NO or DEFAULT argument add method, sum= "+(a+b));

}

//overloaded Method add

void add(int k,int j)

{

a=k;

b=j;

System.out.println("Parameterized argument add method, sum= "+(a+b));

}

}

public class One {

public static void main(String args[])

{

Overload ga=new Overload();

System.out.println("Default Method call");

ga.add();

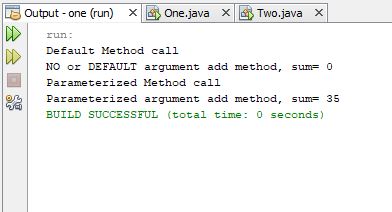
System.out.println("Parameterized Method call");

ga.add(12,23);

}

}

**Output**

****

**Program** : Constructor Overloading

package one;

class Generic

{

int a=0,b=0;

Generic()

{

a=b=1;

}

//overloaded constructor

Generic(int a,int b)

{

this.a=a;

this.b=b;

}

void print()

{

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

}

}

public class One {

public static void main(String args[])

{

System.out.println("Default constructor call");

Generic ga=new Generic();

ga.print();

System.out.println("Parameterized constructor call");

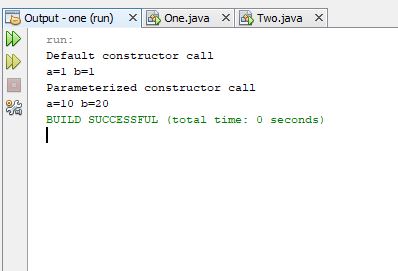
Generic ag=new Generic(10,20);

ag.print();

}

}

**Output**

****

**Experiment No. 2**

**Aim:** Toimplement Inner Classes and demonstrate its access protection.

**Theory:**

**Inner Class (Non-static nested classes):** An inner class is declared inside the curly braces of another enclosing class. **Inner class** acts as a member of the enclosing class and can have any access modifiers: **abstract, final, public, protected and private**. Inner class can access all members of the outer class including those marked private. The methods and fields of an outer class are used as if they were part of the inner class. An inner class is declared in the same way as any other class but should be inside some other class.

The basic structure of creating an inner class is as follows,

class OuteClass{

class InnerClass{

// body of InnerClass

}

//More members of OuterClass

}

From the above structure, it is clear that the class InnerClass acts as a member of the enclosing class OuterClass in much the same way as its other members. So it can be marked with an access specifier, to specify whether the class should be public, protected or private. If none of the access specifiers is specified then default package access specifier is used. These access specifiers determine whether other classes can access the inner class or not.  
More often it is the outer class that creates an instance of the inner class since it is usually the outer class that wants to use the inner instance as a helper object for its personal private use. To create an instance of the inner class, you must have the instance of the enclosing class so that you can associate them. To create an object of type InnerClass you have to write the following statement.

OuterClass outer = new OuterClass();

OuterClass.lnnerClass inner = outer.new InnerClass();

**Program:**

package two;

import java.io.\*;

class Outer

{

int outdata = 22;

void display()

{

Inner inobj = new Inner();

System.out.println("Accessing from outer class");

System.out.println("The value of outdata is " +outdata);

System.out.println("The value of indata is " +inobj.indata);

}

class Inner

{

int indata = 20;

void inmethod()

{

System.out.println("Accessing from inner class");

System.out.println("The sum of indata & outdata is " +(outdata + indata));

}

}

}

public class Two {

public static void main(String args[])

{

Outer outobj = new Outer();

outobj.display();

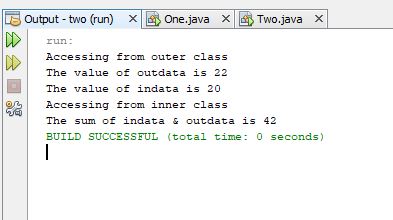
Outer.Inner inobj1 = outobj.new Inner();

inobj1.inmethod();

}

}

**Output**

****

**Experiment No. 3**

**Aim: To implement the following**

i) An abstract class "Shape" with the following properties: an instance variable shapeName of type String, an abstract method area(), a toString() method that returns the name of the shape.

ii) Create a subclass named "Sphere" which has radius and its area given by the formula 4\*PI\*r^2.

iii) Create a subclass named "Rectangle" which has length and width and its area is length times width.

iv) Create a subclass named "Triangle" which has base and height and its area is ½\*base\*height.

v) Create an another class which displays the calculated area.

**Theory:** An abstract class is a class that is declared abstract. It may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed. An abstract method is a method that is declared without an implementation (without braces, and followed by a semicolon), like this: abstract void moveTo(double deltaX, double deltaY);

If a class includes abstract methods, then the class itself *must* be declared abstract. Objects of an abstract class cannot be created. Even then, constructors may be provided for an abstract class for use by its sub classes. An abstract class is made usable by extending it to a sub class. The sub class needs to provide implementations of all the abstract methods defined in its super class, just in the way methods are overridden.

**Program:**

package three;

import java.util.Scanner;

abstract class Shape

{

final double PI=3.14; //constant

String shapeName;

abstract void area(); //abstract method

public String toString() //tostring method to return calling classname

{

return this.getClass().getName();

}

}

class Sphere extends Shape

{

double rad;

Sphere(double r) //constructor to set radius

{

rad=r;

}

void area()

{

System.out.println("Area = "+(4\*PI\*rad\*rad)); //area of sphere formula

}

}

class Rectangle extends Shape

{

double len, width;

Rectangle(double l,double w)

{

len=l;

width=w;

}

void area()

{

System.out.println("Area = "+(len\*width));

}

}

class Triangle extends Shape

{

double base, height;

Triangle(double b,double h)

{

base=b;

height=h;

}

void area()

{

System.out.println("Area = "+(0.5\*base\*height));

}

}

public class Three {

public static void main(String args[])

{

Scanner s=new Scanner(System.in);

System.out.println("Enter the radius of a circle");

double r=s.nextInt(); //taking radius input

Sphere sp=new Sphere(r); //sphere object creation

System.out.println(sp.toString()); //respective class object calls tostring()

sp.area(); //calling respective class area method

System.out.println("Enter the len and wid: ");

double l=s.nextInt();

double w=s.nextInt();

Rectangle rc=new Rectangle(l,w);

System.out.println(rc.toString());

rc.area();

System.out.println("Enter the base and height: ");

double b=s.nextInt();

double h=s.nextInt();

Triangle ta=new Triangle(b,h);

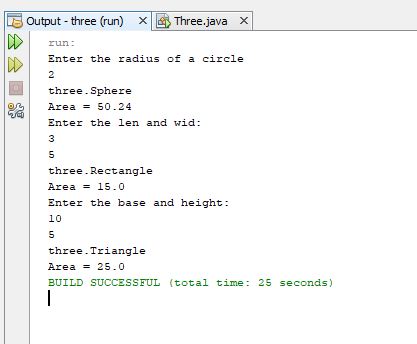
System.out.println(ta.toString());

ta.area();

}

}

**Output:**



Experiment No: 4

**Aim:** To perform following operations

i) Check the length and capacity of String and StringBuffer objects

ii) Reverse the contents of a string given on console and convert the resultant string in

Upper Case.

iii) Input a string from the console and append it to above resultant string.

iv) Extract the substring from resultant string.

**Theory:** Java provides String and String Buffer classes. String represents fixed-length, immutable character sequences while StringBuffer represents growable and writable character sequences.  String class is used to manipulate character strings that cannot be changed. Simply stated, objects of type String are read only and immutable. The StringBuffer class is used to represent characters that can be modified.

StringBuffer Class Construcors are

1.StringBuffer() : Creates an empty string buffer whose initial capacity is 16 characters.

2. StringBuffer(int): Creates an empty string buffer with the specified initial capacity.

3. StringBuffer(String): Creates a string buffer whose value is initialized by the specified String. The capacity of the string buffer is the length of the original string plus 16.

Some of the most used methods are

1. length() and capacity():The length of a StringBuffer can be found by the length( ) method, while the total allocated capacity can be found by the capacity( ) method.

2. append():It is used to add text at the end of the existence text.

Some of the Constructors used in String Class are

1. String(): Create an empty string

2.String(byte[]) : Creates a string whose value is set from the contents of an array of bytes.

3.String(char[]) : Creates a string whose value is set from the contents of an array of characters.  
4.String(String) : Creates a string whose value is set from another string. Using this constructor with a literal string argument is not recommended, because it creates two identical strings.

5.String(StringBuffer): Creates a string whose value is set from a string buffer.

6.String(StringBuilder) : Creates a string whose value is set from a string builder.

**Program:**

package four;

import java.util.\*;

class Cases

{

//length and capacity of string and stringbuffer

String s="Jain University"; //String initialization

StringBuffer sb1=new StringBuffer(); //String buffer with default size but no input

StringBuffer sb2=new StringBuffer("SETJU"); //String buffer with input

StringBuffer sb3=new StringBuffer(50); //String buffer with capacity input

void demo1()

{

System.out.println("Case1");

System.out.println("with input, string length is: "+s.length()); //String length

System.out.println("Before input, stringbuffer length is: "+sb1.length()); //StringBuffer length

System.out.println("With input, stringbuffer length is: "+sb2.length());

System.out.println("With capacity input, stringbuffer length is: "+sb3.length());

System.out.println("Before input, stringbuffer length is: "+sb1.capacity()); /\* StringBuffer

capacity \*/

System.out.println("With input, stringbuffer length is: "+sb2.capacity());

System.out.println("With capacity input, stringbuffer length is: "+sb3.capacity());

}

String demo2(String read)

{

String reverse = "";

int i=read.length()-1;

while(i>=0) {

reverse = reverse + read.charAt(i--);

}

return reverse;

}

String demo3(String read1, String read2)

{

return read1.concat(read2);

}

void demo4(String read)

{

System.out.println("\nCase4");

System.out.println("Given only begining index as 2 "+read.substring(2));

System.out.println("Given begining and end index as (2 to 4) "+read.substring(2,4));

}

}

public class Four {

public static void main(String args[])

{

Scanner sc=new Scanner(System.in);

Cases c1=new Cases();

c1.demo1();

System.out.println("\nCase2");

System.out.println("Enter string to reverse: ");

String s1=sc.nextLine();

String res1=c1.demo2(s1);

System.out.println("Reversed string is "+res1);

System.out.println("\nCase3");

System.out.println("Enter string to append: ");

String s2=sc.nextLine();

String res2=c1.demo3(res1,s2);

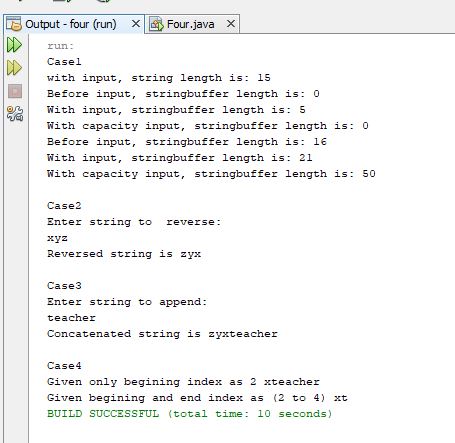
System.out.println("Concatenated string is "+res2);

c1.demo4(res2);

}

}

**Output:**

****

Experiment No: 5

Aim: To implement the following

i) A class "Account" with minimum balance 1000rs, deposit() method to deposit amount, withdraw() method to withdraw amount and also throws LessBalanceException if an account holder tries to withdraw money which makes the balance less than 1000rs.

ii) A class "LessBalanceException" which returns the statement that says "withdraw amount ( \_\_ rs) is not valid".

iii) A class which creates 2 accounts through which both deposit and withdraw operations are performed. Appropriate action has to be taken for LessBalanceException.

Theory: An exception (or exceptional event) is a problem that arises during the execution of a program. When an Exception occurs the normal flow of the program is disrupted and the program or Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled. Exceptions in java can arise from different kind of situations such as wrong data entered by user, hardware failure, network connection failure, Database server down etc.

**Program:**

package five;

import java.util.\*;

class LessBalanceException extends Exception //Defining user-defined exception class

“LessBalanceException”

{

LessBalanceException(String s) //Constructor which takes user error message

{

super(s); //calls super class constructor

}

}

class Account //class account

{

double minbal;

Account()

{

minbal=1000; //minimum balance of any account is made 1000rs

}

void deposit(double amt)

{

minbal=minbal+amt; //deposit amount is added to minimum balance

}

void withdraw(double amt)

{

double t;

t=minbal;

t=t-amt;

try {

if (t < 1000) //check against balance not to be lesser than 1000rs at any time

throw new LessBalanceException("Balance can't go less then 1000 (Withdraw amount is not

valid)");

else {

minbal = minbal - amt; //withdraw amount is subtracted from latest minimum balance

System.out.print("Balance after withdraw:");

System.out.println(minbal);

}

}

catch(LessBalanceException l)

{

System.out.println(l.getMessage());

}

}

double balance()

{

return minbal;

}

}

public class Five {

public static void main(String args[]) {

Account a1=new Account(); //creating account1

Account a2=new Account(); //creating account2

double dep,wit;

Scanner sc=new Scanner(System.in);

System.out.print("Min bal of account1: ");

System.out.println(a1.balance()); //account1 minimum balance

System.out.println("Enter the amount to deposit: ");

dep = sc.nextDouble();

a1.deposit(dep); //account1 deposit method call

System.out.print("Balance after deposit:");

System.out.println(a1.balance());

System.out.println("Enter the amount to Withdraw: ");

wit = sc.nextDouble();

a1.withdraw(wit); //account1 withdraw method call

System.out.print("\nMin bal of account2: ");

System.out.println(a2.balance()); //account2 minimum balance

System.out.println("Enter the amount to deposit: ");

dep = sc.nextDouble();

a2.deposit(dep); //account2 deposit method call

System.out.print("Balance after deposit:");

System.out.println(a2.balance());

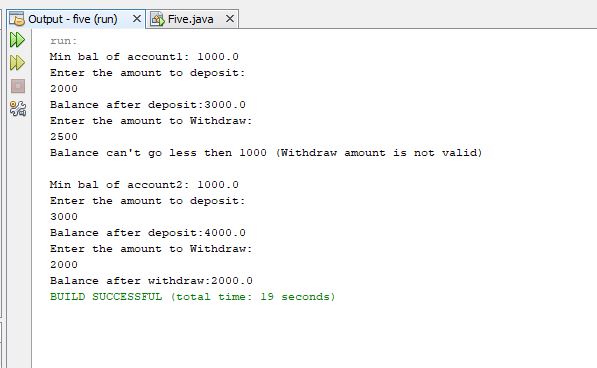
System.out.println("Enter the amount to Withdraw: ");

wit = sc.nextDouble();

a2.withdraw(wit); //account2 withdraw method call

} }

**Output:**



Experiment No: 6

**Aim:** Toimplement Linear Queue using user defined exception handling (also use 'throw' and ‘throws’ keyword).

**Theory:** In java user can create their own exception class and throw that exception using throw keyword. These exceptions are known as **user-defined** or **custom** exceptions. Exception classes can be created by extending the Exception class.The extended class contains constructors, data members and methods like any other class. The throw and throws keywords are used while implementing the user-defined exceptions.

The **throw** keyword in Java is used to explicitly throw an exception from a method or any block of code. We can throw either [checked or unchecked exception](https://www.geeksforgeeks.org/checked-vs-unchecked-exceptions-in-java/). The throw keyword is mainly used to throw custom exceptions.

Syntax: **throw Instance**

Example:

**throw new ArithmeticException("/ by zero");**

The **throws** is a keyword in Java which is used in the signature of method to indicate that this method might throw one of the listed type exceptions. The caller to these methods has to handle the exception using a try-catch block.

Syntax: **type method\_name(parameters) throws exception\_list**

exception\_list is a comma separated list of all the exceptions which a method might throw.

**Program:**

package six;

class LinearQueue extends Exception //user-defined exception class “LinearQueue”

{

int front,rear;

final int size=5;

int q[]=new int[size];

LinearQueue()

{

front=0;

rear=-1;

}

LinearQueue(String s)

{

super(s); //calling constructor of super class Exception

}

void enqueue(int ele) throws LinearQueue

{

try {

if (rear == -1 || rear < size-1) {

q[++rear] = ele;

System.out.println("Q has "+q[rear]+ " at pos "+rear);

}

else {

throw new LinearQueue("Queue is full\n");

}

}

catch(LinearQueue lq) {

System.out.println(lq.getMessage());

}

}

void dequeue() throws LinearQueue

{

try {

if (front < size) {

System.out.println("Deleted element is " + q[front++]);

}

else {

if(front==size)

front=-1;

throw new LinearQueue("No elements to delete");

}

}

catch(LinearQueue lq)

{

System.out.println(lq.getMessage());

}

}

void display()

{

if(front>-1)

{

System.out.println("Elements are");

int k=front;

for(int i=k;i<=rear;i++)

System.out.println(q[k++]);

}

} }

public class Six

{

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

{

LinearQueue q=new LinearQueue();

q.enqueue(10);

q.enqueue(20);

q.enqueue(30);

q.enqueue(40);

q.enqueue(50);

q.display();

System.out.println("When tried to put sixth element to full queue");

q.enqueue(60);

q.dequeue();

q.dequeue();

q.dequeue();

q.display();

q.dequeue();

q.dequeue();

System.out.println("When tried to remove sixth element from empty queue");

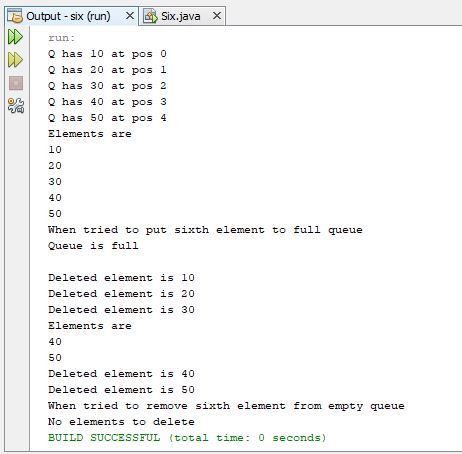
q.dequeue();

q.display();

}

}

**Output:**



Experiment No: 7

**Aim:** To implement the concept of Producer Consumer using synchronized threads.

**Theory:** [Multi-threaded](http://quiz.geeksforgeeks.org/multithreading-in-java/)programs may often come to a situation where multiple threads try to access the same resources and finally produce erroneous and unforeseen results. For example, if multiple threads try to write within a same file then they may corrupt the data because one of the threads can override data or while one thread is opening the same file at the same time another thread might be closing the same file.

Java provides a way of creating threads and synchronizing their task by using synchronized blocks. Synchronized blocks in Java are marked with the synchronized keyword. A synchronized block in Java is synchronized on some object. All synchronized blocks synchronized on the same object can only have one thread executing inside them at a time. All other threads attempting to enter the synchronized block are blocked until the thread inside the synchronized block exits the block.

Syntax :

Synchronized(objectidentifier){

// Access shared variables and other shared resources

}

Where the objectidentifier is a reference to an object whose lock associates with the monitor that the synchronized statement represents.

**Program:**

package restaurant;

import java.io.\*;

class Order

{

private static int i=0;

private int count=i++;

public Order()

{

if(count==10)

{

System.out.println("\n out of food stock");

System.exit(0);

}

}

public String toString()

{

return "Order" +count;

}

}

class waitperson extends Thread

{

private Restaurant rest;

public waitperson(Restaurant r)

{

rest = r;

start();

}

public void run()

{

while(rest.order==null)

synchronized(this)

{

try

{

wait();

}

catch(InterruptedException e)

{

throw new RuntimeException(e);

}

System.out.println("wait person got"+rest.order);

rest.order=null;

}

}

}

class chef extends Thread

{

private Restaurant rest;

private waitperson wp;

public chef(Restaurant r,waitperson w)

{

rest = r;

wp = w;

start();

}

public void run()

{

while(true)

{

if(rest.order==null)

{

rest.order = new Order();

System.out.println("order up");

synchronized(wp)

{

wp.notify();

}

}

try

{

sleep(1000);

}

catch(InterruptedException e)

{

throw new RuntimeException(e);

}

}

}

}

class Restaurant

{

Order order;

public static void main(String args[])

{

Restaurant rest = new Restaurant();

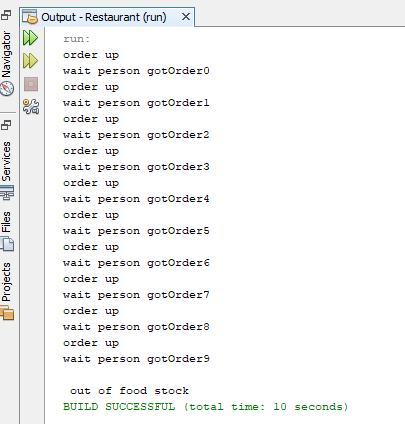
waitperson wp = new waitperson(rest);

chef ch = new chef(rest,wp);

}

}

**Output:**

****

Experiment No: 8

**Aim:** To Create the following:

i) Create an Interface for ‘Stack’ operations.

ii) A class that implements the Stack interface and create a fixed length stack.

iii) A class that implements the Stack interface and create a dynamic length stack.

iv) A class that uses the above stacks through interface reference and does the stack

operations that demonstrates the runtime binding.

**Theory:** An interface definition is similar to a class definition except that it uses the interface keyword. All methods in an interface are abstract methods, that is, they are declared without the implementation part since they are to be implemented in the subclasses that use them. To declare an interface, use **interface** keyword. It is used to provide total abstraction. That means all the methods in interface are declared with empty body and are public and all fields are public, static and final by default. A class that implement interface must implement all the methods declared in the interface. To implement interface use **implements** keyword.

**Program:**

package stackinterface;

import java.io.\*;

interface operations

{

void push();

void pop();

void display();

}

//Class Of Fixed\_Stack

class Fstack implements operations

{

private int top=-1;

private int[] fstack=new int[5];

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

public void push()

{

if(top==fstack.length-1)

System.out.println("\nStack Overflow");

else

{

try

{

System.out.print("Enter the Item:\t");

fstack[++top]=Integer.parseInt(br.readLine());

}

catch(Exception e)

{

System.out.println(e);

}

}

}

public void pop()

{

if(top==-1)

System.out.println("\nStack Underflow...!");

else

System.out.println("\nThe Deleted Item Is:\b"+fstack[top--]);

}

public void display()

{

if(top==-1)

System.out.println("\nStack Is Empty");

else

{

System.out.println("The Elements Of Stack are:");

for(int i=top;i>=0;i--)

System.out.println(fstack[i]);

}

}

}

//Class Of Dynamic\_Stack

class Dstack implements operations

{

private int top=-1;

private int[] dstack;

Dstack(int size)

{

dstack=new int[size];

}

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

public void push()

{

if(top==dstack.length-1)

System.out.println("\nStack Overflow");

else

{

try

{

System.out.print("Enter the Item:\t");

dstack[++top]=Integer.parseInt(br.readLine());

}

catch(Exception e)

{

System.out.println(e);

}

}

}

public void pop()

{

if(top==-1)

System.out.println("\nStack Underflow...!");

else

System.out.println("\nThe Deleted Item Is:\b"+dstack[top--]);

}

public void display()

{

if(top==-1)

System.out.println("\nStack Is Empty");

else

{

System.out.println("The Elements Of Stack are:");

for(int i=top;i>=0;i--)

System.out.println(dstack[i]);

}

}

}

//Main Class

class Stackinterface

{

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

{

int ch;

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

while(true)

{

System.out.println("\n1.Fixed\_Stack(5)\n2.Dynamic\_Stack\n3.Exit");

System.out.print("\nEnter Your Choice:\t");

ch=Integer.parseInt(br.readLine());

switch(ch)

{

case 1: operations obj=new Fstack();

while(true)

{

System.out.println("\n1.PUSH\n2.POP\n");

System.out.println("3.DISPLAY\n4.MAIN\_MENU");

System.out.print("Enter Your Choice:\t");

ch=Integer.parseInt(br.readLine());

switch(ch)

{

case 1:obj.push();

break;

case 2:obj.pop();

break;

case 3:obj.display();

break;

case 4:break;

default:System.out.println("Invalid Choice");

}

if(ch==4)

break;

}

break;

case 2: System.out.print("Enter The Size Of Array:\t");

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

operations obj1=new Dstack(size);

while(true)

{

System.out.println("\n1.PUSH\n2.POP\n");

System.out.println("\n3.DISPLAY\n4.MAIN\_MENU");

System.out.print("Enter Your Choice:\t");

ch=Integer.parseInt(br.readLine());

switch(ch)

{

case 1:obj1.push();

break;

case 2:obj1.pop();

break;

case 3:obj1.display();

break;

case 4:break;

default:System.out.println("Invalid Choice");

}

if(ch==4)

break;

}

break;

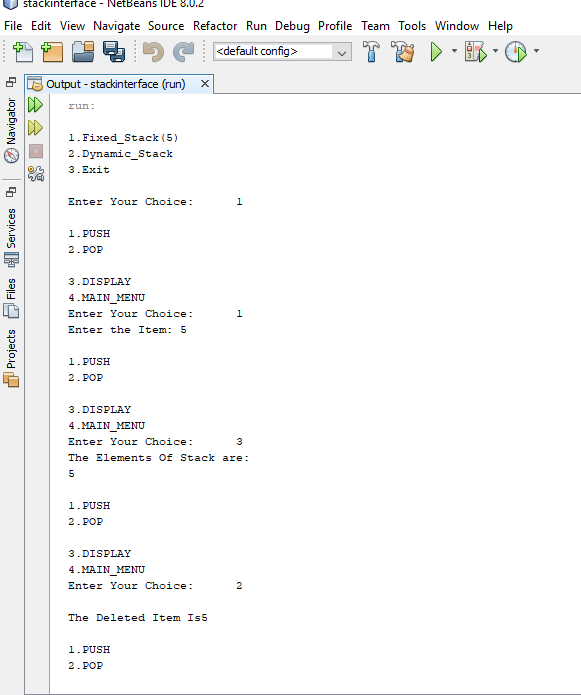
case 3: System.exit(0);

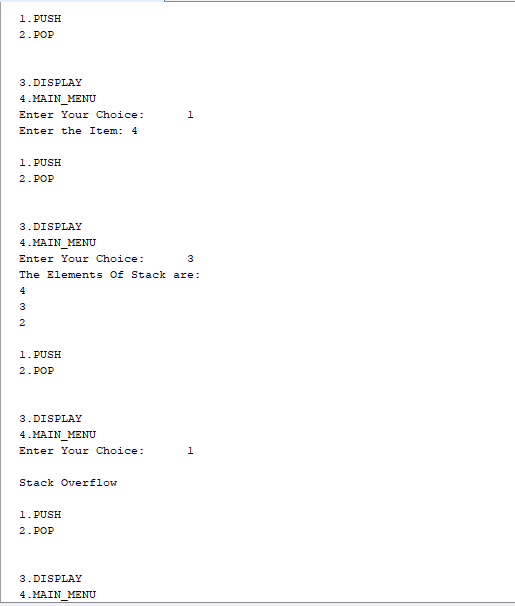
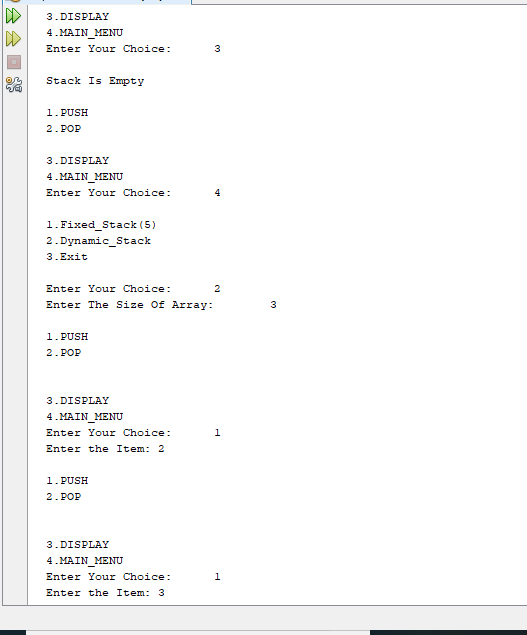
System.out.println("\n Invalid Choice");

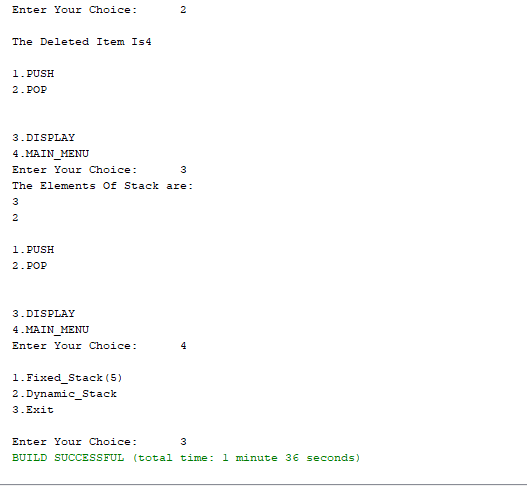
}

} } }

**Output:**







Experiment No: 9

**Aim:** Develop the following:

i) Create a package named "Calculator".

ii) Create some classes in the package representing some common operations like addition, subtraction, multiplication and division.

iii) Import and compile these classes in other program.

**Theory:** While class definitions are practical modular units, the Java programming language has another mechanism that facilitates programming teams and reusable software units. The package facility allows for appropriate classes to be grouped into packages. As with standard design rationale for objects where relevant methods are placed in the same class definition, packages in Java form the next level of software containment for classes with logically related functionality. Packaging also partitions the name space to avoid name clashes. Computations in Java are reliant on objects, and the result of system design is a set of class definitions. Where teams of programmers work independently with the intention of the results to be subsequently integrated, there is a chance that they may choose the same name for their classes. Packaging thus allows for the names of classes to be confined to the originating package**.**

Package hierarchy is specified via the package keyword preceding a class definition as shown below. Here, class XYZ belongs within package A. Its complete qualified name is thus A.XYZ.

package A;

class XYZ {

int h;

void j() { ... }

}

**Program:**

addition.java

package Calculator;

public class addition {

public int add(int a, int b){

return (a+b);

}

}

subtraction.java

package Calculator;

public class subtraction {

public int sub(int a, int b){

return (a-b);

}

}

multiplication.java

package Calculator;

public class multiplication {

public int mul(int a, int b){

return (a\*b);

}

}

division.java

package Calculator;

public class division {

public int div(int a, int b){

return (a/b);

}

}

op.java

package op;

import Calculator.\*;

import java.util.Scanner;

public class Op {

public static void main(String[] args) {

while(true){

System .out.println("1.Addition"+"\n"+"2.Subtraction"+"\n"+"3.Multiplication"+"\n"+"4.Division"+"\n"+"5.Exit");

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

Scanner in= new Scanner(System.in);

int n= in.nextInt();

if(n==5){

System.exit(0);

}

System.out.println("enter the number");

int a= in.nextInt();

int b= in.nextInt();

switch(n){

case 1: addition ad= new addition();

int k= ad.add(a,b);

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

break;

case 2: subtraction sb= new subtraction();

int l= sb.sub(a, b);

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

break;

case 3: multiplication mu= new multiplication();

int m= mu.mul(a, b);

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

break;

case 4: division dv= new division();

int h= dv.div(a, b);

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

break;

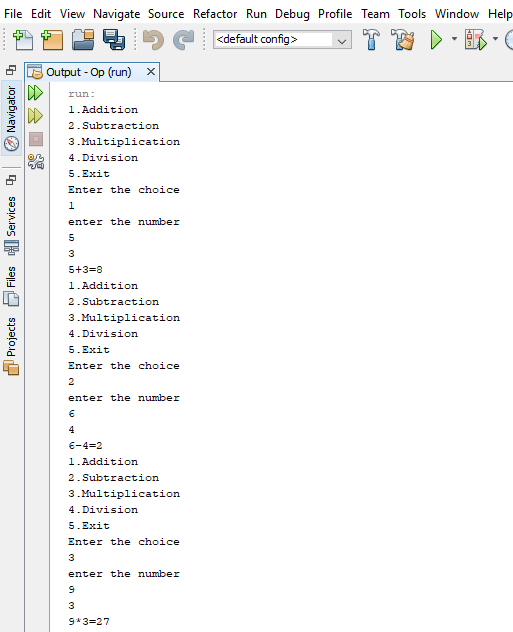
}

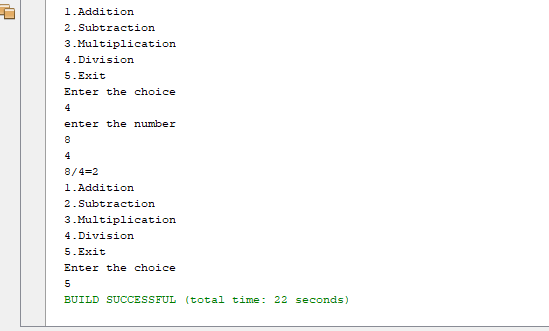
}

}

}

Output:

****

****Experiment No: 10

**Aim:** Create an enumeration DayofWeek with seven values SUNDAY through SATURDAY. Add a method isWorkday() to the DayofWeek class that returns true if the value on which it is called is MONDAY through FRIDAY.

**Theory:** Enumeration is basically a list of named constant. In Java, it defines a class type. It can have [constructors](https://www.edureka.co/blog/constructor-in-java/), methods and [instance variables](https://www.edureka.co/blog/instance-variable-in-java/). It is created using the enum keyword. By default, each enumeration constant is [public](https://www.edureka.co/blog/access-modifiers-in-java/#Public_Access_Modifier), static and final.

Example:

enum Fruit {

APPLE, MANGO, ORANGE;

}

Enumeration can be define either inside or outside class. But we can’t define enum inside a method.

To refer the constants in the enum above:

Fruit fruit1= Fruit.MANGO;

The fruit1 variable can take one of the Fruit enum constants as a value (APPLE, MANGO, ORANGE). In this cages fruit1 is set to MANGO.

**Enumeration with Constructor**: Constructor is executed separately for each enum constant at the time of enum class loading. You can add fields to a Java enum. Thus, each constant enum value gets these fields. The field values must be supplied to the constructor of the enum when defining the constants.

enum Fruit{

APPLE(1),MANGO(2),ORANGE(3);// Calls constructor with corresponding values.

int value;

Fruit(int value){

this.value=value;}

}

public class F1 {

public static void main(String[] args) {

Fruit f= Fruit.MANGO;

System.out.println(f.value);

} }

In the above example has a constructor which take an int value. The enum constructor sets the int field. When a constant enum values are defined, an int value is passed to the enum constructor. The enum constructor must be either private or package scope (default).

**Program**

package ten;

enum DayofWeek{ MONDAY(1),TUESDAY(2),WEDNESDAY(3),THURSDAY(4),FRIDAY(5),SATURDAY(6),SUNDAY(7);

int val;

DayofWeek(int val){

this.val=val;

}

boolean isWorkday() {

if (val<6){

return true;

}

else {

return false;

}

}

}

public class Ten{

public static void main(String[] args) {

DayofWeek Day= DayofWeek.SUNDAY;

System.out.println(" verfication of sunday(isWorkDay())");

System.out.println(Day.isWorkday());

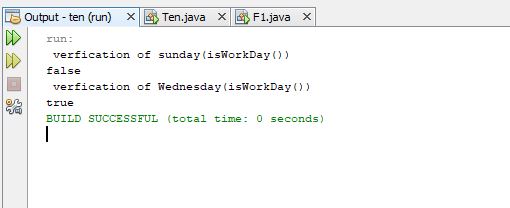
System.out.println(" verfication of Wednesday(isWorkDay())");

System.out.println(DayofWeek.WEDNESDAY.isWorkday());

}

}

**Output**

****

Experiment No: 11

**Aim:** Using File I/O streams, write a program to demonstrate file operations.

**Theory:** Some of the common file handling operations are;

1. Create file
2. Delete file
3. Read file
4. Write file
5. Change file permissions

We can use File class **createNewFile()** method to create new file. This method returns true if file is successfully created, otherwise it returns false. File class **delete()** method is used to delete a file or an empty directory. File delete method returns true if file is deleted successfully or else it returns false.

FileInputStream- This stream is used for reading data from the files. Objects can be created using the keyword **new** and there are several types of constructors available.

Following constructor takes a file name as a string to create an input stream object to read the file - InputStream f = new FileInputStream("C:/java/hello");

Following constructor takes a file object to create an input stream object to read the file. First we create a file object using File() method as follows –

File f = new File(“C:/java/hello”);

InputStream file = new FileInputStram(f);

FileOutputStream- FileOutputStream is used to create a file and write data into it. The stream would create a file, if it doesn't already exist, before opening it for output.Here are two constructors which can be used to create a FileOutputStream object.Following constructor takes a file name as a string to create an input stream object to write the file −

OutputStram out= new FileOutputStream(“C:/java/hello”);

Following constructor takes a file object to create an output stream object to write the file. First, we create a file object using File() method as follows –

File f1 = new File(“C:/java/hello”);

OutputStram out= new FileOutputStream(f1);

**Program:**

package fileprogram;

import java.io.\*;

public class Fileprogram {

public static void main (String s[]) throws FileNotFoundException, IOException

{

File f1 = new File("C:/exam/input.txt");

File f2 = new File("C:/exam/output.txt");

if(!f1.exists()) // If file is not created then create file

f1.createNewFile();

FileInputStream in = new FileInputStream(f1);

FileOutputStream out= new FileOutputStream(f2);

int k;

while ((k=in.read())!=-1){

System.out.println((char)k);

}

int c;

while ((c = in.read()) != -1) {

out.write(c); // writing

}

File f= new File("data.txt");

if(f.createNewFile()){

System.out.println("File created");

}

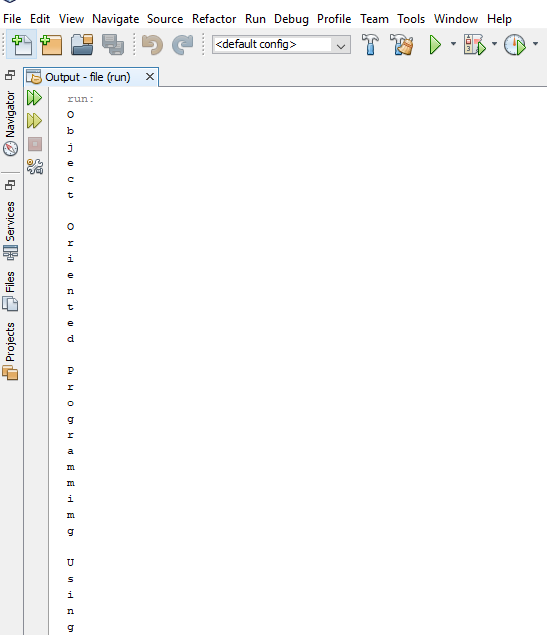
boolean de= f.delete(); // File deletion

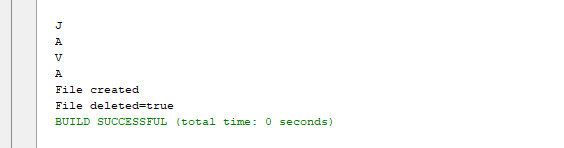
System.out.println("File deleted="+de):

}

}

**Output:**

****

****

Experiment No: 12

**Aim:** To create a Swing Application which uses:

* 1. JTabbed Pane
  2. Each Tab should use JPanel, which includes any one component given below in each Panel
     1. ComboBox / List / Tree / Radiobutton

**Theory: Java Swing** is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java. The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

There are two ways to create a frame:

* By creating the object of Frame class (association)
* By extending Frame class (inheritance)

We can write the code of swing inside the main(), constructor or any other method.

**JTabbed Pane:** The JTabbedPane class is used to switch between a group of components by clicking on a tab with a given title or icon. It inherits JComponent class.**JComboBox:** The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits JComponent class.

**Commonly used Constructor are:**

1. **JTabbedPane()**: Creates an empty TabbedPane with a default tab placement of JTabbedPane.Top.
2. **JTabbedPane(int tabPlacement)** : Creates an empty TabbedPane with a specified tab placement.
3. **JTabbedPane(int tabPlacement, int tabLayoutPolicy):** Creates an empty TabbedPane with a specified tab placement and tab layout policy.

**JList:** The object of JList class represents a list of text items. The list of text items can be set up so that the user can choose either one item or multiple items. It inherits JComponent class.

**Commonly used Constructor are :**

1. **JList()**: creates an empty blank list
2. **JList(E [ ] l)**: creates an new list with the elements of the array.
3. **JList(ListModel d)**: creates a new list with the specified List Model.
4. **JList(Vector l)** : creates a new list with the elements of the vector.

**JTree:** The JTree class is used to display the tree structured data or hierarchical data. JTree is a complex component. It has a 'root node' at the top most which is a parent for all nodes in the tree. It inherits JComponent class.

**Commonly used Constructor are:**

1. **JTree():**Creates a JTree with a sample model.
2. **JTree(Object[] value):** Creates a JTree with every element of the specified array as the child of a new root node.
3. **JTree(TreeNode root):** Creates a JTree with the specified TreeNode as its root, which displays the root node.

**JRadioButton:** The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz. It should be added in ButtonGroup to select one radio button only.

**Commonly used Constructor are:**

1. **JRadioButton():**Creates an unselected radio button with no text.
2. **JRadioButton(String s):** Creates an unselected radio button with specified text.
3. **JRadioButton(String s, boolean selected):** Creates a radio button with the specified text and selected status.

**Program:**

import java.awt.Dimension;

import javax.swing.\*;

import javax.swing.tree.DefaultMutableTreeNode;

public class TabbedPaneExamples {

TabbedPaneExamples(){

JFrame f=new JFrame();

JTabbedPane tp=new JTabbedPane();

tp.setBounds(50,50,200,200);

tp.add("CombokBox",new combopanel());

tp.add("List",new List());

tp.add("Tree",new Treepanel());

tp.add("Radiobotton",new rpanel());

f.add(tp);

f.setSize(500,600);

f.setLayout(null);

f.setVisible(true);

f.setDefaultCloseOperation(f.EXIT\_ON\_CLOSE);

}

public static void main(String[] args) {

new TabbedPaneExamples();

}

}

class combopanel extends JPanel{

combopanel(){

JComboBox combo=new JComboBox();

combo.addItem("Java");

combo.addItem("C++");

combo.addItem("C");

add(combo);

}

}

class List extends JPanel{

List(){

String cities[]={"Bangalore","Chennai","Hyderabad","Bombay"};

JList list= new JList(cities);

JScrollPane scr= new JScrollPane(list);

scr.setPreferredSize(new Dimension(120,90));

add(scr);

}

}

class Treepanel extends JPanel{

Treepanel(){

DefaultMutableTreeNode top=new DefaultMutableTreeNode("Programmes");

DefaultMutableTreeNode a=new DefaultMutableTreeNode("B.Tech programs");

top.add(a);

DefaultMutableTreeNode a1=new DefaultMutableTreeNode("AI");

DefaultMutableTreeNode a2=new DefaultMutableTreeNode("CTIS");

DefaultMutableTreeNode a3=new DefaultMutableTreeNode("MACT");

DefaultMutableTreeNode a4=new DefaultMutableTreeNode("Data Science");

DefaultMutableTreeNode a5=new DefaultMutableTreeNode("IS");

a.add(a1); a.add(a2);a.add(a3);a.add(a4);a.add(a5);

JTree ob= new JTree(top);

add(ob);

}

}

class rpanel extends JPanel{

rpanel(){

ButtonGroup bg= new ButtonGroup();

JRadioButton m= new JRadioButton("Male");

add(m);

JRadioButton f= new JRadioButton("Female");

add(f);

bg.add(m);

bg.add(f);

}

}

**Output:**

