MANIPAL INSTITUTE OF TECHNOLOGY

MANIPAL

(A constituent unit of MAHE, Manipal)

Manipal Academy of Higher Education, Manipal – 576 104

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that Ms./Mr. .....

Reg. No.: ..... Section: ..... Roll No.: .....

has satisfactorily completed the **LAB EXERCISES PRESCRIBED FOR OBJECT ORIENTED PROGRAMMING LAB (CSE 2163)** of Second Year B.Tech. Degree in Computer Science and Engineering at MIT, Manipal, in the Academic Year 2022– 2023.

Date: .....

Signature
Faculty In Charge

CONTENTS

Course Objectives

- To outline the concepts of object orientation using Java
- To implement and execute application programs
- To develop skills in concurrent programming
- To develop efficient Graphical User Interfaces (GUI) using JavaFx components
- To outline the event handling mechanism of Java

Course Outcomes

At the end of this course, students will be able to

- Outline object-oriented paradigm of software development
- Achieve reusability using inheritance, packages, and generics
- Appreciate the use of exception handling, achive concurrency through multithreading, and implement small java applications using Design and implement small Java applications using JavaFx

Evaluation plan

- Internal Assessment Marks : 60%
  - ✓ Continuous evaluation component (for each experiment):10 marks
  - ✓ Mini Project: 10 marks
  - ✓ The assessment will depend on punctuality, program execution, maintaining the observation note and answering the questions in viva voce
  - ✓ Total marks of the 12 experiments and Mini Project reduced to marks out of 60
- End semester assessment of 2 hour duration: 40 %

INSTRUCTIONS TO THE STUDENTS

Pre-Lab Session Instructions

1. Students should carry the Lab Manual Book and the required stationery to every lab session
2. Be in time and follow the institution dress code
3. Must Sign in the log register provided
4. Make sure to occupy the allotted seat and answer the attendance

5. Adhere to the rules and maintain the decorum

In-Lab Session Instructions

- Follow the instructions on the allotted exercises
- Show the program and results to the instructors on completion of experiments
- On receiving approval from the instructor, copy the program and results in the Lab record
- Prescribed textbooks and class notes can be kept ready for reference if required

General Instructions for the exercises in Lab

- Implement the given exercise individually and not in a group.
- The programs should meet the following criteria:
  - Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs.
  - Programs should perform input validation (Data type, range error, etc.) and give appropriate error messages and suggest corrective actions.
  - Comments should be used to give the statement of the problem and every member function should indicate the purpose of the member function, inputs and outputs.
  - Statements within the program should be properly indented.
  - Use meaningful names for variables, classes, interfaces, packages and methods.
  - Make use of constant and static members wherever needed.
- Plagiarism (copying from others) is strictly prohibited and would invite severe penalty in evaluation.
- The exercises for each week are divided under three sets:
  - Solved exercise
  - Lab exercises – to be completed during lab hours
  - Additional Exercises – to be completed outside the lab or in the lab to enhance the skill
- In case a student misses a lab class, he/she must ensure that the experiment is completed during the repetition class with the permission of the faculty concerned but credit will be given only to one day's experiment(s).

- Questions for lab tests and examination are not necessarily limited to the questions in the manual, but may involve some variations and / or combinations of the questions.
- A sample note preparation is given as a model for observation.

Mini Project Objectives and Guidelines

Objectives:

- Select a domain and identify Object Oriented Principles in it.
- Formulate the synopsis for mini project by briefly describing the domain.
- List the requirements that can be implemented in the project.
- Implement the functional requirements by identifying appropriate classes. Make use of Inheritance, incorporate exception handling undesired scenarios. (Depending on the requirements and functionalities, do not limit to use only these concepts – Multithreading, Design of Interfaces and properly packaging your code may be performed)
- Design JavaFx based UI using various components such as TextBox, ListView, Button etc suitably to invoke one or more functions implemented

Guidelines: All the students are instructed to form a team of four members. The team members must be from the same lab batch. Once the synopsis is submitted, no change of team member(s) shall be entertained. A single copy of the Synopsis must be submitted by the team on or before the end of the fifth week. Title and code of the mini project should be unique among the teams.

Synopsis Format:

- Title, Team Members
- Abstract (Briefly describe the selected domain and various functionalities that can be modelled via Object Oriented Concepts)
- Flowchart to show the entire working model of the project
- Functional Requirements (List all the functions that the working project can demonstrate)
- Expected output of the project

The students should not

- Bring mobile phones or any other electronic gadgets to the lab.
- Go out of the lab without permission.

SAMPLE LAB OBSERVATION NOTE PREPARATION

A Java program Sample.java to display **Hello Java** message

```
class Sample{
    public static void main(String args[ ]){
        System.out.println("Hello Java");
    }
}
```

Sample input and output:

Hello Java

Introduction to Java

Java is both compiled and interpreted. Programs written in Java are compiled into machine language, but it is a machine language for a computer that doesn't really exist. This so-called "virtual" computer is known as the Java virtual machine (JVM). The machine language for the JVM is called Java byte-code. But a different Java bytecode interpreter is needed for each type of computer. Once a computer has a Java bytecode interpreter, it can run any Java bytecode program. In other words, the same Java bytecode program can be run on any computer that has such an interpreter. This is one of the essential features of Java: the same compiled program can be run on many different types of computers as illustrated below. The combination of Java and Java bytecode together makes Java the platform-independent, secure, and network- compatible while allowing programming in a modern high-level object-oriented language.

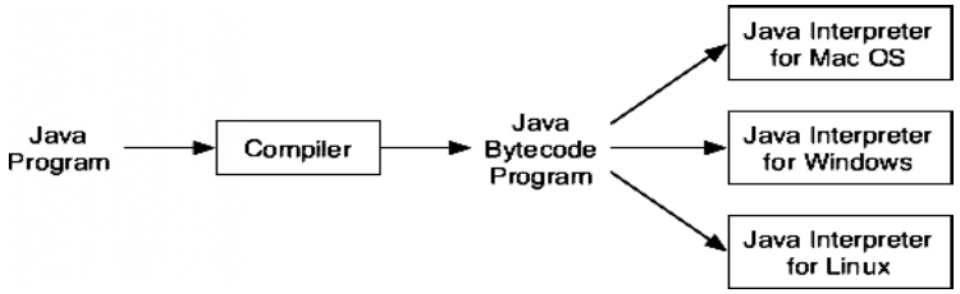


Fig 1.1 Byte Code Generation and running in different OS

Java is:

- Object Oriented; Platform independent
- Simple; Secure
- Architectural-neutral; Portable
- Robust; Multi-threaded
- Interpreted; High Performance
- Distributed; Dynamic

Lab No.: 1 Date:

Simple Java Programs using Control Structures

Objectives:

In this lab, student will be able to:

1. Write Java programs
2. Compile and interpret Java programs
3. Debug Java programs

Solved exercise

1. A Java program Sample.Javato display **Hello Java**

```
class Sample{
        public static void main(String args[ ]){
            System.out.println("Hello Java");
        }
    }
```

Output

Class Declaration

- **class** Sample: It declares an object oriented construct called a class. Java is a pure object oriented language requires variable declarations and method definitions be placed inside the class. The **class** keyword defines a new class. Sample is an identifier or name of the class. Every class definition in Java begins with an opening brace({) and ends with a matching closing brace(}).
- **public static void** main (String args[]) : This defines a method named main. Every Java application program must include the main() method. This is the starting point for the interpreter to begin the execution of the program. There may be any number of classes in a Java program. The name of the class containing main( ) method must

be the file name with extension **java**. String is the built-in class available in java. lang package, which gets imported automatically to all the java programs.

- public, static and void: These are the three keywords of Java having the following meanings:

  public: It is an access specifier that declares the main method as unprotected and therefore makes it accessible to all other classes

  static: It declares this method as one that belongs to the entire class and not a part of any objects of the class. The main must always be declared as static since the interpreter uses this method before any objects are created.

  void: It indicates that the main( ) method does not return anything.

- System.out.println ("Hello Java");This is similar to the cout<<"Hello Java"<<endl; construct of C++. The println( )method is a member of the out object which is a static data member of System class. This line prints the string Hello Java to the screen .The println( ) always appends a newline character to the end of the string (as opposite to the use of print( )). This means that every subsequent output will start on a new line. Every Java statement must end with a semicolon.
- To Edit, Save, Compile and Interpret/Run a Java Program Sample.java:
  - ✓ Use an editor to edit the code
  - ✓ Save the file as Sample.Java
  - ✓ Compile Sample.Java in the terminal using the command
    javac Sample.java
    This command creates Sample.classfile representing bytecode
  - ✓ Interpret Sample.class in the terminal using the command
    java Sample

For taking user input we use **Scanner Class**. To use this we **import java.util.Scanner** Class. First we create object of Scanner Class and use any of the methods in the table below to accept a specific data type value.

```
Scanner sc=new Scanner(System.in);
int a = sc.nextInt();       // to accept integer value
String s=sc.nextLine();     // to accept a line input ( multiple strings )
```

Commonly used public methods of Scanner class

Sl. No.	Method	Description
1.	String next()	Returns the next token from the scanner.
2.	String nextLine()	Moves the scanner position to the next line and returns the value as a string.
3.	byte nextByte()	Scans the next token as a byte.
4.	short nextShort()	Scans the next token as a short value.
5.	int nextInt()	Scans the next token as an int value.
6.	long nextLong()	Scans the next token as a long value.
7.	float nextFloat()	Scans the next token as a float value.
8.	double nextDouble()	Scans the next token as a double value.

Lab exercises

Write and execute Java programs to do the following:

1. Create a method **max( )** that has three integer parameters **x**, **y**, and **z,** and it returns the largest of the three. Do it two ways: once using an **if-else-if** ladder and once using nested **if** statements.
2. a. Write a method **reverse** to accept one integer parameter and to return the reversed digits of accepted number
   b. Using this method, check whether the inputted number is palindrome or not.
3. a. Write a method **fact** to accept one integer parameter and to find the factorial of a given number.

   b. Using **fact** method, compute ${}^{N}C_{R}$ in the main method.

Additional exercises

1. a. Write a method **isPrime** to accept one integer parameter and to check whether that parameter is prime or not.
   b. Using this method, generate first N prime numbers in the main method.
2. Write a method **findSum** to find the sum of digits of a number.

Lab No.: 2 Date:

1D and 2D Arrays

Objectives:

In this lab, student will be able to:

1. Create 1D and 2D arrays and their alternate syntax
2. Use the length array member
3. Use the for-each style for loop to iterate over arrays

Introduction to Arrays

An array is a group of like-typed variables that are referred to by a common name. Arrays of any type can be created and may have one or more dimensions. A specific element in an array is accessed by its index. Arrays offer a convenient means of grouping related information. The array elements are placed in a contiguous memory location.

1. Dimensional Array

A one-dimensional array is a list of like-typed variables. A particular value in an array is accesses by writing an integer number called index number or subscript in square brackets after the array name. The least value that an index can take in array is 0.

Array Declaration:

```
datatype[] arr; (or)
datatype arr[];
```

Instantiation of an Array in Java

```
arr= new dataType [size];
```

- ✓ size specifies how many elements the array has to contain . It can be a variable or a constant
- ✓ where dataType is a primitive (like int, float, char...) or user defined data type

- ✓ arr is a valid identifier
- ✓ The square brackets ([ ]) after the "dataType" indicate that arr is going to be an 1D array

The following two declarations are equivalent:

```
int al[] = new int[3];
int[] a1 = new int[3];
```

2. Dimensional Array

In Java, 2D array is actually an array of 1D arrays.

Array Declaration:

```
dataType[][] variable name = new dataType[rows][cols];
```

For example, the following line declares a two dimensional array variable called two D with 4 rows and 5 columns.

```
int twoD[][] = new int[4][5];
```

In Java it is possible to create 2D arrays with different number of elements in each row. For example, the following line declares a two dimensional array variable called two D with 3 rows and first row with1column, second with 2 columns, and third with 3 columns.

```
int twoD[][] = new int[3][];
twoD[0] = new int[1];
twoD[1] = new int[2];
twoD[2] = new int[3];
```

The following declarations are equivalent:

```
char twod1[][] = new char[3][4];
char[][] twod2 = new char[3][4];
```

Size of the array can be obtained by the length property of the array.

twod1.length will give the number of rows in 2D array.

twod1[0].length will give the number of columns in the first row of a 2D array.

The number of elements in a 1D array can be given by ArrayName.length

Solved exercises

1. Program to read elements into a 1D array and print it:

```
class Testarray{
public static void main(String args[]){
int a[]=new int[3];                    //declaration and instantiation
a[0]=10;                               //assignment
a[1]=20;
a[2]=70;
                                       //printing the array
for(int i=0;i<a.length;i++)            //length is the property of array
        System.out.println(a[i]);
 }
}
```

Output:

```
10
20
70
```

2. Program to read array size and array elements using scanner class and display it

```
import java.util.Scanner;
class ArrayReadDisp {
public static void main(String []args) {
        int n, c;
        Scanner in = new Scanner(System.in);
        System.out.println("Input number of integers");
        n = in.nextInt();              // to read size of the array
        int array[] = new int[n]; // allocate memory for the array dynamically

        System.out.println("Enter " + n + " integers"); // to read array elements
        for (c = 0; c < n; c++)
        array[c] = in.nextInt( );
```

```
        // to display array elements
        System.out.println("The array is");
      for (c = 0; c < n; c++)
      System.out.println(array[c]);
  }}
```

Output

```
Input number of integers
3
Enter 3 integers
2
4
6
The array is
2
4
6
```

Lab exercises

Write and execute Java programs to do the following:

1. Write a program that creates an array of integers and then uses a for loop to reverse the order of the elements in the array.
2. Write a program that creates an integer array of length N, fills the array with the sequence of values (from 1 to N) using a for loop, and then loops through the array printing out the values. Use a for-each style for loop to print out the values.
3. Print all the prime numbers in a given 1D array.
4. Find the addition of two matrices and display the resultant matrix.

Additional exercises

1. Search an element in a 1D array using linear search

2. Write a program that creates a "triangular" two-dimensional array **A** of 10 rows. The first row has length 1, the second row has length 2, the third row has length 3, and so on. Then initialize the array using nested **for** loops so that the value of **A[i][j]** is **i+j**. Finally, print out the array in a nice triangular form

Classes and Objects

Objectives:

In this lab student will be able to:

1. Acquire the fundamentals of the class
2. Implement the creation of objects
3. Analyze how reference variables are assigned
4. Implement new, garbage collection and this

Introduction:

The class defines a new datatype that can be used to create objects of that type. Thus, a class is a template for an object, and an object is an instance of a class.

Defining a class:

```
class classname{
        type instance-variable1;
        type instance-variable2;
        // ...
        type instance-variableN;
    type methodname1(parameter-list) {
                // body of method
    }
    type methodname2(parameter-list) {
                // body of method
    }
    // ...
    Type methodnameN(parameter-list) {
                // body of method
    }
}
```

Object creation:

It is a two-step process.

1. Declare a variable of the class type. This variable does not define an object. Instead, it is simply a variable that can refer to an object.
2. Get a physical copy of the object and assign it to that variable using the new operator. The new operator dynamically allocates (that is, allocates at run time) memory for an object and returns a reference to it.

Consider a class Box whose object mybox is created as follows

Box mybox = new Box();

This statement combines the two steps just described. It can be rewritten like this to show each step more clearly:

```
Box mybox;          // declare reference to object
mybox = new Box();  // allocate a Box object
```

The steps are illustrated below:

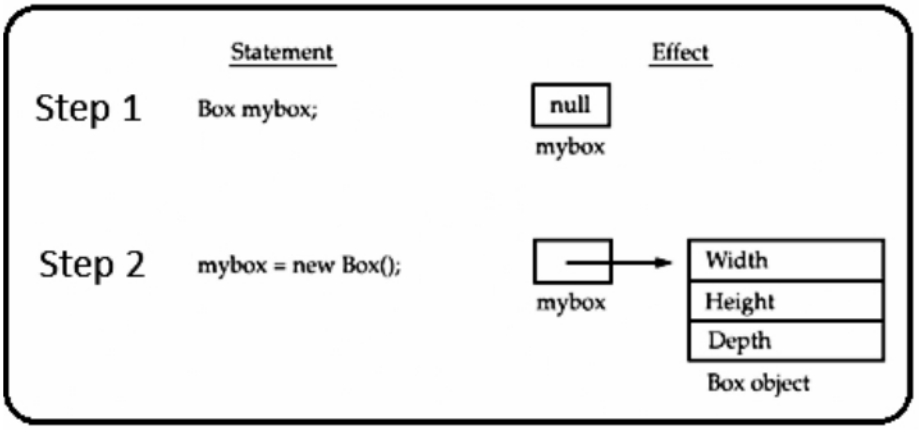


Fig 3.1 Steps to declare reference to object and allocate a box object

Solved exercise

1. Java program to find the volume of a box using classes: class

```
Box {
    double  width;
    double height;
```

```
    double depth;
// compute and return volume
    double volume() {
    return width * height * depth;
    }
// sets dimensions of box
    void setDim(double w, double h, double d) {
    width = w;
    height = h;
    depth = d;
}
}
class BoxDemo {
    public static void main(String args[]) {
            Box mybox1 = new Box(); //declare reference to an object
            Box mybox2 = new Box(); //allocate a Box object
            doublevol;
                                    // initialize each box
            mybox1.setDim(10, 20, 15);
            mybox2.setDim(3, 6, 9);
                                    // get volume of first box
            vol = mybox1.volume();

            System.out.println("Volume is " + vol);
            vol = mybox2.volume(); // get volume of second box
            System.out.println("Volume is " + vol);
            }
    }
```

Output

Volume is 3000.0

Volume is 162.0

Lab exercises

1) Define a Class STUDENT having following Members: sname, marks_array, total, avg and provide the following methods:

i) assign(): to assign initial values to the STUDENT object
ii) display(): to display the STUDENT object
iii) compute(): to Compute Total, Average marks

Write a Java program Illustrating Class Declarations, Definition, and Accessing Class Members to test the class defined.

2) Define a class EMPLOYEE having following members: Ename, Eid, Basic, DA, Gross_Sal, Net_Sal and following methods:
   i) read(): to read N employee details
   ii) display(): to display employee details
   iii) compute_net_sal(): to compute net salary

Write a Java program to read data of N employee and compute and display net salary of each employee Note: (DA = 52% of Basic, gross_Sal = Basic + DA; IT = 30% of the gross salary)

3. Define a class Mixer to merge two sorted integer arrays in ascending order with the following instance variables and methods:

   instance variables:

   int arr[] //to store the elements of an array

   Methods:

   void accept() // to accept the elements of the array in ascending order without any duplicates

   Mixer mix(Mixer A) // to merge the current object array elements with the parameterized array elements and return the resultant object

   void display() // to display the elements of the array

   Define the main() method to test the class.

4. Create a class called Stack for storing integers. The instance variables are:
   - An integer array

•An integer for storing the top of stack (tos)

Include methods for initializing tos, pushing an element to the stack and for popping an element from the stack. The push()method should also check for "stack overflow" and pop() should also check for "stack underflow". Use a display( ) method to display the contents of stack.

Additional Exercises

1. The International Standard Book Number (ISBN) is a unique numeric book identifier which is printed on every book. The ISBN is based upon a 10-digit code. The ISBN is legal if:

   1xdigit1 + 2xdigit2 + 3xdigit3 + 4xdigit4 + 5xdigit5 + 6xdigit6 + 7xdigit7 + 8xdigit8 + 9xdigit9 + 10xdigit10 is divisible by 11.

   example: For an ISBN 1401601499:

   Sum=1×1 + 2×4 + 3×0 + 4×1 + 5×6 + 6×0 + 7×1 + 8×4 + 9×9 + 10×9 = 253 which is divisible by 11.

   Write a program to implement the following methods:

   inputISBN( ) to read the ISBN code as a 10-digit integer.

   checkISBN() to perform the following check operations :

   i) If the ISBN is not a 10-digit integer, output the message "ISBN should be a 10 digit number" and terminate the program.

   ii) If the number is 10-digit, extract the digits of the number and compute the sum as explained above. If the sum is divisible by 11, output the message, "Legal ISBN"; otherwise output the message, "Illegal ISBN"

2. Create a Die class with one integer instance variable called sideUp. Give it a getSideUp() method that returns the values of sideUp and a void roll() method that changes sideUpto a random value from 1 to 6.Then create a DieDemo class with a method that creates two Die objects, rolls them, and prints the sum of the two sides up.

Lab No.: 4 Date:

Constructors and Static Members

Objectives:

In this lab student will be able to:

1. Utilize various types of constructors
2. Overloading constructors
3. Analyzing static member

Introduction:

Constructor:

A constructor initializes an object when it is created. It has the same name as that of class and has no return type (not even void). Constructors are utilized to give initial values to the instance variables defined by the class, or to perform any other start up procedures required to create a fully formed object. In general there are two different types of constructors:

1. Default
2. Parameterized

The following example shows how they are created and called.

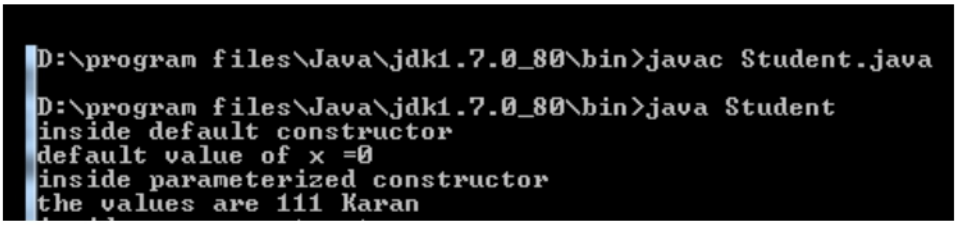
Solved exercise

1. Program to illustrate default and parameterized constructors.

```
import java.util.*;
class Student{
int id;
   String name;
Student(){          //  zero  argument  constructor
            System.out.println("inside  default  constructor");
            System.out.println("the default values are "+id+" "+name);
```

```
}
Student(int i,String n){          // Parameterized constructor
        id = i;
        name = n;
        System.out.println("inside parameterized constructor");
        System.out.println("the values are "+id+" "+name);
   }
public static void main(String args[]){
          Student s1 = new Student(); //calling default constructor
          Student s2 = new Student(111,"Karan"); //calling parameterized constructor
       }
}
```

Output



Static variables and methods

Normally a class member must be accessed through an object of its class, but it is possible to create a member that can be used without reference to a specific instance. To create this type of member, precede its declaration with the keyword static. When a member is declared static, it can be accessed before any objects of its class are created without reference to any object. Both methods and variables can be declared as static.

Following example illustrates the usage of static variable and static method:

```
class StaticMeth{
        static int val=1024;
        static int valDiv2(){
                return val/2;
        }
}
class SDemo2{
```

```
    public static void main(String[] args){
        System.out.println("val is "+StaticMeth.val);
        System.out.println("StaticMeth.valDiv2() : "+StaticMeth.valDiv2());
        StaticMeth.val=4;
        System.out.println("val is "+StaticMeth.val);
        System.out.println("StaticMeth.valDiv2() : "+StaticMeth.valDiv2());
    }
}
```

Output

```
val is 1024
StaticMeth.valDiv2() : 512
val is 4
StaticMeth.valDiv2() : 2
```

Since the methodvalDiv2() is declared static, it can be called without any instance of its class StaticMeth being created, but by using class name.

Lab Exercises:

1. Consider the already defined STUDENT class. Provide a default constructor and parameterized constructor to this class. Also provide a display method. Illustrate all the constructors as well as the display method by defining Complex objects.
2. Consider the already defined EMPLOYEE class. Provide a default constructor, and parameterized constructor. Also provide a display method. Illustrate all the constructors as well as the display method by defining Time objects.
3. Define a class to represent a **Bank account**. Include the following members.
   Data members:
   1. Name of the depositor
   2. Account number.
   3. Type of account.
   4. Balance amount in the account.
   5. Rate of interest (static data)

   Provide a default constructor and parameterized constructor to this class. Also provide Methods:
   1. To deposit amount.

2. To withdraw amount after checking for minimum balance.
3. To display all the details of an account holder.
4. Display rate of interest (a static method)

Illustrate all the constructors as well as all the methods by defining objects.

4. Create a class called Counter that contains a static data member to count the number of Counter objects being created. Also define a static member function called showCount() which displays the number of objects created at any given point of time. Illustrate this.

Additional Exercises

1. Define a class **IntArr** which hosts an array of integers. Provide the following methods:
   1. A ***default constructor.***
   2. A ***parameterized constructor*** which initializes the array of the object.
   3. A method called ***display*** to display the array contents.
   4. A method called ***search*** to search for an element in the array.
   5. A method called ***compare*** which compares 2 **IntArr** objects for equality.

2. Define a class called Customer that holds private fields for a customer ID number, name and credit limit. Include appropriate constructors to initialize the instance variables of the Customer Class. Write a main() method that declares an array of 5 Customer objects. Prompt the user for values for each Customer, and display all 5 Customer objects.

Lab No.: 5 Date:

Strings

Objectives:

In this lab, student will be able to:

1. Know different ways of creating String objects and constants
2. Learn and use string handling methods
3. Know the difference between String and StringBuffer classes

Introduction to Strings:

- String is probably the most commonly used class in Java's class library. Every string created is actually an object of type String. Even string constants are actually String objects. For example, in the statement System.out.println ("This is a String, too"); the string "This is a String, too" is a String constant.
- The objects of type String are immutable; once a String object is created, its contents cannot be altered. To change a string, create a new one that contains the modifications or Java defines a peer class of String, called StringBuffer, which allows strings to be altered, so all of the normal string manipulations are still available in Java.
- Strings can be constructed in a variety of ways. The easiest is to use a statement like this:

```
String myString = "this is a test";
System.out.println(myString);
```

- In Java, + operator is used to concatenate two strings.
  For example, this statement String myString = "I" + "like" + "Java.";
  results in myString containing "I like Java."
- Some of the important methods of String class.

```
boolean equals(String object) // To test two strings for equality
int length( )                 // obtain the length of a string
charcharAt(int index)         // To get the character at a specified index
                                 within a string
```

- Like array of any other type of objects, array of Strings is also possible.

Solved exercises

1. Program to demonstrate Strings.

```
class StringDemo {
    public static void main(String args[]) {
            String strOb1 = "First String";
            String strOb2 = "Second String";
            String strOb3 = strOb1 + " and " + strOb2;// using '+' for concatenation
            System.out.println(strOb1);
            System.out.println(strOb2);
            System.out.println(strOb3);
}
}
```

The output produced by this program is shown here:

```
First String
Second String
First String and Second String
```

2. Program to demonstrate array of Strings.

```
class StringArray {
    public static void main(String args[]) {
            String str[] = { "one", "two", "three" };
            for(inti=0; i<str.length; i++)
                    System.out.println("str[" + i + "]: " +str[i]);
}
}
```

Here is the output from this program:

```
str[0]: one
str[1]:   two
str[2]: three
```

Lab Exercise:

1. Add the following string processing methods to the class Employee:
   a) formatEmployeeName(): A method that formats the employee's name by capitalizing the first letter of each word and converting the remaining letters to lowercase. For example, if the employee's name is "JOHN DOE", this method would transform it to "John Doe".
   b) generateEmail(): A method that generates an email address for the employee based on their name. For example, if the employee's name is "John Doe", this method would generate an email address like "jdoe@example.com".
2. Add the following string processing methods to the lass STUDENT:
   a) extractInitials(): A method that extracts the initials from the student's name. For example, if the student's name is "John Doe", this method would return "JD".
   b) removeWhitespace(): A method that removes any whitespace characters from the student's name. For example, if the student's name is "John Doe", this method would transform it to "JohnDoe".
   c) List all the student names containing a particular sub string.
   d) Sort the students alphabetically

3 Design a class which represents a student. Every student record is made up of the following fields.
   a. Registration number (int)
   b. Full Name (String)
   c. Date of joining (Gregorian calendar)
   d. Semester (short)
   e. GPA (float)
   f.CGPA (float)

Whenever a student joins he will be given a new registration number. Registration number is calculated as follows. If year of joining is 2012 and he is the 80^{th} student to join then his registration number will be 1280.

Write member functions to do the following.

a) Provide parameterized constructors to this class
b) Write display method which displays the record. Test the class by writing suitable main method.
c) Create an array of student record to store minimum of 5 records in it. Input the records and display them.

4 Perform the following operations by adding member functions to the program implemented in the above question

a) Sort the student records with respect to semester and CGPA.
b) Sort the student record with respect to name.

5 Add member functions to the above code that perform the following operations

a) List all the students whose name starts with a particular character.
b) List all the student names containing a particular sub string.
c) Change the full name in the object to name with just initials and family name. For example, Prakash Kalingrao Aithal must be changed to P. K. Aithal and store it in the object. Display modified objects.

6 Write and execute a Java program to convert strings containing numbers into comma-punctuated numbers, with a comma every third digit from the right.

e.g., Input String : "1234567"

Output String : "1,234,567"

Additional exercises:

1. Write and execute a Java program to pull out all occurrences of a given sub-string present in the main string.
2. Write and execute a Java program to count number of occurrences of a particular string in another string.

Lab No.: 6 Date:

Inheritance

Objectives:

In this lab student will be able to:

1. Implement Inheritance basics
2. Use super keyword to access super class members and constructors
3. Implement dynamic polymorphism by overriding methods
4. Differentiate between abstract classes and concrete classes

Introduction:

Inheritance

Java supports inheritance by allowing one class to incorporate another class into its declaration. This is done using the extends keyword. Thus the subclass adds (extends) to the superclass.

Solved exercise

1. Program creates a superclass called TwoDShape which stores the width and height of a two dimensional object, and a subclass called Triangle extends from it.

```
classTwoDShape{
    private double width,height;
    double getWidth() {   return   width;}
    double getHeight(){   return height; }
    void setWidth(double w){     width=w;       }
    void setHeight(double h){     height=h;       }
    void show(){System.out.println("width and height are "+width+" and
    "+height);}
}
class Triangle extends TwoDShape{
```

```
    String style;
    double area(){    return getWidth()*getHeight()/2;     }
    Triangle(String s, double w, double h){ setWidth(w); setHeight(h); style = s; }
    //show () method here overrides the one in TwoDShape class which is the base
    class
    void show() { super.show();System.out.println("Triangle is" + style ); }
  }
  class inheritanceEx{
      public static void main(String[] args){
            Triangle t1=new Triangle("outlined",8.0,12.0);
            Triangle  t2=new  Triangle("filled",  4.0,4.0);
            t1.show();
            t2.show();
            }
}
```

Output

```
D:\program files\Java\jdk1.7.0_80\bin>javac inheritanceEx1.java

D:\program files\Java\jdk1.7.0_80\bin>java inheritanceEx1
width and height are 8.0 and 12.0
Triangle isoutlined
width and height are 4.0 and 4.0
Triangle isfilled
```

Lab Exercises

1. To the already defined STUDENT class, add two subclasses ScienceStudent and ArtsStudent. Add a private data member practicalMarks (int): to the ScienceStudent class to represent the marks obtained by the student in the practical subject. The ScienceStudent class should override the compute() method to include the practical marks in the total marks and average marks calculation. The ScienceStudent class should provide a method displayPracticalMarks() to display the practical marks obtained by the science student. Add the private data member electiveSubject (String): to the ArtsStudent class to represent the elective subject chosen by the arts student. Also add appropriate constructors to the subclasses. In the Main class, create objects of STUDENT, ScienceStudent, and ArtsStudent, and demonstrate the

keyword 'super' and other functionalities of the classes by assigning values, computing marks, and displaying the information of the students. Also demonstrate dynamic polymorphism.

2. To the already defined EMPLOYEE class, add two subclasses FullTimeEmp and PartTimeEmp. Add the following data Members and Member Functions to the PartTimeEmp class. hoursWorked (int): Represents the number of hours worked by the part-time employee. hourlyRate (double, static and final): Represents the hourly rate at which the part-time employee is paid. The calculateSalary(): Overrides the calculateSalary() method of the base class to calculate the salary of the part-time employee based on the hours worked and hourly rate. The displayEmployeeDetails(): Overrides the displayEmployeeDetails() method of the base class to display the part-time employee's details, including the hours worked and hourly rate.

The FullTimeEmployee subclass includes the data members bonus and deductions as additional data members and are of type double, and overrides the calculateSalary() and displayEmployeeDetails() methods to incorporate these factors. Define Main class to check the functionality of all the classes.

3. **Person** class with private instance variables for the person's name and birth date. Add appropriate accessor methods for these variables. Then create a subclass **College Graduate** with private instance variables for the student's GPA and year of graduation and appropriate accessors for these variables. Include appropriate constructors for your classes. Then create a class with **main()** method that demonstrates your classes.

4. Create a Building class and two subclasses, House and School. The Building class contains fields for square footage and stories. The House class contains additional fields for number of bedrooms and baths. The School class contains additional fields for number of classrooms and grade level (for example, elementary or junior high). All the classes contain appropriate get and set methods. Create a main method that declares objects of each type.

5. Create an abstract class Figure with abstract method area and two integer dimensions. Create three more classes Rectangle, Triangle and Square which extend Figure and implement the area method. Show how the area can be computed dynamically during run time for Rectangle, Square and Triangle to achieve dynamic polymorphism. (Use

the reference of Figure class to call the three different area methods)

6. Design a base class called Student with the following 2 fields:- (i) Name (ii) Id. Derive 2 classes called Sports and Exam from the Student base class. Class Sports has a field called s_grade and class Exam has a field called e_grade which are integer fields. Derive a class called Results which inherit from Sports and Exam. This class has a character array or string field to represent the final result. Also it has a method called display which can be used to display the final result. Illustrate the usage of these classes in main.

Lab No.: 7 Date:

Packages and Interfaces

Objectives:

In this lab student will be able to:

1. Know the purpose and creation of packages
2. Know how to import packages and how packages affect access
3. Implement interface fundamentals
4. Implement interfaces and apply interface references
5. Use interface constants, extend interfaces.

Introduction to Packages:

While naming a class, the name chosen for a class is reasonably unique and not collides with class names chosen by other programmers. Packages are containers for classes that are used to keep the class name space compartmentalized. The package is both a naming and a visibility control mechanism. It is possible to define classes inside a package that are not accessible bytecode outside that package.

Simple example of java package

The **package keyword** is used to create a package in java.

```
                                    //save as Simple.java
    package mypack;
    public class Simple{
                public static void main(String args[]){
                        System.out.println("Welcome to package");
      }
    }
```

To compile java package

Give the command with the following format in the terminal:

javac -d directory javafilename

For example

javac -d . Simple.java

The -d switch specifies the destination where to put the generated class file. Any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. can be used. To keep the package within the same directory, use . (dot).

To run java package program: Use fully qualified name e.g. Java mypack.Simple to run the class.

Output:

Welcome to package

To access a package from another package: There are three ways to access the package from outside the package

1. import package.*;
2. import package.classname;
3. fully qualified name

1) Using packagename.*

If package.*is used, then all the classes and interfaces of this package will be accessible but not subpackages. The import keyword is used to make the classes and interface of another package accessible to the current package.

Example of package that import the packagename.*

```
//save by A.java
package    pack;
public class A{
        public void msg(){System.out.println("Hello");}
}
//save  by  B.java
package mypack;
import     pack.*;
class B{
public static void main(String args[]){
              A obj = new A();
             obj.msg();
  }
}
```

Output:

Hello

2) Using packagename.classname

If package.classname is imported, then only declared class of this package will be accessible.

Example of package by import package.classname

```
                //save by A.java
package   pack;
public class A{
        public void msg(){System.out.println("Hello");}
}
                //save by B.java
package mypack;
import    pack.A;
class B{
        public static void main(String args[]){
         A obj = new A();
        obj.msg();
  }
}
```

Output:

Hello

3) Using fully qualified name

If the fully qualified name is used then only declared class of this package will be accessible. Now there is no need to import. But the fully qualified name must be used every time while accessing the class or interface.

It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Example of package by import fully qualified name

```
//save by A.java
package   pack;
public class A{
        public void msg(){System.out.println("Hello");}}
```

```
package mypack;
class B{
//save by B.java
  public static void main(String args[]){
            pack.Aobj = new pack.A();//using fully qualified name
            obj.msg();
  }
}
```

Output:
Hello

Introduction to Interfaces:

An interface is a blueprint of a class. It has static constants and abstract methods only. There are mainly three reasons to use interface. They are given below.

i) It is used to achieve absolute abstraction.
ii) Interface can be used to achieve the functionality of multiple inheritances.
iii) It can be used to achieve loose coupling.

Solved exercise

1. Program to illustrate usage of interfaces

```
interface Printable{
        void print();
}
interface Showable{
        void show();
}
class A implements Printable,Showable{
        public  void  print(){System.out.println("Hello");}  public
        void show(){System.out.println("Welcome");}

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

```
        obj.print();
        obj.show();
        }
}
```

Output:

```
Hello
Welcome
```

Lab Exercises

1. Design a Building class as in Lab 6, program no. 2. Place the Building, House, and School classes in a package named com.course.structure. Create a main method that declares objects of each type and uses the package.

2. Define a class Maximum with the following overloaded static methods

   a. max (which finds maximum among three integers and returns the maximum integer)
   b. max (which finds maximum among three floating point numbers and returns the maximum among them)
   c. max (which finds the maximum in an array and returns it)
   d. max (which finds the maximum in a matrix and returns the result)

   Place this in a package called myPackages.p1. Write a main method to use the methods of Maximum class in package p1.

3. Design an interface called Series with the following methods
   i) Get Next (returns the next number in series)
   ii) reset(to restart the series)
   iii) set Start (to set the value from which the series should start)

   Design a class named **By Twos** that will implement the methods of the interface Series such that it generates a series of numbers, each two greater than the previous one. Also design a class which will include the main method for referencing the interface.

4. Design a class Student with the methods, get Number and put Number to read and display the Roll No. of each student and get Marks() and put Marks() to read and

display their marks. Create an interface called Sports with a method put Grade() that will display the grade obtained by a student in Sports. Design a class called Result that will implement the method put Grade() and generate the final result based on the grade in sports and the marks obtained from the superclass Student.

Additional Exercises

1. Create a class Phone{string brand, int memCapacity}, which contains an interface (Nested interface) Callable{makeAudioCall(string cellNum), makeVideoCall(string cellNum)}. Create subclasses BasicPhone and SmartPhone and implement the methods appropriately. Demonstrate the creation of both subclass objects by calling appropriate constructors which accepts values form the user. Using these objects call the methods of the interface.
2. Develop following methods
   i) IntSort (sort n integers in ascending order using Selection Sort)
   ii) BinSearch (performs Binary search for an element among a given list of integers)

   Place this in a package called **pIntegers**. Let this package be present in a folder called "myPackages", which is a folder in your present working directory (eg:- c\student\3rdsem\mypackages\pIntegers). Write a main method in package **pMain** to read an array of integers and display it. Using the methods IntSort and BinSearch from package **pIntegers** perform sort and search operations on the list of array elements.

Lab No.: 8 Date:

Exception Handling

Objectives:

In this lab, student will be able to:

1. Know the exception mechanism of Java
2. Know Java's built-in exceptions and to create custom exceptions

Introduction to Exceptions

A method catches an exception using a combination of the **try** and **catch** keywords. A try/catch block is placed around the code that might generate an exception.

The following code has an array declared with 2 elements. Then the code tries to access the 3rd element of the array which throws an exception.

```
public class ExcepTest{
public static void main(String args[]){
        int a[ ] = new int[2];
        try{
            System.out.println("Access element three :" + a[3]);
             }
          catch(ArrayIndexOutOfBoundsException e){
                System.out.println("Exception thrown :" + e);
}
        finally{
                a[0] = 6;
                System.out.println("First element value: " +a[0]);
                System.out.println("The finally statement is executed");
                } } }
```

Output:

```
Exception thrown :java.lang.ArrayIndexOutOfBoundsException: 3
First element value: 6
The finally statement is executed
```

The finally Keyword: The finally keyword is used to create a block of code that follows a try/catch block. A finally block of code always executes, whether or not an exception has occurred. Using a finally block allows to run any cleanup-type statements like closing of file.

The throws/throw Keywords: If a method does not handle a checked exception, the method must declare it using the **throws** keyword. The throws keyword appears at the end of a method's signature. The **throw** keyword can be used to throw an exception, either a newly instantiated one or an exception that is just caught

User defined Exceptions

Note the following while creating own exceptions:

- All exceptions must be a child of Throwable.
- To create checked exception that is automatically enforced by the Handle or Declare Rule, extend the Exception class.
- To create a runtime exception, extend the Runtime Exception class.

2. The following Insufficient Funds Exception class is a user-defined exception that extends the Exception class, making it a checked exception. An exception class is like any other class, containing useful fields and methods.

```
import java.io.*;
// File Name InsufficientFundsException.java
public class InsufficientFundsException extends Exception{
        private double amount;
        public InsufficientFundsException(double amount){
                this.amount = amount;
          }
        public double getAmount(){
                    return amount;
          }
}
```

To demonstrate using our user-defined exception, the following Checking Account class contains a withdraw() method that throws an Insufficient Funds Exception.

```
// File Name CheckingAccount.java
import java.io.*;
public class CheckingAccount{
```

```
        private double balance;
        privateint number;
        publicCheckingAccount(int number){
                this.number = number;
          }
        public void deposit(double amount){
                balance += amount;
          }
        public void withdraw(double amount) throws InsufficientFundsException {
                if(amount <= balance) {
                        balance -= amount;
                  }
              else {
                        double needs = amount - balance;
                        throw new InsufficientFundsException(needs);
                  }
          }
}
```

/*The following BankDemo program demonstrates invoking the deposit() and withdraw() methods of CheckingAccount.*/

```
// File Name BankDemo.java
public class BankDemo{
        public static void main(String [] args) {
          CheckingAccount c = new CheckingAccount(101);
          System.out.println("Depositing $500...");
          c.deposit(500.00);
          try {
              System.out.println("\nWithdrawing $100...");
              c.withdraw(100.00);
              System.out.println("\nWithdrawing $600...");
              c.withdraw(600.00);
             }
```

```
            catch(InsufficientFundsException e) {
                System.out.println("Sorry, but you are short $" + e.getAmount());
                e.printStackTrace();
            }
    }
}
```

Output

```
Depositing $500...
Withdrawing $100...
Withdrawing $600...
Sorry, but you are short $200.0
InsufficientFundsException
atCheckingAccount.withdraw(CheckingAccount.java:25)
atBankDemo.main(BankDemo.java:13)
```

Lab Exercises

1. Design a stack class. Provide your own stack exceptions namely Push Exception and Pop Exception, which throw exceptions when the stack is full and when the stack is empty respectively. Show the usage of these exceptions in handling a stack object in the main.

2. Define a class CurrentDate with data members day, month and year. Define a method createDate() to create date object by reading values from keyboard. Throw a user defined exception by name InvalidDayException if the day is invalid and InvalidMonthException if month is found invalid and display current date if the date is valid. Write a test program to illustrate the functionality.

3. Design a Student class with appropriate data members as in Lab 5. Provide your own exceptions namely Seats Filled Exception, which is thrown when Student registration number is >XX25 (where XX is last two digits of the year of joining) Show the usage of this exception handling using Student objects in the main. (Note: Registration number must be a unique number)

Lab No.: 9 Date:

Multithreading

Objectives:

In this lab, student will be able to:

1. Write and execute multithreaded programs
2. Implement how java achieves concurrency through APIs
3. Learn language level support for achieving synchronization
4. Know and execute programs that use inter-thread communication

Introduction to Multithreading:

Java makes concurrency available through APIs. Java programs can have multiple threads of execution, where each thread has its own method-call stack and program counter, allowing it to execute concurrently with other threads while sharing with them application-wide resources such as memory. This capability is called multithreading.

A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a *thread,* and each thread defines a separate path of execution.

Creating a Thread:

Java defines two ways in which this can be accomplished:

- Implement the Runnable interface
- Extend the Thread class

Solved exercise

1. Program to demonstrate the creation of thread by implementing Runnable interface:

```
class NewThread implements Runnable {
    Thread t;
    NewThread() {
```

```
        // Create a new, second thread
           t = new Thread(this, "Demo Thread");
        System.out.println("Child thread: " + t);
        t.start();          // Start the thread, this will call run method
      }
    public void run() {          // This is the entry point for the second thread.
        try {
           for(int i = 5; i> 0; i--) {
                       System.out.println("Child Thread: " + i);
                       Thread.sleep(500);
                  }
            }
        catch       (InterruptedException       e)       {
                System.out.println("Child interrupted.");
                   }
        System.out.println("Exiting child thread.");
    }
}
Class ThreadDemo {
      public static void main(String args[]) {
           new NewThread(); // create a new thread
           try {
                for(int i = 5; i> 0; i--) {
                        System.out.println("Main Thread: " + i);
                        Thread.sleep(1000);
                    }
            }
        catch          (InterruptedException          e)          {
                System.out.println("Main thread interrupted.");
             }
        System.out.println("Main thread exiting.");
  }
}
```

The output may vary based on processor speed and task load

Output:

Child thread: Thread[Demo Thread,5,main]
Main Thread: 5
Child Thread: 5
Child Thread: 4
Main Thread: 4
Child Thread: 3
Child Thread: 2
Main Thread: 3
Child Thread: 1
Exiting child thread.
Main Thread: 2
Main Thread: 1
Main thread exiting.

2. Program to demonstrate the creation of thread by extending Thread class:

```
class NewThread extends Thread {
        NewThread() {
                        // Create a new, second thread
            super("Demo                    Thread");
            System.out.println("Child thread: " + this);
            start();        // Start the thread, this will call run method
          }
                        // This is the entry point for the second thread.
        public void run() {
            try {
                  for(int    i    =    5;    i>    0;    i--)    {
                     System.out.println("Child Thread: " + i);
                     Thread.sleep(500);
                   }
             }
```

```
        catch (InterruptedException e) {
            System.out.println("Child interrupted.");
        }
        System.out.println("Exiting child thread.");
    }
}
Class Extends Thread {
    public static void main(String args[]) {
        new NewThread();    // create a new thread

        try {
        for(int i = 5; i> 0; i--) {
        System.out.println("Main Thread: " + i);
        Thread.sleep(1000);
            }
        }
         catch (InterruptedException e) {
        System.out.println("Main thread interrupted.");
            }
        System.out.println("Main thread exiting.");
    }
}
```

Output

```
Child thread: Thread[Demo Thread,5,main]
Main Thread: 5
Child Thread: 5
Child Thread: 4
Child Thread: 3
Main Thread: 4
Child Thread: 2
Child Thread: 1
Main Thread: 3
Exiting child thread.
Main Thread: 2
Main Thread: 1
Main thread exiting.
```

3. Create multiple threads.

```
class NewThread implements Runnable {
    String name;                        // name of thread
    Thread t;
    NewThread(String threadname) {
        name = threadname;
        t = new Thread(this, name);
        System.out.println("New thread: " + t);
        t.start( );                     // Start the thread
    }
    public void run() {                 // This is the entry point for thread.
        try {
            for(int i = 5; i> 0; i--) {
                System.out.println(name + ": " + i);
                Thread.sleep(1000);
                    }
                }
            catch (InterruptedException e) {
                System.out.println(name + "Interrupted");
                    }
        System.out.println(name + " exiting.");
        }
}
class MultiThreadDemo {
        public static void main(String args[ ]) {
            new NewThread("One");  // start threads new
            NewThread("Two");
            new NewThread("Three");
            try {
            Thread.sleep(10000);        // wait for other threads to end
                }
            catch(InterruptedException e) {
                System.out.println("Main thread Interrupted");
```

```
        }
        System.out.println("Main thread exiting.");

        }
}
```

Output:

Lab exercises

1. Create a class by extending Thread Class to print a multiplication table of a number supplied as parameter. Create another class Tables which will instantiate two objects of the above class to print multiplication table of 5 and 7.

2. Write and execute a java program to create and initialize a matrix of integers. Create *n* threads(by implementing Runnable interface) where *n* is equal to the number of rows in the matrix. Each of these threads should compute a distinct row sum. The main thread computes the complete sum by looking into the partial sums given by the threads.

3. Write and execute a java program to implement a producer and consumer problem using Inter-thread communication.

Additional exercise:

1. Write and execute a java program to create five threads, the first thread checking the

uniqueness of matrix elements, the second calculating row sum, the third calculating the column sum, the fourth calculating principal diagonal sum, the fifth calculating the secondary diagonal sum of a given matrix. The main thread reads a square matrix from keyboard and will display whether the given matrix is magic square or not by obtaining the required data from sub threads.

2. Create a Counter class with a private count instance variable and two methods. The first method: synchronized void increment( ) tries to increment count by 1. If count is already at its maximum of 3, then it waits until count is less than 3 before incrementing it. The other method: synchronized void decrement( ) tries to decrement count by 1. If count is already at its minimum of 0, then it waits until count is greater than 0 before decrementing it. Every time either method has to wait, it displays a statement saying why it is waiting. Also, every time an increment or decrement occurs, the counter displays a statement that says what occurred and shows count's new value.

   A. Create one thread class whose run( ) method calls the Counter's increment( ) method 20 times. In between each call, it sleeps for a random amount of time between 0 and 500 milliseconds.

   B. Create one thread class whose run( ) method calls the Counter's decrement( ) method 20 times. In between each call, it sleeps for a random amount of time between 0 and 500 milliseconds.

   C. Write a Counter User class with a main( ) method that creates one Counter and the two threads and starts the threads running.

***Note**: Instead of creating two thread classes, you are free to create just one thread class that either increments or decrements the counter, depending on a parameter passed through the thread class's constructor.*

Generics

Objectives:

In this lab, student will be able to:

1. Outline the benefits of generics
2. Create generic classes and methods

Introduction to Generics:

Generics are a powerful extension to Java because they streamline the creation of type-safe, reusable code. The term *generics* means *parameterized types*. Parameterized types are important because they enable the programmer to create classes, interfaces, and methods in which the type of data upon which they operate is specified as a parameter. Using generics, it is possible to create a single class, for example, that automatically works with different types of data. A class, interface, or method that operates on a parameterized type is called *generic,* as in *generic class* or *generic method*.

Solved exercise:

1. Program defines two classes, the generic class Gen, and the second is GenDemo, which uses Gen. Here, T is a type parameter that will be replaced by a real type when an object of type Gen is created.

```
class Gen<T> {
        T ob;             // declare an object of type T
        // Pass the constructor a reference to an object of type T.
        Gen(T o) {
                ob = o;
           }
     // Return ob.
        T getob() {
                returnob;
           }
      // Show type of T.
```

```
void showType() {
      System.out.println("Type of T is " +ob.getClass().getName());
            }
}
// Demonstrate the generic class.
Class GenDemo {
          public static void main(String args[]) {
            // Create a Gen reference for Integers.
                  Gen<Integer>iOb;
             // Create a Gen<Integer> object and assign its
            // reference to iOb. Notice the use of autoboxing
           // to encapsulate the value 88 within an Integer object.
                  iOb = new Gen<Integer>(88);
           // Show the type of data used by iOb.
                  iOb.showType();
           // Get the value in iOb. Notice that no cast is needed.
                  int v = iOb.getob(); System.out.println("value:
                  " + v); System.out.println();
            // Create a Gen object for Strings.
                  Gen<String>strOb = new Gen<String>("Generics Test");
             // Show the type of data used by strOb.
                  strOb.showType();
            // Get the value of strOb. Again, notice that no cast is needed.
                  String    str    =    strOb.getob();
                  System.out.println("value: " + str);
              }
}
```

Output:

```
Type of T is java.lang.Integer
value: 88
Type of T is java.lang.String
value: Generics Test
```

Here, T is the name of a type parameter. This name is used as a placeholder for the actual type that will be passed to Gen when an object is created. The GenDemo class demonstrates the generic Gen class. It first creates a version of Gen for integers, as shown here:

Gen<Integer>iOb;

The type Integer is specified within the angle brackets after Gen. In this case, Integer is a type argument that is passed to Gen's type parameter, T. This effectively creates a version of Gen in which all references to T are translated into references to Integer. Thus, for this declaration, ob is of type Integer, and the return type of getob( ) is of type Integer.

The next line assigns to **iOb**a reference to an instance of an **Integer** version of the **Gen**class:

iOb = new Gen<Integer>(88);

Generics Work Only with Objects. Therefore, the following declaration is illegal:

Gen<int>strOb = new Gen<int>(53); // Error, can't use primitive type

Lab exercises:

1. Write a generic method to exchange the positions of two different elements in an array.
2. Define a simple generic stack class and show the use of the generic class for two different class types Student and Employee class objects.
3. Write a program to demonstrate the use of wildcard arguments for question no. 2.

Additional exercises:

1. Write a generic method that can print array of different type using a single Generic method:
2. Write a generic method to return the largest of three Comparable objects.

JavaFX and Event Handling -Part I

Objectives:

Objectives:

In this lab, student will be able to:

1. Outline importance of light weight components and pluggable look-and-feel
2. Create, compile and run JavaFX applications
3. Know the fundamentals of event handling and role of layout managers

JavaFX:

JavaFX components are lightweight and events are handled in an easy-to-manage, straightforward manner. The JavaFX framework is contained in packages that begin with the javafx prefix. The packages we will use in our code are : javafx.application, javafx.stage, javafx.scene, and javafx.scene.layout.

The Stage and Scene Classes:

Stage is a top-level container. All JavaFX applications automatically have access to one Stage, called the primary stage. The primary stage is supplied by the run-time system when a JavaFX application is started.

Scene is a container for the items that comprise the scene. These can consist of controls, such as push buttons and check boxes, text, and graphics. To create a scene, you will add those elements to an instance of Scene.

Layouts:

JavaFX provides several layout panes that manage the process of placing elements in a scene. For example, the FlowPane class provides a flow layout and the GridPane class supports a row/column grid-based layout.

The Application Class and Life-Cycle methods:

A JavaFX application must be a subclass of the Application class, which is packaged in javafx.application. Thus, your application class will extend Application. The Application class defines three life-cycle methods that your application can override.

1. **void init( )** - The init( ) method is called when the application begins execution. It is used to perform various initializations.

2. **abstract void start(Stage primaryStage)** - The start( ) method is called after init( ). This is where the application begins and it can be used to construct and set the scene. This method is abstract and hence must be overridden by the application.

3. **void stop( )** - When the application is terminated, the stop( ) method is called. It can be used to handle any cleanup or shutdown chores.

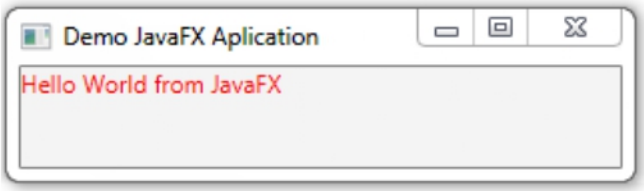
Solved exercise:

1. Write a JavaFX Application program to display a simple message.

```
import javafx.scene.control.*;
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;

public class HelloWorld extends Application {

  public void start(Stage primaryStage) { // entry point for the application
    primaryStage.setTitle("Demo JavaFX Aplication"); // set the title of top level
    //container.
    Label lbl = new Label();    // create a label control
    lbl.setText("Hello World from JavaFX");
    lbl.setTextFill(Color.RED);

    FlowPane root = new FlowPane(); // create a root node.
    root.getChildren().add(lbl); // add the label to the node
    Scene myScene = new Scene(root, 300, 50); // create a scene using the node
    primaryStage.setScene(myScene); // set the scene on the stage
    primaryStage.show(); // show the stage
    }
```

```
public static void main(String[] args) {
      launch(args); // Start the JavaFX application by calling launch( ).
  }
 }
```

Output:



Event Handling:

The JavaFX controls that respond to either the external user input events or the internal events need to be handled. The delegation event model is the mechanism of handling such events. The advantage of this model is that application logic that processes the events is cleanly separated from the User Interface logic that generates the event.

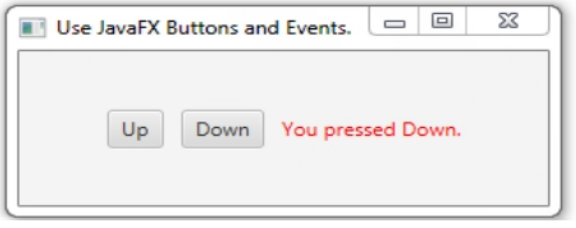
Solved exercise:

2. Write a JavaFX-Application program that handles the event generated by Button control.

```
import javafx.application.*;
import javafx.scene.*;
import javafx.stage.*;
import javafx.scene.layout.*;
import javafx.scene.control.*;
import javafx.event.*;
import javafx.geometry.*;
public class JavaFXEventDemo extends Application {
Label response;
public static void main(String[] args) {
// Start the JavaFX application by calling launch().
launch(args);
}

// Override the start() method.
public void start(Stage myStage) {
```

```
// Give the stage a title.
myStage.setTitle("Use JavaFX Buttons and Events.");
// Use a FlowPane for the root node. In this case,
// vertical and horizontal gaps of 10. FlowPane
 rootNode = new FlowPane(10, 10);
// Center the controls in the scene.
rootNode.setAlignment(Pos.CENTER);
// Create a scene.
Scene myScene = new Scene(rootNode, 300, 100);

// Set the scene on the stage.
myStage.setScene(myScene);
// Create a label.
response = new Label("Push a Button");
// Create two push buttons.
Button btnUp = new Button("Up");
Button btnDown = new Button("Down");
// Handle the action events for the Up button.
btnUp.setOnAction(new EventHandler<ActionEvent>() {
public void handle(ActionEvent ae) {
response.setText("You pressed Up."); } });
// Handle the action events for the Down button.
btnDown.setOnAction(new EventHandler<ActionEvent>() {
public void handle(ActionEvent ae) { response.setText("You
pressed Down."); } });
// Add the label and buttons to the scene graph.
rootNode.getChildren().addAll(btnUp, btnDown, response);
 // Show the stage and its scene.
myStage.show(); } }
```

Output:



Lab exercises:

1. Write a JavaFX application program to do the following:
   a. Display the message "Welcome to JavaFX programming" using Label in the Scene.
   b. Set the text color of the Label to Magenta.
   c. Set the title of the Stage to "This is the first JavaFX Application".
   d. Set the width and height of the Scene to 500 and 200 respectively.
   e. Use FlowPane layout and set the hgap and vgap of the FlowPane to desired values.
2. Write a JavaFX program to accept an integer from the user in a text field and display the multiplication table (up to number *10) for that number. Use FlowPane layout for the application.
3. Write a JavaFX program to display a window as shown below. Use TextField for UserName and PasswordField for Password input. On click of "Sign in" Button the message "Welcome UserName" should be displayed in a Text Control. Use GridPane layout for the application.

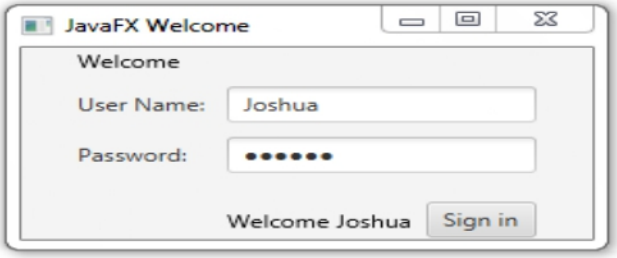


Fig 11.1 Welcome Window

4. Define a class called Employee with the attributes name, empID, designation, basicPay, DA, HRA, PF, LIC, netSalary. DA is 40% of basicPay, HRA is 15% of basicPay, PF is 12% of basicPay. Display all the employee information in a JavaFX application.

Additional exercises:

1. Write a JavaFX program that obtains two positive integers passed from the user in two text fields and displays the numbers and their GCD as the result on a Label.
2. Design a simple calculator to show the working for simple arithmetic operations. Use Grid layout.

JavaFX and Event Handling -Part II

Objectives:

In this lab, student will be able to:

4. Explore JavaFX controls.
5. Explore the application of JavaFX controls in different programs.
6. Know the fundamentals of event handling.

JavaFX:

JavaFX provides a powerful, streamlined, flexible framework that simplifies the creation of modern, visually exciting GUIs. The JavaFX framework has all of the good features of Swing.

JavaFX control classes:

Button	ListView	TextField
CheckBox	RadioButton	ToggleButton
Label	ScrollPane	TreeView

- Button: Button is JavaFX's class for push buttons. The procedure for adding an image to a button is similar to that used to add an image to a label. First obtain an ImageView of the image. Then add it to the button.
- ListView: List views are controls that display a list of entries from which you can select one or more.
- TextField: It allows one line of text to be entered. Thus, it is useful for obtaining names, ID strings, addresses, and the like. Like all text controls, TextField inherits TextInputControl, which defines much of its functionality.
- CheckBox: It supports three states. The first two is checked or unchecked, as you would expect, and this is the default behavior. The third state is indeterminate (also called undefined). It is typically used to indicate that the

state of the check box has not been set or that it is not relevant to a specific situation. If you need the indeterminate state, you will need to explicitly enable it.

- RadioButton: Radio buttons are a group of mutually exclusive buttons, in which only one button can be selected at any one time. They are supported by the RadioButton class, which extends both ButtonBase and ToggleButton.
- ToggleButton: A toggle button looks just like a push button, but it acts differently because it has two states: pushed and released. That is, when you press a toggle button, it stays pressed rather than popping back up as a regular push button does. When you press the toggle button a second time, it releases (pops up).
- Label: Label class encapsulates a label. It can display a text message, a graphic, or both.
- ScrollPane: JavaFX makes it easy to provide scrolling capabilities to any node in a scene graph. This is accomplished by wrapping the node in a ScrollPane. When a ScrollPane is used, scrollbars are automatically implemented that scroll the contents of the wrapped node.
- TreeView: It presents a hierarchical view of data in a tree-like format.

Solved exercise:

1. Write a JavaFX program to display an image in the Label.

```
import javafx.application.*;
import javafx.scene.*;
import javafx.stage.*;
import  javafx.scene.layout.*;
import javafx.scene.control.*;
import      javafx.geometry.*;
import javafx.scene.image.*;
public class LabelImageDemo extends Application {
 public static void main(String[] args) {
     launch(args);    // Start the JavaFX application by calling launch().      }
     public void start(Stage myStage) {    // Override the start() method.
 myStage.setTitle("Use an Image in a Label");    // Give the stage a title.
 FlowPane rootNode = new FlowPane();      // Use a FlowPane for the root node.
```

```
rootNode.setAlignment(Pos.CENTER);     // Use center alignment
 Scene myScene = new Scene(rootNode, 300, 200);       // Create a scene.
myStage.setScene(myScene);     // Set the scene on the stage.
// Create an ImageView that contains the specified image
 ImageView hourglassIV = new ImageView("hourglass.png");
// Create a label that contains both an image and text.
 Label hourglassLabel = new Label("Hourglass", hourglassIV);
rootNode.getChildren().add(hourglassLabel); // Add the label to the scene graph.
 myStage.show(); // Show the stage and its scene. } }
```

Output:

Fig 12.1 Display an image in label

2. Write a JavaFX Application program that handles the event generated by a Toggle Button.

```
import javafx.application.*;
import javafx.scene.*;
import javafx.stage.*;
import javafx.scene.layout.*;
import javafx.scene.control.*;
import javafx.event.*;
import javafx.geometry.*;
```

```
public class ToggleButtonDemo extends Application {
  ToggleButton tbOnOff; Label response;
  public static void main(String[] args) {
    // Start the JavaFX application by calling launch().
      launch(args);  }
  public void start(Stage myStage) {  //  Override  the  start()  method.
    myStage.setTitle("Demonstrate a Toggle Button");     // Give the stage a title.
// Use a FlowPane for the root node. In this case, vertical and horizontal gaps of 10.
    FlowPane rootNode = new FlowPane(10, 10);
    // Center the controls in the scene.
    rootNode.setAlignment(Pos.CENTER);
    Scene myScene = new Scene(rootNode, 220, 120);     // Create a scene.
    myStage.setScene(myScene);     // Set the scene on the stage.
    response = new Label("Push the Button."); // Create a label.
    tbOnOff = new ToggleButton("On/Off");     // Create the toggle button });
  } }
  //  Handle  action  events  for  the  toggle  button.
   tbOnOff.setOnAction(new EventHandler<ActionEvent>() {
   public void handle(ActionEvent ae) {
      if(tbOnOff.isSelected()
         response.setText("Button is on.");
      else  response.setText("Button is off.");      }      });
// Add the label and buttons to the scene graph.
rootNode.getChildren().addAll(tbOnOff, response);
myStage.show(); } }
// Show the stage and its scene.
```

Output:

Fig 12.2 Demonstration for toggle button

Lab Exercises:

1. Write a JavaFX application program that obtains two floating point numbers in two text fields from the user and displays the sum, product, difference and quotient of these numbers using Canvas on clicking compute button with a calculator image placed on it.
2. Write a JavaFX application program to create your resume. Use checkbox to select the languages which you can speak. On clicking the Submit button all the details of the resume should be displayed using Canvas.
3. Write a JavaFX application program that creates a thread which will scroll the message from right to left across the window or left to right based on RadioButton option selected by the user.
4. Write a JavaFX application program that displays the student details that are created in the earlier labs using Canvas based on the option chosen in List View (student register numbers).

Additional exercises:

1. Write a JavaFX program to simulate a static analog clock whose display is controlled by a user controlled static digital clock.
2. Write a JavaFX program to implement a simple calculator using the Toggle buttons.

REFERENCES

1. Herbert Schildt and Dale Skrien, "*Java Fundamentals – A Comprehensive Introduction*", McGrawHill, 2015.
2. Herbert Schildt, "*The Complete Reference JAVA Ninth Edition* ", Tata McGrawHill, 2017.
3. Dietel and Dietel, "*Java How to Program*", 9th Edition, Prentice Hall India, 2012.
4. Steven Holzner, "*Java 2 programming BlackBook*",DreamTech,India 2005.

JAVA QUICK REFERENCE:

- **Class** - A class can be defined as a template/ blue print that describe the behaviours/states that object of its type support.
- **Methods** - A method is basically behaviour. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- **Instance Variables** - Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.
- **Object** - Objects have states and behaviours. Example: A dog has states-colour, name, and breed as well as behaviours -wagging, barking, and eating. An object is an instance of a class.
- **Case Sensitivity** - Java is case sensitive which means identifier **Hello** and **hello** would have different meaning in Java.
- **Class Names** - For all class names the first letter preferred to be the Upper Case. If several words are used to form a name of the class each inner words first letter conventionally be in Upper Case. Example: class **MyFirstJavaClass**
- **Method Names** - All method names preferably start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter conventionally be in Upper Case. Example: **public void myMethodName( )**
- **Program File Name** - Name of the program file should exactly match the class name. When saving the file, save it using the class name (Remember java is case sensitive) and append '**.java**' to the end of the name. (If the file name and the class name do not match the program will not compile). Example: Assume '**MyFirstJavaProgram**' is the class name. Then the file should be saved as '**MyFirstJavaProgram.java**'
- **public static void main(String args[])** - java program processing starts from the **main( )** method which is a mandatory part of every java program.
- **Java Identifiers**: All Java components require names. Names used for classes, variables and methods are called identifiers. In java there are several points to remember about identifiers. They are as follows:

i) All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (_).
ii) After the first character identifiers can have any combination of characters.
iii) A key word cannot be used as an identifier.
iv) Most importantly identifiers are case sensitive.

Examples of legal identifiers:**age,$salary, _value, 1_value**
Examples of illegal identifiers :**123abc, net-salary**

- **Java Modifiers:**Like other languages, it is possible to modify classes, methods, etc., by using modifiers. There aretwo categories of modifiers.
  i) Access Modifiers:**default, public , protected, private**
  ii) Non-access Modifiers:**final, abstract**

- **Java Variable Types:**
  i) Local Variables
  ii) Class Variables (Static Variables)
  iii) Instance Variables (Non static variables)

- **Java Arrays:** Arrays are objects that store multiple variables of the same type. However an Array itself is an object on the heap.

- **Java Keywords:** The following list shows the reserved words in Java. These reserved words may not be used as constant or variable or any other identifier names.

abstract	asset	boolean	break	byte	case	catch	Char	class	const
continue	default	Do	double	else	enum	extends	Final	finally	flaot
for	goto	If	implements	import	Instance of	int	interface	long	native
new	package	Private	protected	public	return	short	static	strictfp	super
switch	synchronized	This	throw	throws	transient	try	Void	volatile	while

- **Comments in Java:** Java supports single line and multi-line comments very similar to C++. All characters available inside any comment are ignored by Java compiler.

Example 1:

```
/*This is my first java program.
* This will print 'Hello World' as the output
* This is an example of multi-line comments.
*/
```

Example 2:

```
//This is an example of single line comment
```

- **Data Types in Java:** There are two categories of data types available in Java:
  i) **Primitive Data Types:** There are eight primitive data types supported by Java. Primitive data types are predefined by the language and named by a key word. They are: **byte, short, int, long, float, double, Boolean, char**
  ii) **Reference/Object Data Types:** Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed. Examples: Employee, Puppy etc. Class objects, and various type of array variables come under reference data type. Default value of any reference variable is null. A reference variable can be used to refer to any object of the declared type or any compatible type.
  iii) Example :Animal animal = new Animal("giraffe");

- **Java Literals:** A literal is a source code representation of a fixed value. They are represented directly in the code without any computation. Literals can be assigned to any primitive type variable. For example: **byte a = 68; char a = 'A';**
  String literals in Java are specified like they are in most other languages by enclosing a sequence of characters between a pair of double quotes. Examples of string literals are:**"Hello World", "two\nlines", "\"This is in quotes\""**

  Java language supports few special escape sequences for String and char literals as well. Theyare:

Notation	Characterrepresented
\n	Newline (0x0a)
\r	Carriage return (0x0d)
\f	Formfeed (0x0c)

\b	Backspace (0x08)
\s	Space (0x20)
\t	tab
\"	Double quote
\'	Single quote
****	backslash
\ddd	Octal character (ddd)
\uxxxx	Hexadecimal UNICODE character (xxxx)

- **Java Access Modifiers:** Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors. The four access levels are:
  i) Visible to the package; the default; No modifiers are needed
  ii) Visible to the class only (private)
  iii) Visible to the world (public)
  iv) Visible to the package and all subclasses (protected)

- **Branch Structures:** The syntax of**if**,**if...else**,**if...else if...else**, Nested **if...else**,**switch-case, break**,and**continue**statements is similar to that of C++.

- **Loop Structures:** The syntax of**while, do...while,** and**for**statements is similar to that of C++. Java also provides *enhanced***for**loop. This is mainly used for arrays and collections.

```
for(declaration : expression){
    //Statements
}
```

- **Java Basic Operators:** Java provides a rich set of operators to manipulate variables. The important Java operators with their precedence and associativity are shown below.

Category	Operator	Associativity
Postfix	() [] . (dot operator)	Left to right
Unary	++ - - ! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>> >>> <<	Left to right
Relational	> >= < <=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

s