MANIPAL INSTITUTE OF TECHNOLOGY
BENGALURU
(A constituent unit of MAHE, Manipal)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Certificate

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

Reg. No. …..…………………… Section: ……………… has satisfactorily completed the lab exercises prescribed for Object Oriented Programming Lab [CSE2163] of Third Semester B. Tech. [CSE] Degree at MIT, Bengaluru, in the academic year 2022-2023.

Date: ……...................................

Signature of the faculty

Signature
Head of the Department

INDEX

Course Objectives

- To understand the programming skills using object orientation concepts through Java
- To write, compile and execute application programs in Java
- To develop skills of Exception Handling, Multithreading, concurrent programming, Stings in Java
- To develop efficient Graphical User Interfaces (GUI) using JavaFx components
- To understand event handling mechanism of Java

Course Outcomes

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

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

Evaluation plan

- Internal Assessment Marks: 60%
    - Continuous evaluation component (in alternate weeks):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

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

```
class Sample{

        public static void
            main(String args[]){
            System.out.println("H
            ello 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.

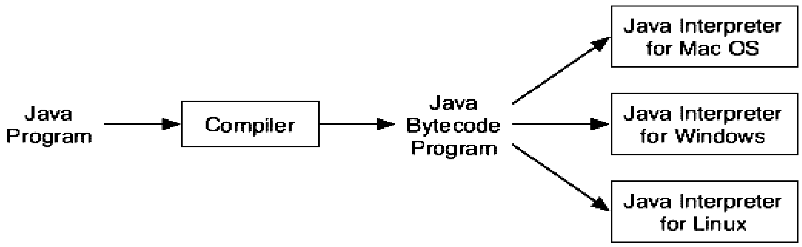

Java Program
Compiler
Java Bytecode Program
Java Interpreter for Mac OS
Java Interpreter for Windows
Java Interpreter for Linux

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:**

JAVA FEATURES & SIMPLE PROGRAMS USING CONTROL STRUCTURES

Objectives:

1. To know the features of Java
2. Understand the Java Development Kit (JDK)
3. To write, compile, and run a Java program
4. To know the execution steps using Netbeans/Eclipse IDE & Command Prompt
5. Write Java programs using control structures

1.1 Features of Java Language

Java is truly object oriented programming language mainly used for Internet applications. It can also be used for standalone application development. Following are the main features of Java:

Simple: Java was designed to be easy for the professional programmer to learn and use effectively. Design goal was to make it much easier to write bug free code. The most important part of helping programmers write bug-free code is keeping the language simple. Java has the bare bones functionality needed to implement its rich feature set. It does not add unnecessary features.

Object Oriented: Java is a true object oriented language. Almost everything in Java is an object. The program code and data are placed within classes. Java comes with an extensive set of classes and these classes are arranged in packages.

Robust: Memory management can be a difficult and tedious task in traditional programming environments. For example, in C/C++, the programmer must manually allocate and free all dynamic memory. This sometimes leads to problems, because programmers will either forget to free memory that has been previously allocated or, worse, try to free some memory that another part of their code is still using. Java virtually eliminates these problems by managing memory allocation and de-allocation. (de-allocation is completely automatic, because Java provides garbage collection for unused objects.) Exceptional conditions in traditional environments often arise in situations such as division by zero or "file not found," and they must be managed with clumsy and hard-to-read constructs. Java helps in this area by providing object-oriented exception handling. In a well-written Java program, all run-time errors can and should be managed by the program.

Multithreaded: Java was designed to meet the real-world requirement of creating interactive, networked programs. To accomplish this, Java supports multithreaded programming, which allows to write programs that do many things simultaneously. The java run-time system comes with a sophisticated solution for multi-process synchronization that enables users to construct smoothly running interactive systems.

Compiled and Interpreted: Java is a two stage system because it combines two approaches namely, compiled and interpreted. First Java compiler translates source code into what is known as bytecode instructions. Bytecodes are not machine instructions and therefore in the second stage, Java interpreter generates machine code that can be directly executed by the machine that is running the Java program. Thus the Java is both a compiled and interpreted language.

Platform Independent and Portable: Java programs can be easily moved from one computer system to another, anywhere and anytime. Changes in upgrades in operating systems, processors and system resources will not force any changes in Java programs. Java ensures portability in two ways. First, Java compiler generates bytecode instructions that can be implemented on any machine. Secondly, the sizes of the data types are machine independent.

Dynamic: Java programs carry with them substantial amounts of run-time type information that is used to verify and resolve accesses to objects at run time. This makes it possible to dynamically link code in a safe and expedient manner.

Security: JVM is an interpreter which is installed in each client machine that is updated with latest **security** updates by internet . When this byte codes are executed , the JVM can take care of the **security**. So, **java** is said to be more **secure** than other programming languages.

1.2 Understand the Java Development Kit (JDK)

The JDK comes with a collection of tools that are used for developing and running Java programs which include:
_ appletviewer (for viewing Java applets)
_ javac (Java compiler)
_ java (Java interpreter)
_ javap (Java disassembler)
_ javah (for C header files)
_ javadoc (for creating HTML documents)
_ jdb (Java debugger)

Following table 1.1 lists these tools and their descriptions:

Table 1.1 : The JDK tools

Tool	Description
javac	Java compiler, which translates Java source code to bytecode files that the interpreter can understand
java	Java interpreter, which runs applets and applications by reading and interpreting bytecode files.
javadoc	Creates HTML format documentation from Java source code files.
javah	Produces header files for use with native methods.
javap	Java disassembler, which enables us to convert bytecode files into a program
jdb	Java debugger, which finds errors in programs.
applet-viewer	Enables us to run Java applets (without using a Java compatible browser)

The way these tools are applied to build and run application programs are shown below:

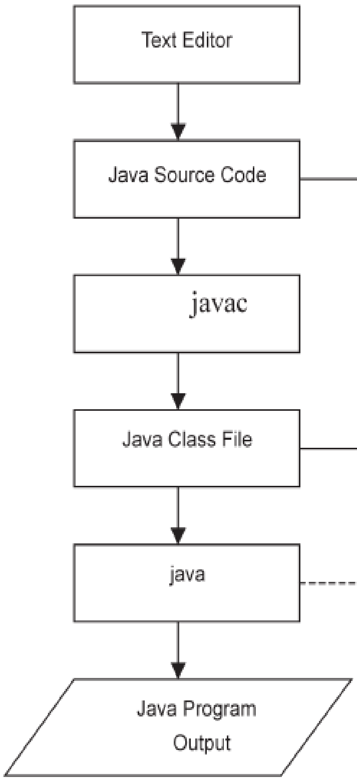


Fig 1.1. Process of building and running Java application programs.

To create a Java program it needs to create a source code file using a text editor. The source code is compiled using javac and executed using Java interpreter. The Java debugger jdb is used to find errors. A compiled Java program can be converted into a source code using Java disassembler javap.

Java Virtual Machine (JVM)

All language compilers translate source code into machine code for a specific computer. Java compiler produces an intermediate code known as bytecode for a machine that does not exist. This machine is called as Java Virtual Machine and it exists only inside the computer memory. Following figure shows the process of compiling a Java program into bytecode which is also called as Java Virtual Machine code.

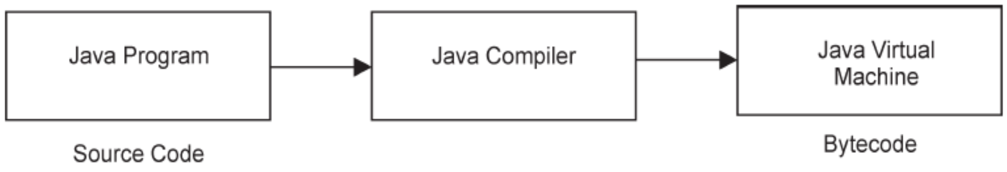


Fig. 1.2 Process of Compilation

The Java Virtual Machine code is not machine specific. The machine specific code (known as machine code) is generated by the Java interpreter by acting as an intermediate between the virtual machine and the real machine as shown in following Fig 1.3. The interpreter is different for different machines.

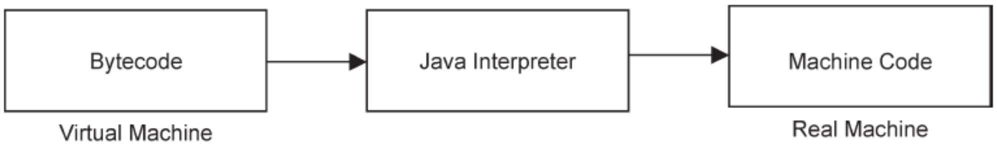


Fig 1.3. Process of converting bytecode into machine code

1.3 Write, compile and run a Java program

First Sample Program: Program to display the message “Hello World”

Aim: To write a program in Java that displays a message “Hello World”

```
/*
        This is a simple a program.
        Call this file "HelloWorld.java".
*/

        class HelloWorld{

        // program begins with a call to main()
                public static void main(String args[]){
                        System.out.println(“Hello World”);
                }
        }
```

Sample output:

```
Hello World

BUILD SUCCESSFUL (total time: 7 seconds)
```

Entering the Program

The first thing about Java is that the name given to a source file is very important. For the example given above, the name of the source file should be **HelloWorld.java**. In Java, a source file is officially called a *compilation unit.* It is a text file that contains one or more class definitions. The Java compiler requires that a source file use the **.java** file name extension.

The name of the class defined by the program is also **HelloWorld**. This is not a coincidence. In Java, all code must reside inside a class. By convention, the name of that class should match the name of the file that holds the program. It should also make sure that the capitalization of the filename matches the class name. The reason for this is that Java is case-sensitive. At this point, the convention that filenames correspond to class names may seem arbitrary. However, this convention makes it easier to maintain and organize your programs.

Compiling the program

To compile the **HelloWorld** program, execute the compiler, **javac**, specifying the name of the source file on the command line, as shown below:

```
C:\\>javac HelloWorld.java
```

The **javac** compiler creates a file called **HelloWorld.class** that contains the bytecode version of the program. As discussed earlier, the Java bytecode is the intermediate representation of program that contains instructions the Java interpreter will execute. Thus, the output of **javac** is not code that can be directly executed.

To actually run the program, the Java interpreter is used which is , called **java**. To do so, pass the class name **HelloWorld** as a command-line argument, as shown below:

```
C:\\>java HelloWorld
```

When the program is run, the following output is displayed:

When Java source code is compiled, each individual class is put into its own output file named after the class and using the **.class** extension. Therefore, it is a good idea to give the Java source files the same name as the class they contain—the name of the source file will match the name of the **.class** file. When the Java interpreter executes as just shown, by specifying the name of the class that the interpreter will execute. It will automatically search for a file by that name that has the **.class** extension. If it finds the file, it will execute the code contained in the specified class.

A Closer Look at the First Sample Program

The program begins with the following lines:

```
/*
	This is a simple Java program.
	Call this file "HelloWorld.java".
*/
```

This is a *comment.* Like most other programming languages, Java allows to enter a remark into a program's source file. The contents of a comment are ignored by the compiler. Instead, a comment describes or explains the operation of the program to anyone who is reading its source code. In this case, the comment describes the program and reminds that the source file should be called **HelloWorld.java**. In real applications, comments generally explain how some part of the program works or what a specific feature does.

Java supports three styles of comments. The one shown at the top of the program is called a *multiline comment.* This type of comment must begin with **/*** and end with ***/**. Anything between these two comment symbols is ignored by the compiler. As the name suggests, a multiline comment may be several lines long.

The next line of code in the program is shown below:

```
class HelloWorld{
```

This line uses the keyword **class** to declare that a new class is being defined. **HelloWorld** is an *identifier* that is the name of the class. The entire class definition, including all its members, will be between the opening curly brace

({) and the closing curly brace (}). The use of the curly braces in Java is identical to the way they are used in C++.

The next line in the program is the *single-line comment,* shown here:

```
//  program begins with a call to main().
```

This is the second type of comment supported by Java. A *single-line comment* begins with a // and ends at the end of the line. As a rule, programmers use multi line comments for longer remarks and single-line comments for brief, line-by-line descriptions.

The next line of code is shown here:

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

This line begins the **main( )** method. As the comment preceding it suggests, this is the line at which the program will begin executing. All Java applications begin execution by calling **main( )**. (This is just like C/C++.) Since most of the programs will use this line of code, let's take a brief look at each part.

The **public** keyword is an *access specifier,* which allows the programmer to control the visibility of class members. When a class member is preceded by **public**, then that member may be accessed by code outside the class in which it is declared. (The opposite of **public** is **private**, which prevents a member from being used by code defined outside of its class.) In this case, **main( )** must be declared as **public**, since it must be called by code outside of its class when the program is started. The keyword **static** allows **main( )** to be called without having to instantiate a particular instance of the class. This is necessary since **main( )** is called by the Java interpreter before any objects are made. The keyword **void** simply tells the compiler that **main( )** does not return a value.

As stated, **main( )** is the method called when a Java application begins. Keep in mind that Java is case-sensitive. Thus, **Main** is different from **main**. It is important to understand that the Java compiler will compile classes that do not contain a **main( )** method. But the Java interpreter has no way to run these classes. So, if Main is typed **Main** instead of **main**, the compiler would still compile your program. However, the Java interpreter would report an error because it would be unable to find the **main( )** method. Any information which is needed to pass to a method is received by variables specified within the set of parentheses that follow the name of the method. These variables are called *parameters.* If there are no parameters required for a given method, it still need to include the empty parentheses. In **main( )**, there is only one parameter, **String args[ ]** that declares a parameter named **args**, which is an array of instances of the class **String**. (*Arrays* are collections of similar objects.) Objects of type **String** store character strings. In this case, **args** receives any command-line arguments present when

the program is executed. This program does not make use of this information, but other programs may use this to enter inputs through command line arguments. The last character on the line is the **{**. This signals the start of **main( )**'s body. All of the code that comprises a method will occur between the method's opening curly brace and its closing curly brace.

NOTE: **main( )** is simply a starting place for the interpreter. A complex program will have many classes, only one of which will need to have a **main( )** method to get things started. When it begin creating applets, Java programs that are embedded in web browsers, it won't use **main( )** at all, since the web browser uses a different means of starting the execution of applets.

The next line of code is shown here. Notice that it occurs inside **main( )**.

This line outputs the string "Hello World." followed by a new line on the screen. Output is actually accomplished by the built-in **println( )** method. In this case, **println( )** displays the string which is passed to it. As seen , **println( )** can be used to display other types of information, too. The line begins with **System.out**. **System** is a predefined class that provides access to the system, and **out** is the output stream that is connected to the console.

Since most modern computing environments are windowed and graphical in nature, console I/O is used mostly for simple, utility programs and for demonstration programs. Notice that the **println( )** statement ends with a semicolon. All statements in Java end with a semicolon. The reason that the other lines in the program do not end in a semicolon is that they are not, technically, statements.

The first **}** in the program ends **main( )**, and the last **}** ends the **HelloWorld** class definition.

Second Sample Program

Program to display the area of a rectangle (Hint: area=length x breadth)

Aim: To write a program in Java to find the area of a rectangle and verify the same with various inputs(length, breadth).

Program:

```
//RectangleArea.java


//program to find area of a rectangle

class RectangleArea {
   public static void main(String args[]){
         int length,breadth;
         length=Integer.parseInt(args[0]);  //command line arguments
         breadth=Integer.parseInt(args[1]); //convert string to integer
         int area=length *breadth;
         System.out.println("length of rectangle ="+ length);
         System.out.println("breadth of rectangle ="+ breadth);
         System.out.println("area of rectangle ="+ area);
}}
```

Sample input and output:

```
C:\\>javac RectangleArea.java
C:\\> java RectangleArea 10 8
         length of rectangle = 10
         breadth of rectangle = 8
         area of rectangle = 80;

C:\\> java RectangleArea 12 15
         length of rectangle = 12
         breadth of rectangle = 15
         area of rectangle =180;
```

NOTE: The **Integer** class provides **parseInt( )** method that returns the **int** equivalent of the numeric string with which it is called. (Similar methods and classes also exist for the other data types)

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

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

1. Write and execute Java programs to do the following:

2. 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.
3. Compute all the roots of a quadratic equation using switch case statement.
    i. Hint : $x = \frac{-b\pm\sqrt{b^2-4ac}}{2a}$
4. Write a method **fact** to find the factorial of a given number.
    a. Using this method, compute $^{N}C_{R}$ in the main method.
5. Write a method isPrime to accept one integer parameter and to check whether that parameter is prime or not.
    a. Using this method, generate first N prime numbers in the main method.

Additional exercises

1. Write a method is Armstrong to check if an entered number is an Armstrong number.
2. Write a method find Sum to find the sum of digits of a number.

LAB NO: 2 **Date:**

DATA TYPES, OPERATORS AND ARRAYS

Objectives:

1. To learn the different data types in Java
2. To be familiar with bit-wise, arithmetic, Boolean, logical and relational operators
3. To/ write simple Java programs to demonstrate the usage of taking input from keyboard, data types, type conversion, and operators
4. Understand and create 1D and 2D arrays and their alternate syntax

2.1 Java data types

Java defines eight simple (or elemental) types of data: **byte**, **short**, **int**, **long**, **char**, **float**, **double**, and **boolean**. These can be put in four groups:

i) Integers: This group includes **byte**, **short**, **int**, and **long**, which are for whole valued signed numbers.

Name	**Width**	**Range**
long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	–2,147,483,648 to 2,147,483,647
short	16	– 32,768 to 32,767
byte	8	– 128 to 127

ii) Floating-point numbers: This group includes **float** and **double**, which represent numbers with fractional precision.

Name	**Width in Bits**	**Range**
double	64	1.7e–308 to 1.7e+308
float	32	3.4e–038 to 3.4e+038

iii) Characters: This group includes **char**, which represents symbols in a character set, like letters and numbers. Java uses unicode to represent characters. *Unicode* defines a fully international character set that can represent all of the characters found in all human languages. In Java **char** is a 16-bit type. The range of a **char** is 0 to 65,536. There are no negative **char**s. The standard set of characters known as ASCII ranges from 0 to 127

```
// Demonstrate char data type.
class CharDemo {
public static void main(String args[]) {
        char ch1, ch2;
        ch1 = 88; // code for X
        ch2 = 'Y';
        System.out.print("ch1 and ch2: ");
        System.out.println(ch1 + " " + ch2);
        }
}
```

This program displays the following output:

```
ch1 and ch2: X Y
```

iv) Boolean: This group includes **boolean**, which is a special type for representing true/false values. This is the type returned by all relational operators, such as **a < b**. **Boolean** is also the type *required* by the conditional expressions that govern the control statements such as **if** and **for**.

2.2 Java type conversion and casting

i) Automatic type conversion: When one type of data is assigned to another type of variable, an *automatic type conversion* will take place if the following two conditions are met:

- The two types are compatible.
- The destination type is larger than the source type.

When these two conditions are met, a *widening conversion* takes place. For example, the **int** type is always large enough to hold all valid **byte** values, so no explicit cast statement is required. For widening conversions, the numeric types, including integer and floating-point types, are compatible with each other. However, the numeric types are not compatible with **char** or **Boolean**. Also, **char** and **Boolean** are not compatible with each other.

ii) Casting incompatible types: Although the automatic type conversions are helpful, they will not fulfill all needs. For example, if it wants to assign an **int** value to a **byte** variable, the conversion will not be performed automatically, because a **byte** is smaller than an **int**. This kind of conversion is sometimes called a *narrowing conversion,* since explicitly making the value narrower so that it will fit into the target type. A *cast* is simply an explicit type conversion. It has this general form: (*target-type*) *value*

```
int a;
byte b;
// ...
b = (byte) a;
```

iii) Type promotion rules: First, all **byte** and **short** values are promoted to **int**. Then, if one operand is a **long operand**, the whole expression is promoted to **long**. If one operand is a **float** operand, the entire expression is promoted to **float**. If any of the operands is **double**, the result is **double**.It is illustrated in the below fig.2.1

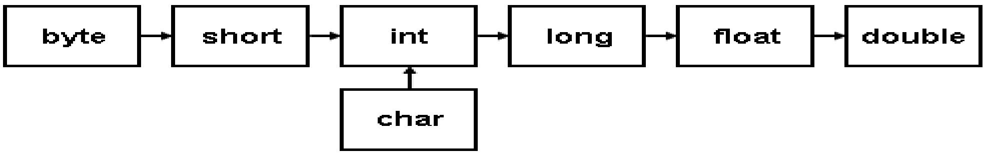


Fig 2.1. Type Promotion Rules

2.3 Short-Circuit Logical Operators

Java provides two interesting Boolean operators not found in most other computer languages. These are secondary versions of the Boolean AND and OR operators, and are known as short-circuit logical operators. As seen from the preceding table, the OR operator results in true when A is true, no matter what B is. Similarly, the AND operator results in false when A is false, no matter what B is. If operator results in false when A is false, no matter what B is. If used the | | and && forms, rather than the | and & forms of these operators, java will not bother to evaluate the right-hand operand alone. This is very useful when the right-hand operand depends on the left one being true or false in order to function properly. For example, the following code fragment shows how it can take advantage of short-circuit logical evaluation to be sure that a division operation will be valid before evaluating it:

```
if ( denom != 0 && num / denom >10)
```

Since the short-circuit form of AND (&&) is used, there is no risk of causing a run-time exception when denom is zero. If this line of code were written using the single & version of AND, both sides would have to be evaluated, causing a run-time exception when denom is zero.

It is standard practice to use the short-circuit forms of AND and OR in cases involving Boolean logic, leaving the single-character versions exclusively for bitwise operations. However, there are exceptions to this rule. For example, consider the following statement:

```
if ( c==1 & e++ < 100 ) d = 100;
```

Here, using a single & ensures that the increment operation will be applied to e whether c is equal to 1 or not.

2.4 Bit-wise operators

Java defines several *bitwise operators* which can be applied to the integer types, **long**, **int**, **short**, **char**, and **byte**. These operators act upon the individual bits of their operands. They are summarized as given below:

Operator	**Description**
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
|=	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

2.5 Reading keyboard input

Java provides Scanner class to get input from the keyboard which is present in java.util package. Therefore this package should be imported to the program. First create an object of Scanner class and then use the methods of Scanner class.

Scanner a = new Scanner(System.in);

Here “Scanner” is the class name, “a” is the name of object, “new” keyword is used to allocate the memory and “System.in” is the input stream. Following methods of Scanner class are used in the program below :-

1) nextInt to input an integer
2) nextFloat to input a float
3) nextLine to input a string
4) nextDouble to input a double

This program firstly asks the user to enter a string followed by an integer number and a float value. Immediately after entering each input, the value entered by the user will be printed on the screen.

```
import java.util.Scanner;
class GetInputFromUser{
  public static void main(String args[])   {
        int a;
        float b;
        String s;
        Scanner in = new Scanner(System.in);
        System.out.println("Enter a string");
        s = in.nextLine();
        System.out.println("You entered string "+s);
        System.out.println("Enter an integer");
        a = in.nextInt();
        System.out.println("You entered integer "+a);
        System.out.println("Enter a float");
        b = in.nextFloat();
        System.out.println("You entered float "+b);
  }
}
```

2.6 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.

i) One-dimensional arrays: A one-dimensional array is, essentially, a list of like-typed variables. The general form of a one dimensional array declaration is : *type var-name[ ];*

int month_days[]; // declares an array named month_days with the type "array of int".

Although this declaration establishes the fact that month_days is an array variable, no array actually exists. The value of month_days is set to null, which represents an array with no value. To link month_days with an actual, physical array of integers, it must allocate one using new and assign it to month_days. new is a special operator that allocates memory.

```
array-var= new type[size];
month_days = new int[12];
```

It is possible to combine the declaration of the array variable with the allocation of the array itself, as shown below:

```
int month_days[] = new int[12];
```

```
// example to know the usage of array

class AutoArray {
public static void main(String args[]) {
int month_days[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31,30, 31 };
System.out.println("April has " + month_days[3] + " days."); }}
```

ii)Multi-dimensional arrays: Multidimensional arrays are arrays of arrays. To declare a multidimensional array variable, specify each additional index using another set of square brackets. For example, the following declares a two-dimensional array variable called twoD.

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

```
// Demonstrate a two-dimensional array.

class TwoDArray {
public static void main(String args[]) {
int twoD[ ][ ]= new int[4][5];
int i, j, k = 0;
for(i=0; i<4; i++)
for(j=0; j<5; j++) {
twoD[i][j] = k;
k++;
}
for(i=0; i<4; i++) {
for(j=0; j<5; j++)
System.out.print(twoD[i][j] + " ");
System.out.println();
}
}
}
```

Ouput:

```
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
```

Alternate array declaration syntax: type[ ] var-name;

```
int al[ ] = new int[3];
int[ ] a2 = new int[3];
```

Lab exercises

1. Write a Java program to find whether a given year is leap or not using boolean data type. [Hint: leap year has 366 days;]
   Algorithm:
   ```
   if (year is not exactly divisible by 4) then (it is a common year)
   else
   if (year is not exactly divisible by 100) then (it is a leap year)
   else
   if (year is not exactly divisible by 400) then (it is a common year)
   else (it is a leap year)
   ```

2. Write a Java program to read an int number, double number and a char from keyboard and perform the following conversions:- int to byte, char to int, double to byte, double to int

3. Write a Java program to multiply and divide a number by 2 using bitwise operator. [Hint: use left shift and right shift bitwise operators]

4. Write a Java program to execute the following statements. Observe and analyze the outputs.

```
a. int x =10;                b. double x = 10.5;        c. double x=10.5;
   double y = x;                int y = x;                   int y = (int) x
   System.out.println(y);       System.out.println(y);
System.out.println(y);
```

5. Create the equivalent of a four-function calculator. The program should request the user to enter a number, an operator, and another number. (Use floating point.) It should then carry out the specified arithmetic operation: adding, subtracting, multiplying, or dividing the two numbers. Use a switch statement to select the operation. Finally, display the result. When it finishes the calculation, the program should ask if the user wants to do another calculation. The response can be 'y' or 'n'. [Hint: use do-while loop]

   Example
   ```
   Enter first number, operator, second number: 10 / 3
   Answer = 3.333333
   Do another (y/n)? n
   ```

6. Write a Java program to display non diagonal elements and find their sum. [Hint: **Non Principal diagonal**: The diagonal of a diagonal matrix from the top right to the bottom left corner is called non principal diagonal.]

7. Write a Java program to display principal diagonal elements and find their sum. [Hint: Principal Diagonal: The principal diagonal of a rectangular matrix is the diagonal which runs from the top left corner and steps down and right, until the right edge or the bottom edge is reached].

8. Find whether a given matrix is symmetric or not. [Hint: A = AT]

9. Write a program to add and multiply two integer matrices. The algorithm for matrix multiplications are give below:
   a) To multiply two matrixes sufficient and necessary condition is "number of columns in matrix A = number of rows in matrix B".
   b) Loop for each row in matrix A.
   c) Loop for each columns in matrix B and initialize output matrix C to 0.
   d) This loop will run for each rows of matrix A.
   e) Loop for each columns in matrix A.

f) Multiply A[i,k] to B[k,j] and add this value to C[i,j]
g) Return output matrix C.

Additional exercises

1. Write a Java program to find the result of the following expressions for various values of a & b:
   a. (a << 2) + (b >> 2)
   b. (b > 0)
   c. (a + b * 100) / 10
   d. a & b

2. Write a Java program to find largest and smallest among 3 numbers using ternary operator.

3. Write a Java program to execute the following statements. Observe and analyze the outputs

   a.
   ```
   boolean x =true;
   int y = x;
   ```
   b.
   ```
   boolean x =true;
   int y =(int)x;
   ```

4. Write a Java program to find whether the matrix is a magic square or not. [Hint: Compare the sum for every row, the sum with every column, the sum of the principal diagonal and the sum of the non-principal diagonal elements. If they are all same, then the matrix is a magic square matrix].
5. Print all the prime numbers in a given 1D array.
6. Find the largest and smallest element in 1D array.
7. Search for an element in a given matrix and count the number of its occurrences.
8. Write a program to merge two arrays in third array. Also sort the third array in ascending order.
9. Find the trace and norm of a given square matrix. [Hint: Trace= sum of principal diagonal elements; Norm= Sqrt(sum of squares of the individual elements of an array)]

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, aclass 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 anobject and returns a reference to it.

Consider a class Box whose object mybox is
created as followsBox mybox = new Box();
This statement combines the two steps just described. It can be rewritten like this toshow each step more clearly:
Box mybox; // declare
reference to objectmybox = new Box(); // allocate a
Box object

The steps are illustrated below:

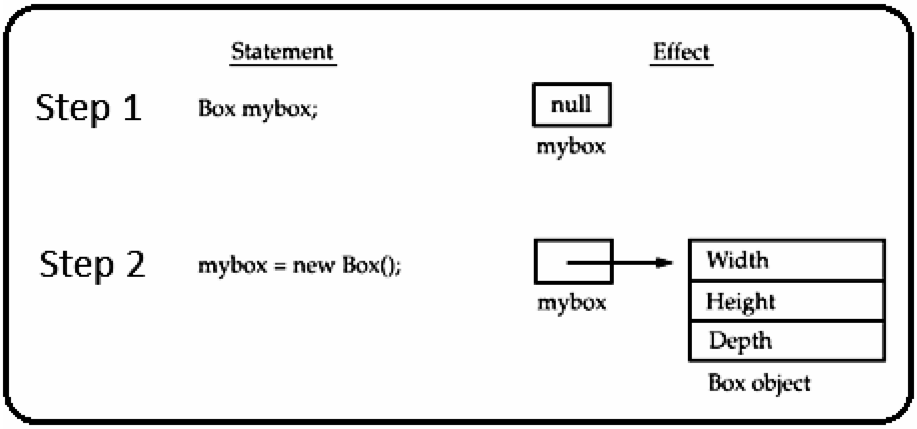


Fig 3.1 Steps to declare reference to object and allocate a box

object

Solved exercise

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

```
double width;
double
height;
double depth;
// compute and
return volumedouble
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 objectBox
mybox2 = new Box(); //allocate a Box object doublevol;

mybox1.setDim(10, 20, 15);
mybox2.setDim(3, 6, 9);

vol = mybox1.volume(); // initialize each box


// get volume of first box

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 followingmethods:
   - To assign initial values to the Complex object.
   - To display a complex number in a+ib format.
   - To add 2 complex numbers. (the return type should be Complex)
   - To subtract 2 complex numbersWrite a main method to test the class.

[Hint: Make use of Math.abs() during subtraction.]

2. Create a class called Time that has instance variables to represent hours, minutes andseconds. Provide the following methods:
   - To assign initial values to the Time object.
   - To display a Time object in the form of hh:mm:ss {24 hours format}
   - To add 2 Time objects (the return type should be a Time)
   - To subtract 2 Time objects (the return type should be a Time)
   - To compare 2 Time objects and to determine if they are equal or if the first isgreater or smaller than the second one.

3. Define a class Mixer to merge two sorted integer arrays in ascending order with thefollowing 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 withoutany duplicates

Mixer mix(Mixer A) // to merge the current object array elements

with theparameterized array elements and return the resultant object
void display() // to display the
elements of the arrayDefine 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 displaythe contents of stack.

Additional Exercises

1. The International Standard Book Number (ISBN) is a unique numeric book identifierwhich 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=$1\times1 + 2\times4 + 3\times0 + 4\times1 + 5\times6 + 6\times0 + 7\times1 + 8\times4 + 9\times9 + 10\times9 = 253$ 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 :

- If the ISBN is not a 10-digit integer, output the message "ISBN should be a 10 digit number" and terminate the program.
- If the number is 10-digit, extract the digits of the number and compute the sum asexplained 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 agetSideUp() 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. Understanding static

Introduction:

Constructor:

A constructor initializes an object when it is created. It has the same name as that of classand 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

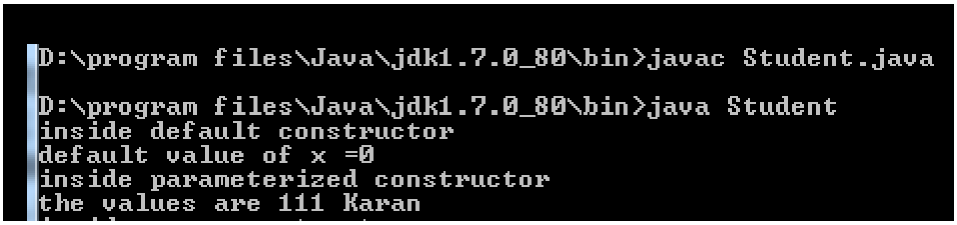
1. Program to illustrate default and parameterized constructors.

```
import java.util.*;
class Student{
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
constructorid = 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



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

D:\program files\Java\jdk1.7.0_80\bin>java Student
inside default constructor
default value of x =0
inside parameterized constructor
the values are 111 Karan
```

Static variables and methods

Normally a class member must be accessed through an object of its class, but it is possibleto 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("valis"+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 itsclass StaticMeth being created, but by using class name.

Lab Exercises:

1. Consider the already defined Complex 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 Time 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 followingmethods:
   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 5Customer 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 stringcreated 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"); thestring "This is a String, too" is a String constant.
- The objects of type String are immutable; once a String object is created, its contentscannot be altered. To change a string, create a new one that contains the modificationsor Java defines a peer class of String, called StringBuffer, which allows strings to bealtered, 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 equalityint
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]:twostr[2]: three
```

Lab Exercise:

1. Design a class which represents a student. Every student record is made up of thefollowing 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 therecords and display them.

2. Perform the following operations by adding member functions to the programimplemented in the above question
   a) Sort the student records with respect to semester and CGPA.
   b) Sort the student record with respect to name.

3. 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, Raguru Jaya Krishna must be changed to R. J. Krishna and store it in the object. Display modified objects.

4. 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-stringpresent in the main string.
2. Write and execute a Java program to count number of occurrences of a particularstring in another string.

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 super class 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

<u>Introduction:</u>

Inheritance

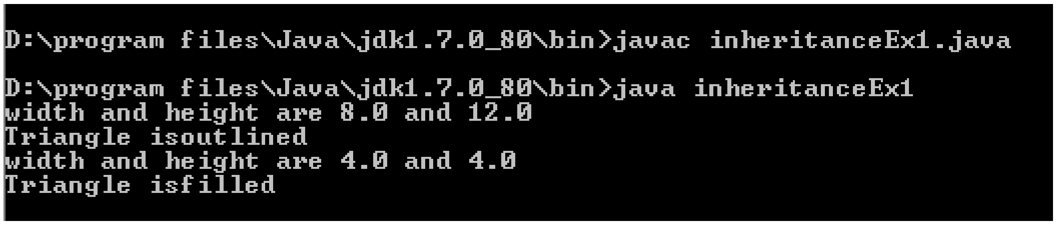
Java supports inheritance by allowing one class to incorporate another class into its declaration. This is done using the extends keyword. Thus the subclass adds (extends) tothe superclass.

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
```

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 thepackage 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.javapackage
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 beaccessible.

<u>Example of package by import package.classname</u>

```
//save by A.javapackage
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 birthdate. 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.
2. Define a class Maximum with the following overloaded methods
   - max (which finds maximum among three integers and returns the maximuminteger)
   - max (which finds maximum among three floating point numbers and returns themaximum among them)
   - max (which finds the maximum in an array and returns it)
   - 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.

3. 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 duringrun time for Rectangle, Square and Triangle to achieve dynamic polymorphism. (Usethe reference of Figure class to call the three different area methods)

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 fieldsfor 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.structure. Create a main method that declares objects of each type and uses the package.

Additional Exercises

1. 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 integerfields. Derive a class called Results which inherit from Sports and Exam. This classhas a character array or string field to represent the 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.

Develop following methods

IntSort (sort n integers in ascending order using Selection Sort)
BinSearch (performs Binary search for an element among a given list ofintegers)

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** toread an array of integers and display it. Using the methods IntSort and BinSearch frompackage **pIntegers** perform sort and search operations on the list of array elements.

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 interfacesinterface 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. Atry/catch block is placed around the code that might generate an exception.

2. 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 followsa 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 closingof 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, eithera 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 DeclareRule, extend the Exception class.
- To create a runtime exception, extend the Runtime Exception class.

3. 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 likeany 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 classcontains a withdraw() method that throws an Insufficient Funds Exception.

```
// File Name
CheckingAccount.javaimport
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.javapublic
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 andPop 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 inthe 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 dateis valid. Write a test program to illustrate the functionality.

3. Design a Student class with appropriate data members as in Lab 5. Provide your ownexceptions namely Seats Filled Exception, which is thrown when Student registrationnumber is >XX25 (where XX is last two digits of the year of joining) Show the usageof this exception handling using Student objects in the main. (Note: Registration number must be a unique number)
4. 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 Seriessuch 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. 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. 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() thatwill 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 thegrade in sports and the marks obtained from the superclass Student.

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 threadsof 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 partof 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 threadtry {
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.
```

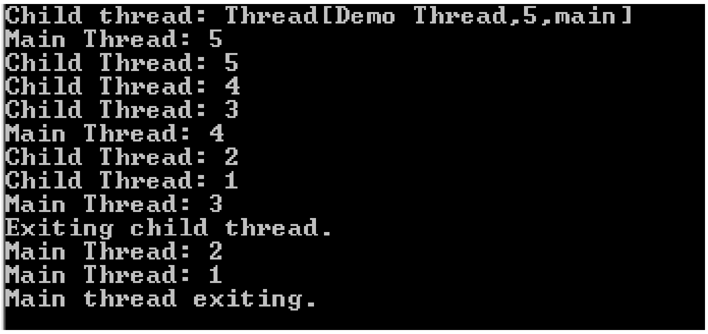
2. Program to demonstrate the creation of thread by extending Thread class:class NewThread extends Thread {

```
NewThread() {
// Create a new, second thread
super("Demo
                                                            Th
read");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 threadThread 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 threadsnew
NewThread("Two");
new NewThread("Three");try {
Thread.sleep(10000); // wait for other threads to end
}

catch (InterruptedException e) {
```

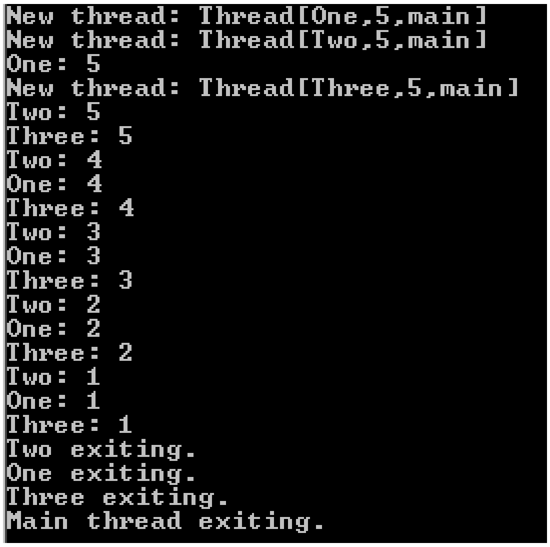
```
            System.out.println("Main thread Interrupted");

      }

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

      }

      }
```



```
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. 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 nthreads(by implementing Runnable interface) where n is equal to the number of rowsin 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.

4. Write a program to demonstrate thread priority.

Additional exercise:

1. Write and execute a java program to create five threads, the first thread checking theuniqueness of matrix elements, the second calculating row sum, the third calculatingthe 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 matrixfrom 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 isalready 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 classthat either increments or decrements the counter, depending on a parameter passed through the thread class's constructor.*

Lab No.: 9 Date:

Generics

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, whichuses 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.Integervalue: 88
Type of T is
java.lang.Stringvalue:
Generics Test
```

Here, T is the name of a type parameter. This name is used as a placeholder for the actualtype 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 shownhere:

Gen<Integer>iOb;

The type Integer is specified within the angle brackets after Gen. In this case, Integer is atype argument that is passed to Gen's type parameter, T. This effectively creates a versionof Gen in which all references to T are translated into references to Integer. Thus, for thisdeclaration, 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

Genclass:

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 twodifferent class types Student and Employee class objects.
3. Define a generic List class to implement a singly linked list and show the use of thegeneric class for two different class types Integer and Double class objects.
4. Write a program to demonstrate the use of wildcard arguments.

Additional exercises:

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

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 towhich 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 ofinput: from a disk file, a keyboard, or a network socket. Likewise, an output stream mayrefer 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 convenientmeans 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 forhandling input and output of characters. They use Unicode and, therefore, can beinternationalized. 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 =newFileReader("D:\\in put1.txt");
Out =newFileWriter("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 subdirectories usingrecursion.
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 console.
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 is found, by preceding it withits line number.
4. Write a program to copy the contents of one file to another file using FileReader andFileWriter classes.

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 Writethose 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.

Lab No.: 11 Date:

JavaFX and Event Handling -Part I

Objectives:

Objectives:

In this lab, student will be able to:

1. Understand 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 withthe 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 addthose 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 Applicationclass defines three life-cycle methods that your application can override.

1. **void init( )** - The init( ) method is called when the application begins execution. It isused 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.

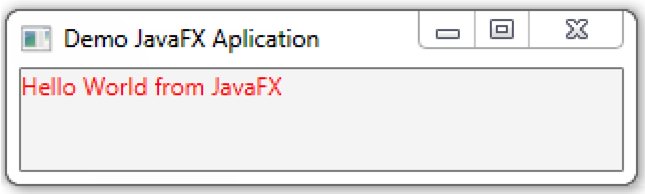
```
import javafx.scene.control.*;
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;

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:



```
Hello World from JavaFX
```

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.

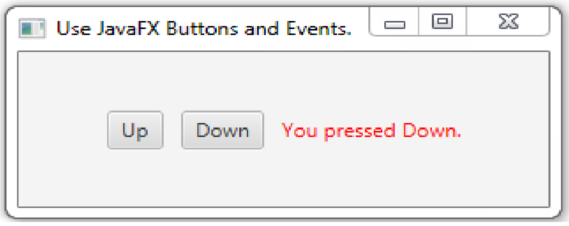
```
Import javafx.application.*;
import javafx.scene.*;
import javafx.stage.*;
import javafx.scene.layout.*;
import javafx.scene.control.*;
import javafx.event.*;
import javafx.geometry.*;
public class JavaFXEventDemo extends Application
{Label response;
public static void main(String[] args) {
// Start 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.
FlowPanerootNode = 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("Youpressed 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:



You pressed Down.

Lab exercises:

1. Write a JavaFX application program to do the following:
   a. Display the message “Welcome to JavaFX programming” using Label inthe 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 desiredvalues.
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 FlowPanelayout for the application.

3. Write a JavaFX program to display a window as shown below. Use TextField forUserName 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.

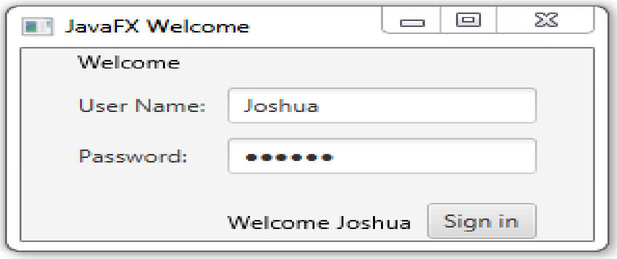


Fig 11.1 Welcome Window

4. Write a JavaFX program that obtains two positive integers passed from the userin two text fields and displays the numbers and their GCD as the result on a Label.

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 all the employee information in a JavaFX application.
2. Design a simple calculator to show the working for simple arithmetic operations.Use Grid layout.

Lab No.: 12 Date:

JavaFX and Event Handling -Part II

Objectives:

In this lab, student will be able to:

4. Explore JavaFX controls.
5. Understand 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 creationof modern, visually exciting GUIs. The JavaFX framework has all of the good features ofSwing.

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 animage 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 youcan 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 enableit.

- RadioButton: Radio buttons are a group of mutually exclusive buttons, in whichonly 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 regularpush 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 ina 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:

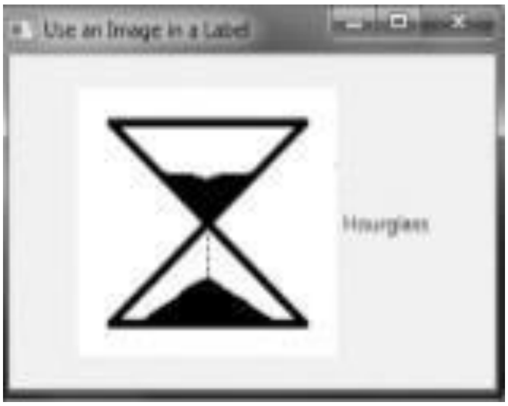


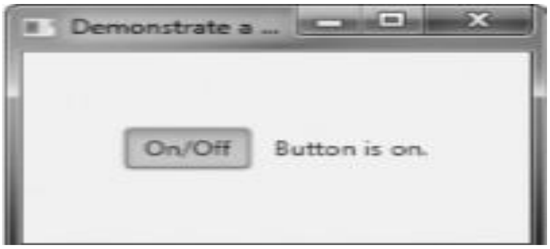
Fig 12.1 Display an image in label

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

```
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:



Button is on.

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 a Circle whose diameter is enteredby the user in a text field and calculates and displays the area, radius, diameter and circumference using Canvas based on the option chosen in List View (Area or radiusand so on).

Additional exercises:

1. Write a JavaFX program to simulate a static analog clock whose display is controlledby 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 ComprehensiveIntroduction*”, 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 notmatch 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 reservedwords 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

- **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 aredeclared 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 referencevariable can be used to refer to any object of the declared type or any compatibletype.

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 toany 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 enclosinga sequence of characters between a pair of double quotes. Examples of string literalsare:**"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 (xxxx)	Hexadecimal UNICODE character

- **Java Access Modifiers:** Java provides a number of access modifiers to set accesslevels 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