

MANIPAL INSTITUTE OF TECHNOLOGY

Manipal – 576 104

DEPARTMENT OF COMPUTER SCIENCE & ENGG.



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 2113] of Second Year B. Tech. Degree at MIT, Manipal, in the academic year

2017-2018.

Date:

Signature
Faculty in Charge

Signature
Head of the Department

CONTENTS

LAB NO.	TITLE	PAGE NO.	REMARKS
	COURSE OBJECTIVES AND OUTCOMES	i	
	EVALUATION PLAN	i	
	INSTRUCTIONS TO THE STUDENTS	ii – iii	
	SAMPLE LAB OBSERVATION NOTE PREPARATION	iv	
	INTRODUCTION TO JAVA	v	
1	SIMPLE JAVA PROGRAMS USING CONTROL STRUCTURES	1 –7	
2	1D AND 2D ARRAYS	8-15	
3	CLASSES AND OBJECTS	16-25	
4	CONSTRUCTORS AND STATIC MEMBERS	26-35	
5	STRINGS	36-42	
6	INHERITANCE AND PACKAGES	43-54	
7	INTERFACES AND EXCEPTION HANDLING	55-66	
8	MULTITHREADING	67-78	
9	GENERICS	79-86	
10	INPUT/OUTPUT	87-95	
11	APPLETS	96-106	
12	SWINGS AND EVENT HANDLING	107-116	
	REFERENCES	117	
	JAVA QUICK REFERENCE SHEET	118	

Course Objectives

- To develop object oriented programming skills using Java language
- To write and execute application and applet programs
- To develop skills in concurrent programming
- To develop efficient Graphical User Interfaces(GUI) using swing components
- To understand event handling mechanism of Java

Course Outcomes

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

- Understand object-oriented paradigm of software development
- Use the constructs of an object-oriented language Java in achieving object oriented principles
- Understand the packages of Java to develop applet and concurrent programs
- Achieve high level reusability using generics
- Design and implement small Java projects

Evaluation plan

- Internal Assessment Marks : 60%
 - ✓ Continuous evaluation component (for each experiment):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 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.

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

LAB NO: 1

Date:

Title: Simple java programs using control structures

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

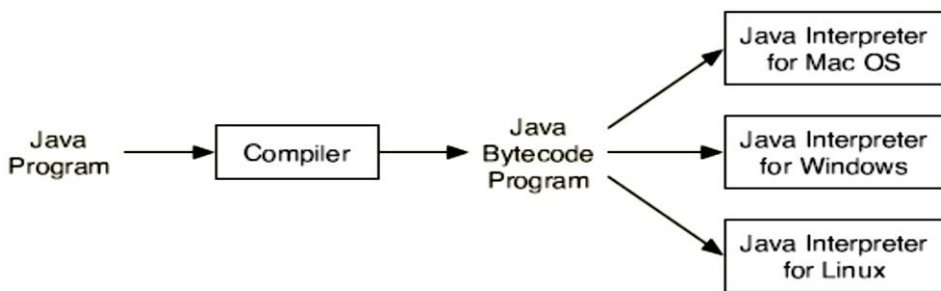
```
class Sample{  
    public static void main(String args[ ]){  
        System.out.println("Hello Java");  
    }  
}
```

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.



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:

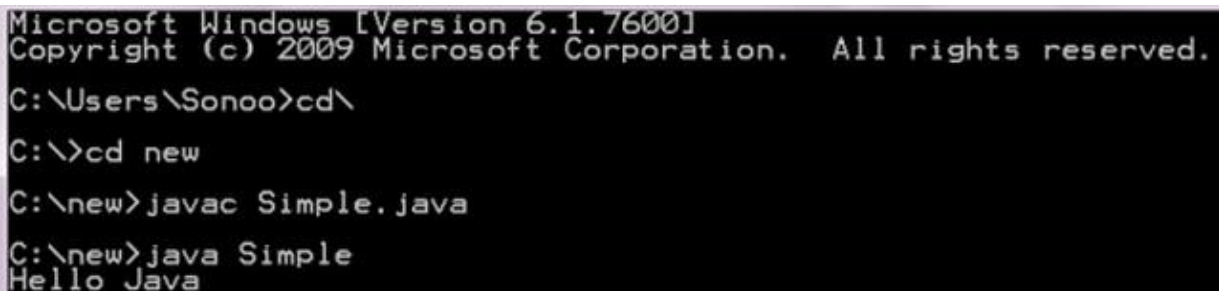
1. Write Java programs
2. Compile and interpret Java programs
3. Debug Java programs

Solved exercise

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

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

Output



```
Microsoft Windows [Version 6.1.7600]  
Copyright (c) 2009 Microsoft Corporation. All rights reserved.  
C:\Users\Sonoo>cd\  
C:\>cd new  
C:\new>javac Simple.java  
C:\new>java Simple  
Hello Java
```

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.class file representing bytecode
 - ✓ Interpret Sample.class in the terminal using the command
java Sample

Lab exercises

Write and execute Java programs to do the following:

1. Write a method **largest** to find the maximum of three numbers. Also write a main program to read 3 numbers and find the largest among them using this method.
2. Generate a pyramid as shown below

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

3. Write a method **fact** to find the factorial of a given number. Using this method, compute ${}^N C_R$ in the main method.
4. Write a method **isPrime** to check whether the given number is prime or not. Using this method, generate first N prime numbers in the main method.

Additional exercises

1. Compute all the roots of a quadratic equation using switch case statement.
Hint : $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
2. Write a method **isArmstrong** to check if an entered number is an Armstrong number.
3. Write a method **findSum** to find the sum of digits of a number.
4. Write a program **oddEven** to count the number of even and odd numbers among the N numbers read from the keyboard.

[OBSERVATION SPACE – LAB 1]

LAB NO: 2

Date:

1D AND 2D ARRAYS

Objectives:

In this lab, student will be able to:

1. Understand and 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 accessed 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 a1[] = 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 twoD with 4 rows and 5 columns.

```
inttwoD[][] = 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 twoD with 3 rows and first row with 1 column, second with 2 columns, and third with 3 columns.

```
inttwoD[][] = 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];
```

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

    // to read array elements
    System.out.println("Enter " + n + " integers");
    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

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. Reverse the elements of an integer array.
2. Arrange the elements in ascending/descending order using Bubble sort method.
3. Insert an element into a 1D array and delete an element from a 1D array.
4. Search an element in a 1D array using linear search.
5. Check if a matrix is symmetric or not. ($A=A^T$)
6. Find the addition of two matrices and display the resultant matrix.

Additional exercises

1. Find the second largest and second smallest elements in a 1D array without sorting.
2. Print all the prime numbers in a given 1D array.
3. Check if a matrix is lower triangular or not.
4. Find the trace and norm of a matrix

[Hint: trace=sum of diagonal elements, norm=sqrt(sum of squares of elements of the matrix)]

[OBSERVATION SPACE – LAB 2]

LAB NO: 3

Date:

CLASSES AND OBJECTS

Objectives:

In this lab student will be able to:

1. Know the fundamentals of the class
2. Understand how objects are created
3. Understand how reference variables are assigned
4. Understand 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:

```
classclassname{
    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.

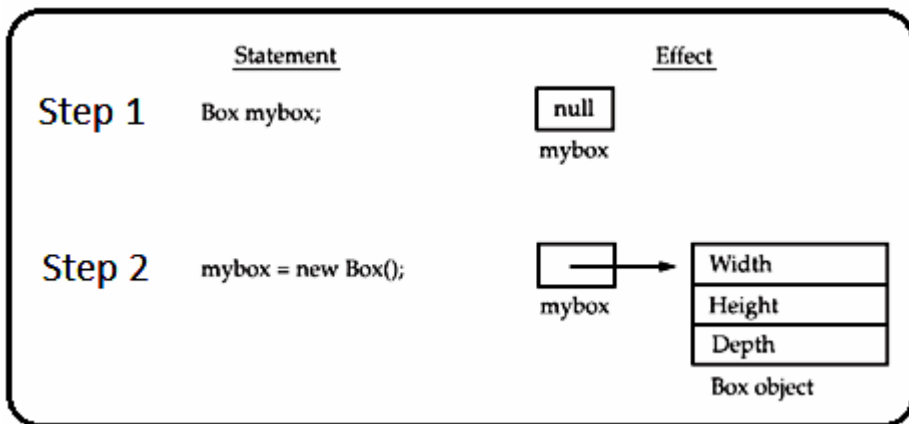
Consider a class `Box` whose object `mybox` is created as follows

```
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:



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
        double vol;

        // 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 to represent a complex number called **Complex**. Provide the following member functions:-

- i. To assign initial values to the **Complex** object.
- ii. To display a complex number in a+ib format.
- iii. To add 2 complex numbers. (the return type should be **Complex**)
- iv. To subtract 2 complex numbers

Write a main method to test the class.

2. Create a class called **Time** that has data members to represent hours, minutes and seconds. Provide the following member functions:-

- i. To assign initial values to the **Time** object.
- ii. To display a **Time** object in the form of hh:mm:ss {24 hours format}

- iii. To add 2 Time objects (the return type should be a Time)
- iv. To subtract 2 Time objects (the return type should be a Time)
- v. To compare 2 Time objects and to determine if they are equal or if the first is greater or smaller than the second one.

3. Define a class **Mixer** to merge two sorted integer arrays in ascending order with the following data members and member functions:

Data members:

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

Member functions:

```
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 data members are:

- i. An integer array
- ii. An integer for storing the top of stack (tos)

Include member functions 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: $1 \times \text{digit1} + 2 \times \text{digit2} + 3 \times \text{digit3} + 4 \times \text{digit4} + 5 \times \text{digit5} + 6 \times \text{digit6} + 7 \times \text{digit7} + 8 \times \text{digit8} + 9 \times \text{digit9} + 10 \times \text{digit10}$ is divisible by 11.

example: For an ISBN 1401601499:

$\text{Sum} = 1 \times 1 + 2 \times 4 + 3 \times 0 + 4 \times 1 + 5 \times 6 + 6 \times 0 + 7 \times 1 + 8 \times 4 + 9 \times 9 + 10 \times 9 = 253$ 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 sideUp to 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.
 3. Create a Swapper class with two integer instance variables **x** and **y** and a method intVar() with two parameters that initialize the two variables. Also include three methods: A getX() method that returns **x** , a getY() method that returns **y**, and a void swap() method that swaps the values **x** and **y**. Then create a SwapperDemo class that tests all the methods.

[OBSERVATION SPACE – LAB 3]

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. Understanding static

Introduction to constructors:

A constructor initializes an object when it is created. It has the same name as that of class without 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);
    }
    Student(Student s){        // Parameterized constructor( object)
        id = s.id;
```

```

        name =s.name;
        System.out.println("inside parameterized(object) 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
        Student s3 = new Student(s2); //calling parameterized constructor passed with an object
    }
}

```

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 method `valDiv2()` 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 `Complex` class. Provide a default constructor and parameterized constructor 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 `Time` class. Provide a default constructor and parameterized constructors (one takes time in seconds, second takes time in minutes and seconds, third one takes time in seconds, minutes and hours) to this class. Also provide a display method. Illustrate all the constructors as well as the display method by defining `Time` objects.
3. Create a **Card** class that represents a playing card. It should have an **int** instance variable named **rank** and a **char** variable named **suit**. Give it a constructor with two parameters for initializing the two instance variables and give it a **getSuit()** method and a **getRank()** method that returns the values of the two instance variables. Then create a **CardTester** class with a main method that creates five **Cards** and test whether that set make up a full house or not (that is three of the cards have the same rank and the other two cards have the same rank) and prints out the ranks and suits of the five **Cards** using **getSuit()** and **getRank()** methods. (Four suits are S, C, H, D and in each suit ranks are from 1 to 13).
4. 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, a parameterized constructor and a copy constructor to this class. Also provide Member Functions:-

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.

5. 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. Create a Date class with three integer instance variables named **day**, **month**, **year**. It has a constructor with three parameters for initializing the instance variables, and it has one method named **daysSinceJan1()**. It computes and returns the number of days since January 1 of the same year, including January 1 and the day in the **Date** object. For example if **day** is a **Date** object with **day** =1 , **month** =3 and **year** =2000 , then the call **date.daysSinceJan1()** should return 61 since there are 61 days between the dates of January 1,2000, and March 1, 2000 including January 1 and March 1. Include a **DateDemo** class that tests the **Date** class. Consider leap years as well.
2. Define a class **IntArr** which hosts an array of integers. Provide the following member functions:-
 1. A **default constructor**.
 2. A **parameterized constructor** which initializes the array of the object.
 3. A function called **display** to display the array contents.
 4. A function called **search** to search for an element in the array.
 5. A function called **compare** which compares 2 **IntArr** objects for equality.
3. Create a class called **TestString** which consists of only one data member which is a character array. Provide the following member functions to this class:-
 1. A default constructor which initializes the pointer to null value.
 2. A parameterized constructor which receives a string as its parameter. {Note:- memory allocation to be done}
 3. A **display** to display the string object.
 4. **ChangeCase**, which converts all lower case to upper case and vice versa.
 5. **Reverse**, which reverses the character array.
4. Define a class called Customer that holds private fields for a customer ID number, name and credit limit. Include appropriate constructors to initialize the data members of the Customer Class. Write a main() function that declares an array of 5 Customer objects. Prompt the user for values for each Customer, and display all 5 Customer objects.

[OBSERVATION SPACE – LAB 4]

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
char charAt(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(int i=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. Design a class which represents a student. Every student record is made up of the following fields.
 - i) Registration number (int)
 - ii) Full Name (String)
 - iii) Date of joining (Gregorian calendar)
 - iv) Semester (short)
 - v) GPA (float)

vi) 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 80th student to join then his registration number will be 1280.

Write member functions to do the following.

- a) Provide default and 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.
 - d) Static method to sort the student records with respect to semester and CGPA.
 - e) Static method to sort the student record with respect to name.
 - f) Static method to list all the students whose name starts with a particular character.
 - g) Static method to list all the student names containing a particular sub string.
 - h) Change the full name in the object to name with just initials and family name. For example, *Calyampudi Radhakrishna Rao* must be changed to *C. R. Rao* and store it in the object. Display modified objects.
2. 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.

[OBSERVATION SPACE – LAB 5]

LAB NO: 6

Date:

INHERITANCE AND PACKAGES

Objectives:

In this lab student will be able to:

1. Understand Inheritance basics
2. Use super keyword to access superclass members and constructors
3. Understand dynamic polymorphism by overriding methods
4. Differentiate between abstract classes and concrete classes
5. Know the purpose and creation of packages
6. Know how to import packages and how packages affect access

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;
    doublegetWidth() {    return width;}
    doublegetHeight(){    return height;  }
    voidsetWidth(double w){    width=w;    }
    voidsetHeight(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

```

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 by 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");}
}

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

Output:

Hello

Lab Exercises

1. Create a **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 **CollegeGraduate** 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.
2. Design a class, which represents a patient. Fields of this class are name, age and hospital number. Provide methods to input and display all fields. Next design a class called Inpatient, which inherits from patient class. Provide fields to represent department name, admission date and room type. Provide methods to input and display these fields. Next design a class called Billing, which inherits from Inpatient class. Provide a field to represent the discharge date. Provide methods to input this field value as well as to display the total amount. The total amount is calculated based on room type and doctor charges as shown below:-

RoomType	Consultancy Charges/day	Room Rent Charges/day
Special	Rs. 1000.00	Rs. 200
SemiSpecial	Rs. 500.00	Rs. 100
General	Rs. 100.00	Rs. 50

Show the use of a Billing object in main (Make use of super keyword).

3. Define a class Max with the following static methods
 - i) max (which finds maximum among three integers and returns the maximum integer)
 - ii) max (which finds maximum among three floating point numbers and returns the maximum among them)
 - iii) max (which finds the maximum in an array and returns it)
 - iv) max (which finds the maximum in a matrix and returns the result)

Place this in a package called p1. Let this package be present in a folder called "myPackages", which is a folder in your present working directory (eg:- c:\student\3rdsem \mypackages\p1). Write a main method to use the methods of Max class in a package p1.

4. Create an abstract class **Figure** with abstract method **area** and two integer dimensions. Extend this class to inherit three more classes **Rectangle**, **Triangle** and **Square** which implement the **area** method. Show how the **area** can be computed dynamically during run time for **Rectangle**, **Square** and **Triangle** to achieve dynamic polymorphism.
5. 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. Place the **Building**, **House**, and **School** classes in a package named **com.course.buildings**. Create a main method that declares objects of each type and uses the package.

Additional Exercises

1. 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.

2. Create a method **TransMat()** to find the transpose of a matrix. Place this in a package called **pTrans**. Let this package be present in a folder called “myPackages”, which is a folder in your present working directory (eg:- c:\student\3rdsem\mypackages\pTrans). Write a main method in package **pCheck** to read a matrix and display it. Using the methods **TransMat()** from package **pTrans** check if the entered matrix is symmetric or not. [Hint :a matrix A is symmetric if $A=A^T$]

[OBSERVATION SPACE – LAB 6]

LAB NO: 7

Date:

INTERFACES AND EXCEPTION HANDLING

Objectives:

In this lab student will be able to:

1. Understand interface fundamentals
2. Implement interfaces and apply interface references
3. Use interface constants, extend interfaces.
4. Know the exception mechanism of Java
5. Know Java's built-in exceptions and to create custom exceptions

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
```

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 is 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 `RuntimeException` class.

3. The following `InsufficientFundsException` 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 `CheckingAccount` class contains a `withdraw()` method that throws an `InsufficientFundsException`.

```
// File Name CheckingAccount.java
```

```
import java.io.*;
public class CheckingAccount{
    private double balance;
    private int number;
    public CheckingAccount(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 an interface called Stack with 2 methods namely push() and pop() in it. Design a class called FixedStack, which implements a fixed length version of the stack. Also design a class called DynamicStack, which implements a growable version of the stack implementing Stack interface. Write main method, which uses both these classes through an interface reference.

2. Design a class, which represents an employee. The data members are:- name (string), age (int), grossSalary (double), takeHomeSalary (float), grade (char). Provide methods called input() and display() which reads all details of a record from the keyboard and displays them respectively. Handle IOException while reading from the keyboard. Provide a menu with the options: Input, Display and Exit to read users choice. (Make use of Wrapper classes)
3. Design a Student class. Provide your own exceptions namely SeatsFilledException , the exception 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 in Student object in the main.(Note: Registration number must be a unique number)
4. Design an interface called Series with the following methods
 - i) getNext(returns the next number in series)
 - ii) reset(to restart the series)
 - iii) setStart(to set the value from which the series should start)

Design a class named **ByTwos** 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

Additional Exercises

1. Write a method which accepts an index and returns the corresponding element from an array. If the index is out of bounds, the method should throw an exception. Handle this exception in main()
2. Design a stack class. Provide your own stack exceptions namely PushException and PopException, 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.
3. Design a class Student with the methods ,getNumber and putNumber to read and display the RollNo of each student and getMarks() and putMarks() to read and display their marks. Create an interface called Sports with a method putGrade() that will display the grade obtained by a student in Sports. Design a class called Result that will implement the method putGrade() and generate the final result based on the grade in sports and the marks obtained from the superclass Student.

[OBSERVATION SPACE – LAB 7]

LAB NO: 8

Date:

MULTITHREADING

Objectives:

In this lab, student will be able to:

1. Write and execute multithreaded programs
2. Understand 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

```

classNewThread 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.");
}
}

classExtendThread {
    public static void main(String args[]) {
        newNewThread();    // 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.

```

classNewThread 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.");
    }
}
classMultiThreadDemo {

```

```

public static void main(String args[ ]) {
    newNewThread("One");           // start threads
    newNewThread("Two");
    newNewThread("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

```

New thread: Thread[One,5,main]
New thread: Thread[Two,5,main]
One: 5
New thread: Thread[Three,5,main]
Two: 5
Three: 5
Two: 4
One: 4
Three: 4
Two: 3
One: 3
Three: 3
Two: 2
One: 2
Three: 2
Two: 1
One: 1
Three: 1
Two exiting.
One exiting.
Three exiting.
Main thread exiting.

```

Lab exercises

1. 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.

2. Create a class *Table* with method *void display(int x)* for printing multiplication table. Create a class called *DisplayTable* which extends *Thread*. *Thread* will call *display()* method of *Table* class. Create a class called *TableDemo* which will instantiate the two thread objects. The first thread prints multiplication table for 5 in the form $1 * 5 = 5$ (first 10 terms in different lines) and the second thread prints multiplication table for 7.
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. A. 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.
 B. 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.
 C. 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.
 D. Write a *CounterUser* 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.*

[OBSERVATION SPACE – LAB 8]

LAB NO: 9

Date:

GENERICIS

Objectives:

In this lab, student will be able to:

1. Understand 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() {
        return ob;
    }
    // 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. Define a simple generic stack class and show the use of the generic class for two different class types Student and Employee class objects.
2. Define a generic class NumericFns to illustrate bounded types. The class defines one type parameter and defines the methods reciprocal() and fraction(). Define another class BoundsDemo which uses NumericFns class.
3. Add a method absEqual() to the class NumericFns defined in question 2, which takes one wild card argument of type NumericFns and compares the absolute value of that argument against the absolute value of invoking object, returning true only if the absolute values are same. Add the statements to invoke the absEqual() method from main() method.
4. Write a generic method that can print array of different type using a single Generic method.

Additional exercises:

1. Write a generic method to exchange the positions of two different elements in an array.
2. Define a generic List class to implement a singly linked list and show the use of the generic class for two different class types Integer and Double class objects.
3. Write a generic method to return the largest of three Comparable objects.

[OBSERVATION SPACE – LAB 9]

LAB NO: 10**Date:**

INPUT / OUTPUT

Objectives:

In this lab, student will be able to:

1. Understand the meaning of streams, byte stream and character stream classes
2. Know to work with files

Introduction:

Streams

Java programs perform I/O through streams. A stream is an abstraction that either produces or consumes information. A stream is linked to a physical device by the Java I/O system. All streams behave in the same manner, even if the actual physical devices to which they are linked differ. Thus, the same I/O classes and methods can be applied to any type of device. This means that an input stream can abstract many different kinds of input: from a disk file, a keyboard, or a network socket. Likewise, an output stream may refer to the console, a disk file, or a network connection. Stream abstracts various types of I/O devices in coding and provides a common access to all those varieties. Java implements streams within class hierarchies defined in the *java.io package*.

Byte Streams and Character Streams

Java defines two types of streams: byte and character. Byte streams provide a convenient means for handling input and output of bytes. Byte streams are used, for example, when reading or writing binary data. Character streams provide a convenient means for handling input and output of characters. They use Unicode and, therefore, can be internationalized. Also, in some cases, character streams are more efficient than byte streams.

Solved exercises:

1. Program to copy contents of input.txt to output.txt using byte stream classes

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

```

{
FileInputStream in = null;
FileOutputStream out = null;
try {
in = new FileInputStream("D:\\input.txt");
out = new FileOutputStream("D:\\output.txt");

int c;
    while((c = in.read()) != -1) {
out.write(c);
    }
} finally {
if (in != null) {
in.close();
    }
if (out != null) {
out.close();
    }
    }
}
}

```

2. Program to copy contents of input.txt to output.txt using character stream classes

```

import java.io.*;
public class CopyFile {
public static void main(String args[]) throws IOException
{
FileReader in = null;
FileWriter out = null;

try {
in = new FileReader("D:\\input1.txt");
out = new FileWriter("D:\\output1.txt");
int c;
while ((c = in.read()) != -1) {
out.write(c);
    }
} finally {
if (in != null) {
in.close();

```

```

    }
    if (out != null) {
        out.close();
    }
}
}
}

```

3. Program to display contents of a directory.

```

import java.io.File;
public class ReadDir {
    public static void main(String[] args) {
        File file = null;
        String[] paths;
        try {
            file = new File("D:\\"); // create new file object
            paths = file.list();      // array of files and directory
            for (String path: paths)  // for each name in the path array
            {
                System.out.println(path); // prints filename and directory name
            }
        } catch (Exception e) {
            e.printStackTrace();      // if any error occurs
        }
    }
}

```

Lab exercises:

1. Write a program to display the listing of a given directory and its sub directories using recursion.
2. Count the number of characters, numbers (sequence of 1 or more digits) , words and lines from a file 'fileNames.txt' and place the result on the screen.
3. Write and execute a Java program that reads a specified file and searches for a specified word, printing all the lines on which that word found, by preceding it with its line number.

Additional exercises:

1. Assume that there exists a file with different file names stored in it. Every file name is placed on a separate line. Create two threads, which scan the first half, and the second half of the file respectively, to search the filenames which end with .java Write those file names onto the screen.
2. Write a program to copy all files from source directory onto destination directory. Read source and destination directory names from keyboard.

[OBSERVATION SPACE – LAB 10]

LAB NO: 11**Date:****APPLETS****Objectives:**

In this lab, student will be able to:

1. Write and execute applet programs
2. Understand applet architecture and override the paint method
3. Learn passing parameters to applets

Introduction to applets

A Java program can be console-based application or an applet program. Applets are small applications that are accessed on an Internet server, transported over the Internet, automatically installed, and run as part of a web document. After an applet arrives on the client, it has limited access to resources so that it can produce a graphical user interface and run complex computations without introducing the risk of viruses or breaching data integrity.

Let's begin with the simple applet shown here:

```
import java.awt.*;  
import java.applet.*;  
public class SimpleApplet extends Applet {  
    public void paint(Graphics g) {  
        g.drawString("A Simple Applet", 20, 20);  
    }  
}
```

This applet begins with two import statements. The first imports the Abstract Window Toolkit (AWT) classes. Applets interact with the user (either directly or indirectly) through the AWT, not through the console-based I/O classes. The AWT contains support for a window-based, graphical user interface. The second import statement imports the applet package, which contains the class Applet. Every applet created must be a subclass of Applet. The next line in the program declares the class SimpleApplet. This class must be declared as public, because it will be accessed by code that is outside the program. To output a string to

an applet, use `drawString()`, which is a member of the `Graphics` class. Typically, it is called from within either `update()` or `paint()`.

It has the following general form:

```
void drawString(String msg, int X, int Y)
```

Here, `msg` is the string to be output beginning at `X, Y`. In a Java window, the upper-left corner is location `0,0`. The `drawString()` method doesn't recognize newline characters. To start a line of text on another line, do so manually by specifying the precise `X,Y` location.

There are two ways in which an applet can be run.

1. Executing the applet within a Java-compatible web browser

To execute an applet in a web browser, write a short HTML text file that contains `APPLET` tag that loads the applet. Here is the HTML file that executes `SimpleApplet`:

```
<applet code="SimpleApplet" width=200 height=60>
</applet>
```

The width and height statements specify the dimensions of the display area used by the applet.

2. Using an applet viewer, such as the standard tool, `appletviewer`

An applet viewer executes the applet in a window. This is generally the fastest and easiest way to test the applet. Include a comment at the head of the Java source code file that contains the `APPLET` tag. By doing so, the code is documented with a prototype of the necessary HTML statements, and the compiled applet can be tested by starting the applet viewer with your Java source code file. With this method, the `SimpleApplet` source file looks like this:

```
import java.awt.*;
import java.applet.*;
/*<applet code="SimpleApplet" width=200 height=60></applet>
*/
public class SimpleApplet extends Applet {
    public void paint(Graphics g) {
        g.drawString("A Simple Applet", 20, 20);
    }
}
```

With this approach, you can quickly iterate through applet development by using these three steps:

1. Edit a Java source file.
2. Compile your program.

3. Execute the applet viewer, specifying the name of your applet's source file. The applet viewer will encounter the APPLET tag within the comment and execute the applet.

C:\>appletviewer SimpleApplet.java

The window produced by SimpleApplet, as displayed by the applet viewer, is shown in the following illustration:



Solved exercise:

1. Write an applet program to set background colour to cyan, to display a message, to show the order of invocation of `init()`, `start()` and `paint()` methods of Applet class, when an applet is initialized, in an applet window and to display the message "This is shown in the status window" in status window of an applet.

```
import java.awt.*;
import java.applet.*;
```

```
/* <applet code="FirstApplet" width=300 height=50></applet> */
```

```
public class FirstApplet extends Applet{
    String msg;
    public void init() {
        setBackground(Color.cyan);
        msg+="init()→";
    }
    public void start() {
        msg+="start()→";
    }
    // Display msg in applet window.
    public void paint(Graphics g) {
        msg+="paint()";
        g.drawString(msg, 10, 20);
    }
}
```

```

        showStatus("This is shown in the status window.");
    }
}

```

Passing Parameters to Applets

The parameters can be passed to an applet using the APPLET tag in HTML. To retrieve a parameter, use the **getParameter()** method. It returns the value of the specified parameter in the form of a **String** object. Thus, for numeric and **boolean** values, conversion from their string representations into their internal formats is necessary.

2. Program that demonstrates passing parameters and using them.

```

import java.awt.*;
import java.applet.*;
/*
<applet code="ParamDemo" width=300 height=80>
    <param name=fontName value=Courier>
    <param name=fontSize value=14>
    <param name=leading value=2>
    <param name=accountEnabled value=true>
</applet>
*/
public class ParamDemo extends Applet{
    String fontName;
    int fontSize;
    float leading;
    boolean active;

    // Initialize the string to be displayed.

    public void start() {
        String param;
        fontName = getParameter("fontName");
        if(fontName == null)
            fontName = "Not Found";
        param = getParameter("fontSize");
    try {
        if(param != null)           // if not found
            fontSize = Integer.parseInt(param);
    }
}

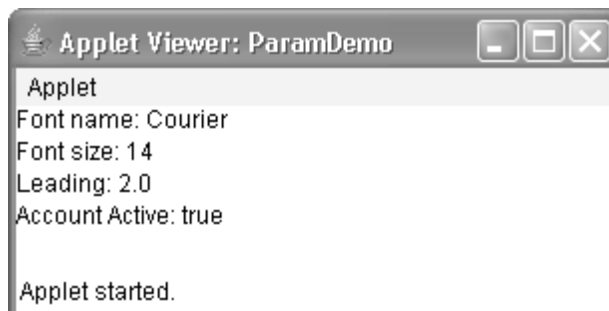
```

```

        else
            fontSize = 0;
    } catch(NumberFormatException e) {
        fontSize = -1;
    }
    param = getParameter("leading");
    try {
        if(param != null)           // if not found
            leading = Float.valueOf(param).floatValue();
        else
            leading = 0;
    } catch(NumberFormatException e) {
        leading = -1;
    }
    param = getParameter("accountEnabled");
    if(param != null)
        active = Boolean.valueOf(param).booleanValue();
    }
    public void paint(Graphics g) {           // Display parameters.
        g.drawString("Font name: " + fontName, 0, 10);
        g.drawString("Font size: " + fontSize, 0, 26);
        g.drawString("Leading: " + leading, 0, 42);
        g.drawString("Account Active: " + active, 0, 58);
    }
}

```

Sample output from this program is shown here:



Lab exercises:

1. Write an applet java program to do the following:
 - Display the message “Welcome to applet programming” in the applet window
 - Set foreground and background with some colours
 - Display the message “This is status window” in the status window of the applet.
2. Write an applet that obtains an integer passed as a parameter and displays the multiplication table for that number.
3. Write an applet program that creates a thread which will scroll the message from right to left across the applet's window.
4. Write an applet that obtains two positive integers passed from the APPLET tag and displays the numbers and their GCD in the applet window.

Additional exercises:

1. 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 the applet with all the employee information. Pass the required parameters inside the applet tag.
2. Design a simple calculator to show the working for simple arithmetic operations. Use Grid layout.

[OBSERVATION SPACE – LAB 11]

LAB NO: 12

Date:

SWINGS AND EVENT HANDLING

Objectives:

In this lab, student will be able to:

1. Understand importance of light weight components and pluggable look-and-feel
2. Understand MVC architecture and model-delegate architecture
3. Create, compile and run swing applications
4. Know the fundamentals of event handling and role of layout managers

Swings:

JDK 1.2 was introduced with a new set of packages the Java Foundation Classes (JFC) that includes an improved user interface components called swing components. These components are a collection of light weight visual components. The root class of all swing components is JComponent which is derived from Container of AWT (Abstract Window Toolkit) package.

The Model-View-Controller (MVC) Architecture:

It is an architecture used to implement a visual component consisting of three distinct aspects:

- i. The state information associated with the component - Model
- ii. The way that the component looks when rendered on the screen - View
- iii. The way that the component reacts to the user - Controller

By separating a visual component into the above three aspects, the specific implementation of each aspect can be changed without affecting the other two. For instance, different view implementations can render the same component in different ways without affecting the model or the controller.

The Model-Delegate (Separable Model) Architecture:

The MVC architecture in principle is conceptually sound. But the high level separation between the View and the Controller was not beneficial for swing components. Instead swing uses a modified version of MVC that combines View and the Controller into a single logical entity called UI delegate. For this reason, swing's approach is called Model-Delegate Architecture. The presence of UI delegates is behind the screen

of coding. Anyhow, swings enable the programmer more focus on GUI components rather than on their implementation details.

Event Handling:

The swing components 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:

1. Write a swing application program to display a message in a frame.

```
//A Swing application.
import javax.swing.*; // Swing programs must import javax.swing.* package
class SwingDemo {
    SwingDemo() {
        //Create a new top-level JFrame container.
        JFrame jfrm = new JFrame("A Simple Swing Application");
        //Give the frame an initial size.
        jfrm.setSize(275, 100);
        //Terminate the program when the user closes the application.
        jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //Create a text-based label.
        JLabel jlab = new JLabel(" Swing means powerful GUIs.");
        jfrm.add(jlab); //Add the label to the content pane.
        jfrm.setVisible(true); //Display the frame.
    }

    public static void main(String args[]) {
        //Create the frame on the event dispatching thread.
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new SwingDemo();
            }
        });
    }
}
```

Output:

2. Swing-Applet program that handles the event generated by a Swing push button.

// A simple Swing-based applet

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
/*
```

This HTML can be used to launch the applet:

```
<object code="MySwingApplet" width=220 height=90>
</object>
```

```
*/
```

```
public class MySwingApplet extends JApplet {
    JButton btnAlpha;
    JButton btnBeta;
    JLabel lab;
    public void init() {          // Initialize the applet
        try {
            SwingUtilities.invokeLater(new Runnable() {
                public void run() {
                    makeGUI();          // initialize the GUI
                }
            });
        }
        catch (Exception exc) {
            System.out.println("Can't create because of " + exc);
        }
    }
}
```

// This applet does not need to override start(), stop(), or destroy().

// Set up and initialize the GUI.

```
private void makeGUI() {
    setLayout(new FlowLayout());          // Set the applet to use flow layout.
    btnAlpha = new JButton("Alpha");      // Make two buttons.
```

```

jbtnBeta = new JButton("Beta");
jbtnAlpha.addActionListener(new ActionListener() { // Add action listener for Alpha.
    public void actionPerformed(ActionEvent le) {
        jlab.setText("Alpha was pressed.");
    }
});

// Add action listener for Beta.
jbtnBeta.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent le) {
        jlab.setText("Beta was pressed.");
    }
});

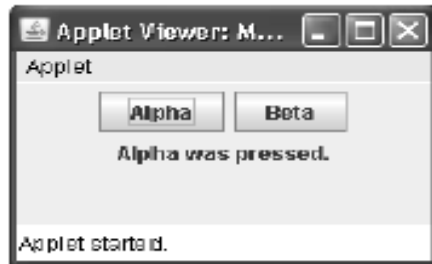
// Add the buttons to the content pane.
add(jbtnAlpha);
add(jbtnBeta);

// Create a text-based label.
jlab = new JLabel("Press a button.");

// Add the label to the content pane.
add(jlab);
}
}

```

Output:



Lab exercises:

1. Write a swing applet 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 on clicking Compute button.
2. Write Java program that allows the user to draw a rectangle by dragging the mouse on the application window. The upper-left coordinate should be the location where the user presses the

mouse button, and the lower-right coordinate should be the location where the user releases the mouse button. Also display the area of the rectangle.

3. Modify the program 2, to draw a shape with the mouse. The shape should be allowed to choose from an oval, an arc, a line, a rectangle with rounded corners and predefined polygon. Also display the mouse coordinates in the status bar.
4. Write a Java program that displays a Circle of random size and calculates and displays the area, radius, diameter and circumference.

Additional exercises:

1. Write a Java program to simulate a static analog clock whose display is controlled by a user controlled static digital clock.
2. Write a Java application or an applet program to implement a simple calculator.

[OBSERVATION SPACE – LAB 12]

REFERENCES

1. Herbert Schildt and Dale Skrien, “*Java Fundamentals – A Comprehensive Introduction*”, McGrawHill, First Edition, 2013.
2. Herbert Schildt, “*The Complete Reference JAVA 2*”, Tata McGrawHill, 8th Edition 2011.
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 are two 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	assert	boolean	break	byte	case	catch	char	class	const
continue	default	do	double	else	enum	extends	final	finally	float
for	goto	If	implements	import	instanceof	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.

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. They are:

<u>Notation</u>	<u>Character represented</u>
<code>\n</code>	Newline (0x0a)
<code>\r</code>	Carriage return (0x0d)
<code>\f</code>	Formfeed (0x0c)
<code>\b</code>	Backspace (0x08)
<code>\s</code>	Space (0x20)
<code>\t</code>	tab
<code>\"</code>	Double quote
<code>'</code>	Single quote
<code>\\</code>	backslash
<code>\ddd</code>	Octal character (ddd)
<code>\uxxxx</code>	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