KHWAJA FAREED
UEIT
RAHIM YAR KHAN

Faculty of
Information
Technology

Laboratory Manual

Object Oriented Programming-Lab

COSC-1202

KHWAJA FAREED UEIT RAHIM YAR KHAN | Faculty of Information Technology

Table of Contents

KHWAJA FAREED UEIT RAHIM YAR KHAN | Faculty of Information Technology

JAVA INTRODUCTION

Lab-01

LAB 01	Java Introduction

Lab Objectives:

1. Compile and Run Java Program
2. Understand Java syntax

Software Required:

JDK / Text Pad/ Note Pad

BASIC SYNTAX

About Java programs, it is very important to keep in mind the following points.

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 should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case. Example: *class MyFirstJavaClass*

Method Names − All method names should 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 should 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, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match, your 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.

TASK 1: Compile and Run the Following Program

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

```
class Example {
        // Your program begins with a call to main().
        public static void main(String args[]) {
        System.out.println("This is a simple Java program.");
        }
}
```

STEPS FOR COMPILING AND EXECUTING THE JAVA PROGRAM

Let's look at how to save the file, compile, and run the program. Please follow the subsequent steps −

- Open notepad and add the code as above.
- Save the file as: MyFirstJavaProgram.java.
- Open a command prompt window and go to the directory where you saved the class. Assume it's C:\.
- Type **javac Example.java** and press enter to compile your code. If there are no errors in your code, the command prompt will take you to the next line (Assumption : The path variable is set).
- To actually run the program, you must use the Java application launcher called java. To do so, pass the class name Example as a command-line argument. Now, type **java Example** to run your program.
- You will be able to see **This is a simple Java program** printed on the window.

TASK 2: Compile and run the Following program.

```
/*
Here is another short example.
Call this file "Example2.java".
*/
class Example2 {

        public static void main(String args []) {
                int num;  // this declares a variable called num
```

```
        num = 100;  // this assigns num the value 100
        System.out.println("This is num: " + num);
        num = num * 2;
        System.out.print("The value of num * 2 is ");
        System.out.println(num);
    }
}
```

Task 3: Write a Java program to print 'Hello' on screen and then print your name on a separate line.

Task 4: Write a Java program to print the sum of two numbers.

Task 5: Write a Java program to display the following pattern.

Sample Pattern :

```
J      a    v      v    a
J     a a   v    v     a a
J    J   aaaaa  V  V    aaaaa
J J  a     a   V    a       a
```

Task 6: Write a Java program to check whether Java is installed on your computer.

```
public class Exercise31 {
public static void main(String[] args) {
    System.out.println("\nJava Version: "+System.getProperty("java.version"));
    System.out.println("Java Runtime Version: "+System.getProperty("java.runtime.version"));
    System.out.println("Java Home: "+System.getProperty("java.home"));
    System.out.println("Java Vendor: "+System.getProperty("java.vendor"));
    System.out.println("Java Vendor URL: "+System.getProperty("java.vendor.url"));
    System.out.println("Java Class Path: "+System.getProperty("java.class.path")+"\n");
 }
}
```

QUESTIONS

Please Fill the blank space with respective answers to following questions:

Question 1: What is byte code and which command is responsible for its

creation?

Question 2: Which command is responsible for the compilation of the java code?

Question 3: What will be the name and the type of the file which will be created if the following command is executed:

javac MyClass.java

Question 4: What will the following line of java code output?

System.out.print(“Pakistan is our country and we are are responsible for it”);

Question 5: What is the difference between print() and println() functions?

THE END

JAVA VARIABLES, DATA TYPES & OPERATORS

Lab-02

LAB 02	Java Variables, Datatypes and Operators

Lab Objectives:

3. Understand Java variables declaration and initialization
4. Understanding java datatypes and their usage
5. Use of operators in java

Software Required:

JDK & Notepad

The Primitive Types:

Java defines eight primitive types of data: byte, short, int, long, char, float, double, and boolean. The primitive types are also commonly referred to as simple types. These can be put in four groups:

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

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

Characters: This group includes char, which represents symbols in a character set, like letters and numbers.

Boolean: This group includes boolean, which is a special type for representing true/false values.

TASK 1: Compute distance light travels in number of days.

Write a class named as “Light”. Write main function in that class which defines variables: lightspeed, days, seconds and distance. The approximate value of “lightspeed” is 186000. Specify number of days e.g. days = 365. Now calculate distance by using following formula:

distance = lightspeed * seconds

seconds can be calculated from number of days by using following formula:

seconds = days * 24 * 60 * 60

Output of the program should be:

In 365 days light will travel about 16070400000000 miles.

Arithmetic Compound Assignment Operators

Java provides special operators that can be used to combine an arithmetic operation with an assignment. As you probably know, statements like the following are quite common in programming:

a = a + 4;

In Java, you can rewrite this statement as shown here:

a += 4;

This version uses the += *compound assignment operator*. Both statements perform the same action: they increase the value of **a** by 4.

TASK 2: Change the following program to use compound assignments:

```
class ArithmeticDemo {

    public static void main (String[] args){

        int result = 1 + 2;
        System.out.println(result);

        result = result - 1;
        System.out.println(result);

        result = result * 2;
        System.out.println(result);

        result = result / 2;
        System.out.println(result);
```

```
        result = result + 8;
        result = result % 7;
        System.out.println(result);

    }
}
```

TASK 3: Run the following program to find output.

```
class PrePostDemo {
    public static void main(String[] args){
        int i = 3;
        i++;
        System.out.println(i);
        ++i;
        System.out.println(i);
        System.out.println(++i);
        System.out.println(i++);
        System.out.println(i);
    }
}
```

TASK 4: Rewrite following expression in form of ternary expression

```
if (expression) {
   number = 10;
}
else {
   number = -10;
}
```

HINT: Variable x = (expression) ? value if true : value if false

TASK 5: (Credit Limit Calculator) Develop a Java application that determines whether any of several department-store customers has exceeded the credit limit on a charge account. For each customer, the following facts are available:

a) account number
b) balance at the beginning of the month
c) total of all items charged by the customer this month
d) total of all credits applied to the customer's account this month
e) allowed credit limit.

The program should input all these facts as integers, calculate the new balance (= beginning balance + charges – credits), display the new balance and determine whether the new balance exceeds the customer's credit limit. For those customers whose credit limit is exceeded, the program should display the message "Credit limit exceeded".

Scanner Class

- There are various ways to read input from the keyboard, the java.util.Scanner class is one of them.
- The Java Scanner class breaks the input into tokens using a delimiter that is whitespace by default. It provides many methods to read and parse various primitive values.
- Java Scanner class is widely used to parse text for string and primitive types using regular expression.

Method	Description
public String next()	it returns the next token from the scanner.
public String nextLine()	it moves the scanner position to the next line and returns the value as a string.
public byte nextByte()	it scans the next token as a byte.
public short nextShort()	it scans the next token as a short value.
public int nextInt()	it scans the next token as an int value.
public long nextLong()	it scans the next token as a long value.
public float nextFloat()	it scans the next token as a float value.
public double nextDouble()	it scans the next token as a double value.

TASK 6: Take following line as input using Scanner class and then read them using appropriate functions. Also print the data after reading

"10 tea 20 coffee 40.05 liters milk 30 tea biscuits"

TASK 7: User Input Validation: For any input, if the value entered is other than 1 or 2, keep looping until the user enters a correct value. Use Scanner class for input.

QUESTIONS

Please Fill the blank space with respective answers to following questions:

Question 1: Fill the following table

TYPE	DEFAULT VALUES
byte	
short	
int	
long	
float	
double	
char	
boolean	

Question 2: What is different between float and double datatypes?

Question 3: How can we create object of the Scanner class? Write line of code.

Question 4: What are the default values of java primitive datatypes. Fill the following table.

TYPE	DEFAULT VALUES

byte	
short	
int	
long	
float	
double	
char	
boolean	

Question 5: What is difference between i++ and ++i where i is a variable?

please explain your answer with example.

THE END

JAVA TYPE CASTING, TYPE PROMOTION & CONTROL STATEMENTS

LAB 03	Casting, Type Promotion and Control Statements

Lab Objectives:

6. Understand Java type casting and type promotion
7. Understanding java control statements

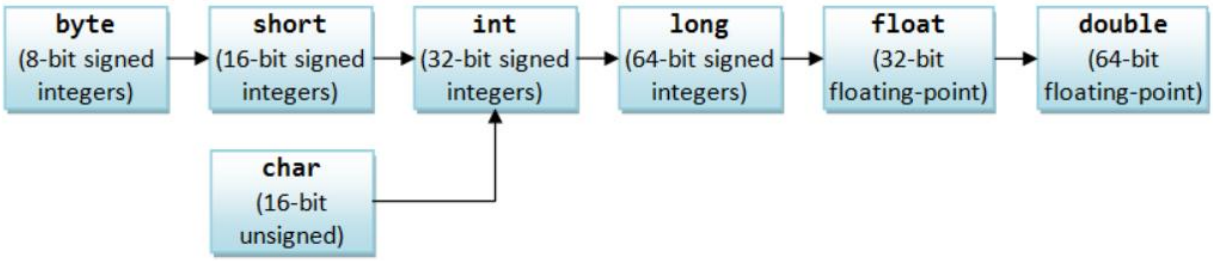
Software Required:

JDK & Notepad/ Textpad

Java's Automatic Conversions

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.



Example:

```
int i = 3;
double d;
d = i; //no Explicit type casting required
```

TASK 1: Write a program which does following conversions and prints the result. Observe the output.

```
int i = 3;
double d;
d = i;              // OK, no explicit type casting required
                    // d = 3.0
d = (double) i;  // Explicit type casting operator used here
double aDouble = 55;   // Compiler auto-casts int 55 to double 55.0
double nought = 0;      // Compiler auto-casts int 0 to double 0.0
                        // int 0 and double 0.0 are different.
```

Casting Incompatible Types

Although the automatic type conversions are helpful, they will not fulfill all needs. For example, what if you want to assign an **int** value to a **byte** variable? This 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 you are explicitly making the value narrower so that it will fit into the target type. To create a conversion between two incompatible types, you must use a cast. A *cast* is simply an explicit type conversion. It has this general form:

To create a conversion between two incompatible types, you must use a cast. A cast is simply an explicit type conversion. It has this general form:

(target-type) value

TASK 2: Implement and study the following code

```
// Demonstrate casts.
    class Conversion {
        public static void main(String args[]) {
            byte b;
            int i = 257;
            double d = 323.142;
            System.out.println("\nConversion of int to byte.");
            b = (byte) i;
            System.out.println("i and b " + i + " " + b);
            System.out.println("\nConversion of double to int.");
            i = (int) d;
            System.out.println("d and i " + d + " " + i);
            System.out.println("\nConversion of double to byte.");
            b = (byte) d;
            System.out.println("d and b " + d + " " + b);
        }
    }
```

TASK 3: Run and observe the output of following code

TASLK 4: Convert Fahrenheit to Celsius

```
class TypeCast{

        public static void main(String args[]){

                int myInt = 123;
                double myDouble = 123.0;
                String myStr = "123";
                String txtStr = "Hello, World!";

/*  It doesn't give actual value because result will be integer and it doesn't include decimal
numbers         */
System.out.println("myInt/5 gives :- " + (myInt / 5));

// Its modulus gives remainder as result
System.out.println("myInt%5 gives :- " + (myInt % 5));

/* It gives actual answer because double is decimal type and result is double */
System.out.println("myDouble/5 gives :- " + (myDouble / 5));

// Another type-cast by forcing double math
System.out.println("myInt/5.0 gives :- " + (myInt / 5.0));

// Attemp to convert string to int but it gives error
//System.out.println("myStr to int :- " + (int)myStr);

/* java has a Integer class which has a method parseInt it to convert string to integer */
System.out.println("Converting string to int :- " + Integer.parseInt(myStr));

//        converting a text string to int gives runtime error
//System.out.println("Converting text string to int :- " + Integer.parseInt(txtStr));

}
}
```

C= (F-32.0)*(5/9);

TASK 5: Write a Java program to compute body mass index (BMI).

Formula: BMI = weight * 0.45359237 / (inches * 0.0254 * inches * 0.0254)

Test Data

Input weight in pounds: 452

Input height in inches: 72

Expected Output:

Body Mass Index is 61.30159143458721

Automatic Type Promotion in Expressions

In addition to assignments, there is another place where certain type conversions may occur: in expressions. To see why, consider the following. In an expression, the precision required of an intermediate value will sometimes exceed the range of either operand. For example, examine the following expression:

```
byte a = 40;
byte b = 50;
byte c = 100;
int d = a * b / c;
```

The result of the intermediate term a * b easily exceeds the range of either of its byte operands. To handle this kind of problem, Java automatically promotes each byte, short, or char operand to int when evaluating an expression.

TASK 6: Implement and study the following program.

```
class Promote {
        public static void main(String args[]) {
                byte b = 42;
                char c = 'a';
                short s = 1024;
                int i = 50000;
                float f = 5.67f;
                double d = .1234;
                …… result = (f * b) + (i / c) - (d * s); //identify the datatype of result
                System.out.println((f * b) + " + " + (i / c) + " - " + (d * s));
                System.out.println("result = " + result);
        }
}
```

TASK 7: Program *Rock.java* contains a skeleton for the game Rock, Paper, Scissors. Open it and save it to your directory. Add statements to the

program as indicated by the comments so that the program asks the user to enter a play, generates a random play for the computer, compares them and announces the winner (and why). For example, one run of your program might look like this:

$ java Rock

Enter your play: R, P, or S

r

Computer play is S

Rock crushes scissors, you win!

Use a switch statement to convert the randomly generated integer for the computer's play to a string.

```
// ****************************************************************
//   Rock.java
//
//   Play Rock, Paper, Scissors with the user
//
// ****************************************************************
import java.util.Scanner;
import java.util.Random;
public class Rock
{
    public static void main(String[] args)
    {
      String personPlay;    //User's play -- "R", "P", or "S"
      String computerPlay; //Computer's play -- "R", "P", or "S"
      int computerInt;      //Randomly generated number used to determine
                            //computer's play
      Scanner scan = new Scanner(System.in);
      Random generator = new Random();
      //Get player's play -- note that this is stored as a string
      //Make player's play uppercase for ease of comparison
      //Generate computer's play (0,1,2)
```

```
        //Translate computer's randomly generated play to string
        switch (computerInt)
        {
        }
        //Print computer's play
        //See who won. Use nested ifs instead of &&.
        if (personPlay.equals(computerPlay))
            System.out.println("It's a tie!");
        else if (personPlay.equals("R"))
            if (computerPlay.equals("S"))
              System.out.println("Rock crushes scissors. You win!!");
            else
              //... Fill in rest of code
    }
}
```

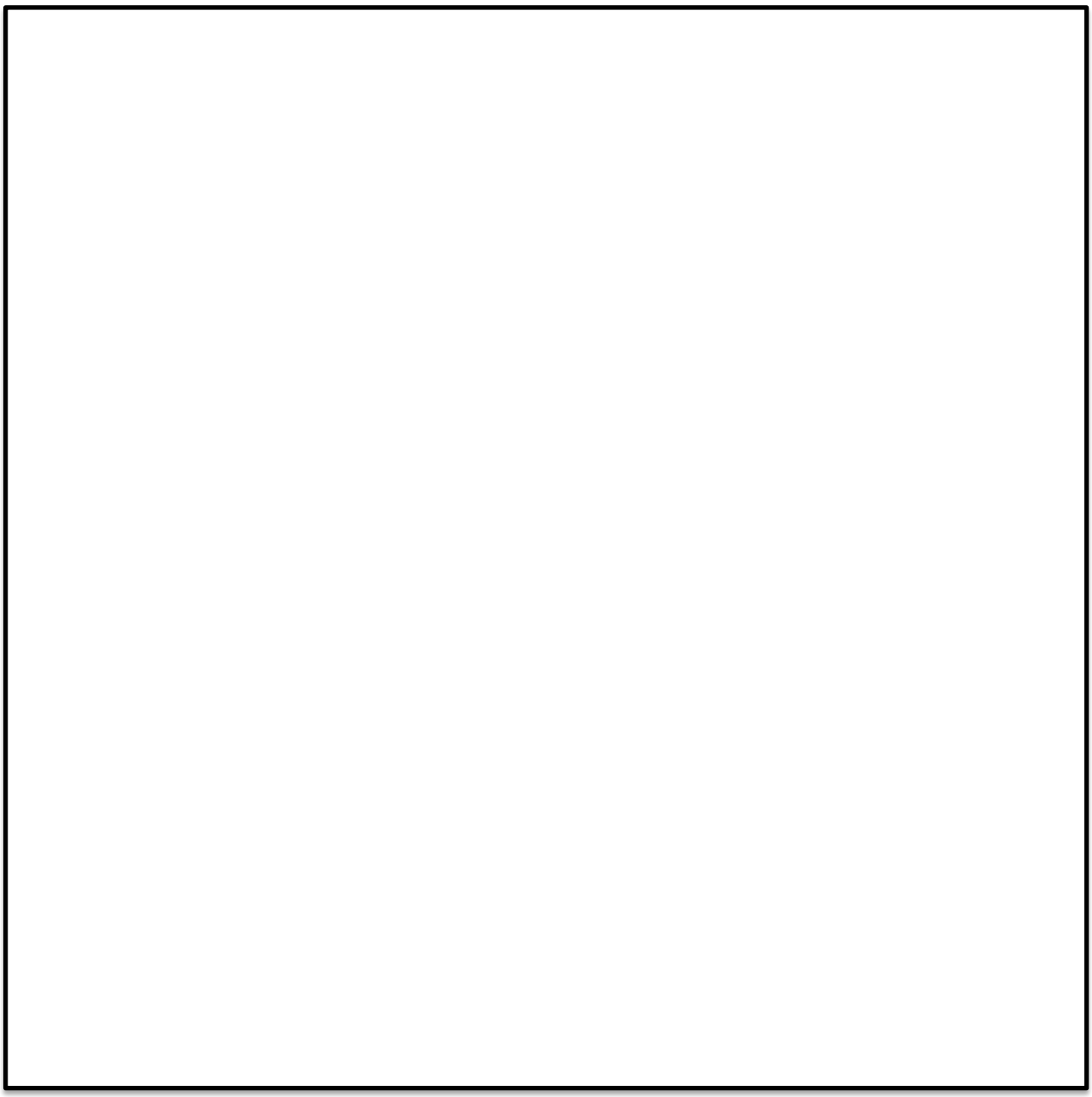
QUESTIONS

Please Fill the blank space with respective answers to following questions:

Question 1: What is different between implicit and explicit type casting?

Question 2: Explain each output in TASK 3 one by one in your own words.

Question 3: Explain type promotion in your own words



Question 4: Draw Flow chart for rock paper scissors game.

THE END

JAVA LOOPS

Lab-04

LAB 04	Loops in Java

Lab Objectives:

1. Understand Java loops and their usage
2. Implement all types of loops
3. Understand practical usage of loops

Software Required:

JDK & Notepad/ Textpad

Question 1: The exponent tells you how many times to multiply the number by itself. For instance, three to the power of four, or 3^4, will be: 3 x 3 x 3 x 3 = 9 x 9 = 81. Write a program for calculating power of a number using loop.

Question 2: Implement question 1 using while loop.

Question 3: Write a program called CozaLozaWoza which prints the numbers 1 to 110, 11 numbers per line. The program shall print "Coza" in place of the numbers which are multiples of 3, "Loza" for multiples of 5, "Woza" for multiples of 7, "CozaLoza" for multiples of 3 and 5, and so on. The output shall look like:

1 2 Coza 4 Loza Coza Woza 8 Coza Loza 11 Coza 13 Woza CozaLoza 16 17 Coza 19 Loza CozaWoza 22 23 Coza Loza 26 Coza Woza 29 CozaLoza 31 32 Coza .....

Question 4: Write a program which shows following output using nested loops

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

Question 5: Ask user input to input values and check if value is multiple of 3. If the given value is multiple of 3 then print "Yes! you reached end" otherwise keep on looping and asking user to enter new number.

QUESTIONS

Please Fill the blank space with respective answers to following questions:

_____

_____

_____

Question 1: What is the difference between do while and while loop?

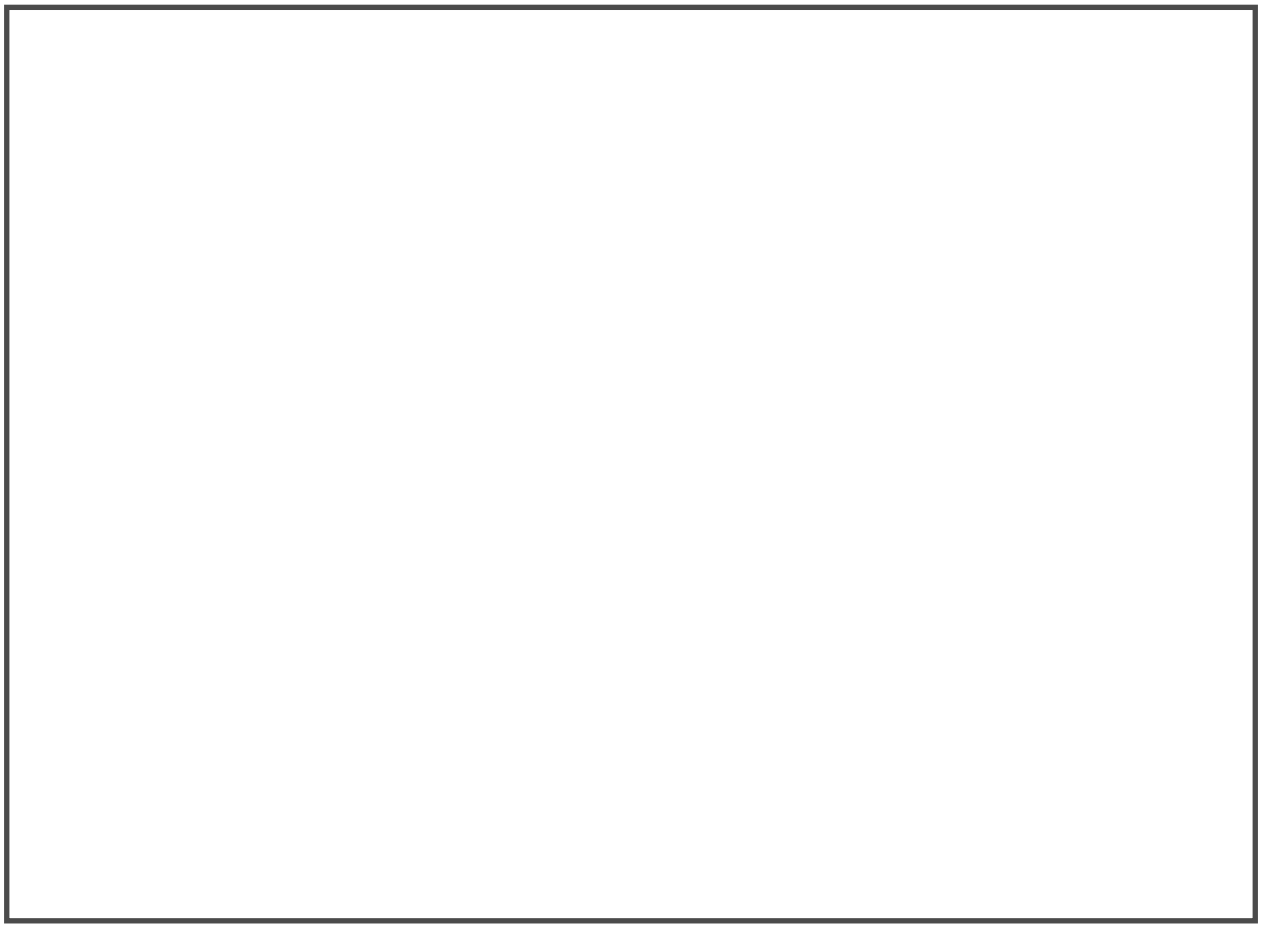
Question 2: What will be the output of the following program?

```
class ForSample
{
   public static void main(String s[])
   {
      for(int i = 0; i <= 5; i++ )
      {
```

```
            System.out.println("i = " + i );
        }
        System.out.println("i after the loop = " + i );
    }
}
```

Question 3: What will be the output of the following program?

```
class ForSample
{
    public static void main(String s[])
    {
        int i = 0;
        for(;i <= 5; i++ )
        {
            System.out.println("i = " + i );
        }
        System.out.println("i after the loop = " + i );
    }
```



}

Question 4: What will be output of following program. Explain your answer.

```
class Output
{
   public static void main(String s[])
   {
      int i = 1, j = 10;
      do
      {
         if(i > j)
            break;
         j--;
         } while (++i < 5);
         System.out.println("i = " + i + " and j = " + j);
```

}

}

Explain Here:

THE END

JAVA ARRAYS

Lab-05

LAB 05	Arrays in Java

Lab Objectives:

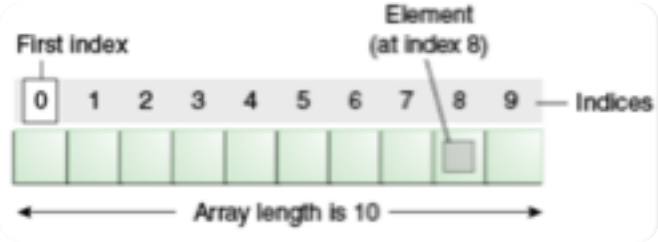
1. Understanding arrays and their implementation in java
2. Manipulating arrays in java

Software Required:

JDK & Notepad/ Textpad

ARRAYS

An *array* is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created.



Question 1:

Describe and explain what happens when you try to compile a program HugeArray.java with the following statement:

```
int n = 1000;
int[] a = new int[n*n*n*n];
```

Question 2:

Set up an array to hold the following values, and in this order: 23, 6, 47, 35, 2, 14. Write a program to get the average of all 6 numbers. Use foreach loop.

Question 3:

Using the above values in Question 5, have your program print out the highest and lowest numbers in the array. Use foreach loop.

Question 4:

Use a one-dimensional array to solve the following problem: Write a program that inputs 5 numbers, each of which is between 10 and 100, inclusive. As each number is read, display it

only if it is not a duplicate of a number already read. Provide for the "worst case," in which all 5 numbers are different. Use the smallest possible array to solve this problem.

Question 5:

Write a code which calculates transpose of 2x3 matrix.

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \end{bmatrix} \quad \text{->} \quad M = \begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 3 & 5 \end{bmatrix}$$

QUESTIONS

Please Fill the blank space with respective answers to following questions:

Question 1: What will be the last element of an array whose size is 24?

Question 2: What will be the output of the following program?

```
public class DemoOnLong
{
   public static void main(String[] args)
   {
      long[][] input = new long[3][];
      input[0] = new long[2];
      input[1] = new long[3];
      input[2] = new long[5];
      input[0][1] = 12L;
      System.out.println(input[0][1]);
   }
```

```

```

}

Question 3: Write a program to multiply every array element with 3.

```
class MultiplyEveryArrayElement
{    public static void main(String s[])
   {
      int[] input = { 3, 5, 6, 7 };
      int[] output = multiplyEveryElement(input);
      System.out.print("Result of multiplying every element by 3 is : ");
      for(int outputElement : output)
      {
         System.out.print(outputElement + ", ");
      }

   }

    public static int[] multiplyEveryElement(int[] input)
    {
         int[] result = null;
         //Write code of this part in the box below to multiply every element with 3 and assign
         it to result.
         return result;
    }
}
```

THE END

JAVA METHODS, CONSTRUCTORS AND METHOD OVERLOADING

Lab-06

LAB 06	Methods, Constructors and Method Overloading

Lab Objectives:

1. Understand java methods, arguments passing and return
2. Understand the purpose and implementation of constructors
3. Understand and implement the mechanism of method overloading

Software Required:

JDK & Notepad/ Textpad

Introduction:

A simplified general form of a **class** definition is shown here:

class *classname* {

type instance-variable1;

type instance-variable2;

// ...

type instance-variableN;

type m

ethodname1(*parameter-list*) {// body of method

}

type methodname2(*parameter-list*) {

// body of method

}

// ...

type methodnameN(*parameter-list*) {

// body of method

}

}

Sample code:

```
/* Here, Box uses a parameterized constructor to initialize the dimensions of a box. */
    class Box {
        double width;
        double height;
```

```
        double depth;

        // This is the constructor for Box.
        Box (double w, double h, double d) {
                width = w;
                height = h;
                depth = d;
        }
        // compute and return volume
        double volume() {
                return width * height * depth;
        }
}

class BoxDemo7 {

        public static void main(String args[]) {
                        // declare, allocate, and initialize Box objects
                        Box mybox1 = new Box(10, 20, 15);
                        Box mybox2 = new Box(3, 6, 9);
                        double vol;

                        // get volume of first box
                        vol = mybox1.volume();
                        System.out.println("Volume is " + vol);

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

Practice Problems:

Task 1: Swap Variables

- Declare a class swapDemo
- Create two variable for holding values of x and y as 10 and 100 respectively. Initialize using constructor.
- Write a method swapVar which swaps the two digits.
- Create object of class swap
- Call class swap method swapVar and pass x and y as arguments.
- Print both original and swapped values.

Task 2: You will create a class that keeps track of the total cost, average cost, and number of items in a shopping bag.

Create a class called ShoppingBag. Objects of this class represent a single shopping bag. Attributes of such an object include the number of items in the bag and the total retail cost of those items.

Provide a constructor that accepts a tax rate as a float parameter.

Provide a transformer method called place that accepts an int parameter indicating the number of the particular items that are being placed in the bag and a float parameter that indicates the cost of each of the items. For example, myBag.place (5, 10. 5); represents placing 5 items that cost $10.50 each into myBag.

Provide getter methods for the number of items in the bag and their total retail cost. Provide a totalCost method that returns the total cost with tax included.

Provide at oString method that returns a nicely formatted string that summarizes the current status of the shopping bag.

Finally, provide a program, a “test driver,” that demonstrates that your ShoppingBag class performs correctly.

Method Overloading

Methods of the same name can be declared in the same class, as long as they have different sets of parameters (determined by the number, types and order of the parameters)—this is called method overloading.

Math Class Overloaded Methods

S.N.	Method & Description
1	static double abs(double a) This method returns the absolute value of a double value.
2	static float abs(float a) This method returns the absolute value of a float value.
3	static int abs(int a) This method returns the absolute value of an int value.
4	static long abs(long a) This method returns the absolute value of a long value.
5	static double copySign(double magnitude, double sign) This method returns the first floating-point argument with the sign of the second floating-point argument.
6	static float copySign(float magnitude, float sign) This method returns the first floating-point argument with the sign of the second floating-point argument.
7	static int getExponent(double d) This method returns the unbiased exponent used in the representation of a double.
8	static int getExponent(float f) This method returns the unbiased exponent used in the representation of a float.
9	static double max(double a, double b) This method returns the greater of two double values.
10	static float max(float a, float b) This method returns the greater of two float values.
11	static int max(int a, int b) This method returns the greater of two int values.

12	static long max(long a, long b) This method returns the greater of two long values.
13	static double min(double a, double b) This method returns the smaller of two double values.
14	static float min(float a, float b) This method returns the smaller of two float values.
15	static int min(int a, int b) This method returns the smaller of two int values.
16	static long min(long a, long b) This method returns the smaller of two long values.
17	static double nextAfter(double start, double direction) This method returns the floating-point number adjacent to the first argument in the direction of the second argument.
18	static float nextAfter(float start, double direction) This method returns the floating-point number adjacent to the first argument in the direction of the second argument.
19	static double nextUp(double d) This method returns the floating-point value adjacent to d in the direction of positive infinity.
20	static float nextUp(float f) This method returns the floating-point value adjacent to f in the direction of positive infinity.
21	static int round(float a) This method returns the closest int to the argument.
22	static double scalb(double d, int scaleFactor) This method returns $d \times 2^{scaleFactor}$ rounded as if performed by a single correctly rounded floating-point multiply to a member of the double value set.
23	static float scalb(float f, int scaleFactor) This method return $f \times 2^{scaleFactor}$ rounded as if performed by a single correctly rounded floating-point multiply to a member of the float value set.
24	static double signum(double d) This method returns the signum function of the argument; zero if the argument is zero, 1.0 if the argument is greater than zero, -1.0 if the argument is less than zero.

25	static float signum(float f) This method returns the signum function of the argument; zero if the argument is zero, 1.0f if the argument is greater than zero, -1.0f if the argument is less than zero.
26	static double ulp(double d) This method returns the size of an ulp of the argument.
27	static double ulp(float f) This method returns the size of an ulp of the argument.

Task 3: Create a class named as DemoOverloading. Now create following methods in the class:

Method named as test() which takes no parameters and returns nothing.

Method named as test() which takes one integer parameter and return nothing.

Method named as test() which takes two integer parameters and returns nothing.

Method named as test() which takes two integer parameters and return their integer type sum.

Method test() which takes one double parameter and returns double type value.

Method test() which takes two argument first one is integer and second one is float and returns nothing.

Method test() which takes two arguments, first one is float and second one is integer and return nothing.

Now Create another class named as demo and call each method of class DemoOverloading, one by one by passing appropriate parameters.

Task 4: Write a method minimum3 that returns the smallest of three floating-point numbers. Use the Math.min method to implement minimum3. Incorporate the method into an application that reads three values from the user, determines the smallest value and displays the result.

Task 5: Modify the program in TASK 4 to compute the minimum of four double values.

QUESTIONS

Please Fill the blank space with respective answers to following questions:

_____

_____

Question 1: What is the purpose of creating constructors?

Question 2: Suppose there is a class name "MyClass". How will its object

_____

_____

be created?

_____

_____

Question 3: When is a constructor called?

Question 4: What is a default constructor? Explain the difference between

_____

_____

_____

_____

default constructor and parametrized constructor.

Question 5: What is wrong with following code?

```
class Student
{
   String name;
   int marks;
   char section;
}
```

Question 6: What are the three rules on which a method can be overloaded?

THE END

KHWAJA FAREED UEIT RAHIM YAR KHAN | Faculty of Information Technology

JAVA ACCESS MODIFIERS, THIS & STATIC KEYWORDS

Lab-07

LAB 07	Access Modifiers, *this* keyword & *static* keyword

Lab Objectives:

1. Understanding the use of *this* keyword
2. Understanding access modifiers
3. Understanding and implementing *static* keyword concept

Software Required:

Notepad ++ or Textpad

Access Modifiers

Access levels and their accessibility can be summarized as:

Private: Same class.

Protected: Same class, Same package, Subclasses.

Public: Same class, Same package, Subclasses, Everyone.

Task 1: Write another class in which you write main method and access the

```
/* This program demonstrates the difference between public and private. */

class Test
{
	int a; // default access
	public int b; // public access
	private int c; // private access

	// methods to access c
	void setc(int i) { // set c's value
		c = i;
	}

	int getc() { // get c's value
		return c;
	}
}
```

attributes of following class.

Task 2: Write constructor of the above class (Test.java) and declare it private. Now try to create object of the above class in another class.

```
package pack1;

class X
{
    protected int i = 1221;

    void methodOfX()
    {
        System.out.println(i);
    }
}

public class MainClass
{
    public static void main(String[] args)
    {
        X x = new X();

        System.out.println(x.i);

        x.methodOfX();
    }
}
```

Task 3: Run following code. Is it written correctly?

this Keyword

There can be a lot of usage of **java this keyword**. In java, this is a **reference variable** that refers to the current object.

- refer current class instance variable.
- invoke current class method (implicitly)
- invoke current class constructor.
- can be passed as an argument in the method call.
- can be passed as argument in the constructor call.
- can be used to return the current class instance from the method.

Sample code

```
class A{
A(){System.out.println("hello a");}
A(int x){
this();
System.out.println(x);
}
```

```
}
class TestThis5{
public static void main(String args[]){
A a=new A(10);
}}
```

OUTPUT:

```
hello a
10
```

Task 4: *this*: to refer current class instance variable. Write a class in which names of class data members are same as constructor arguments.

Definition of the constructor should be similar to following

```
Student(int rollno,String name,float fee){
rollno=rollno;
name=name;
fee=fee;
}
```

Display rollno, name and fee values in display method

Now create two objects of this class and display their value

Observe output and identify problem

Now rewrite constructor function as follows

```
Student(int rollno,String name,float fee){
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
```

again run the code and see output

program in which this keyword is not required

Task 5: *this*: to invoke current class constructor

The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

Reuse of constructor from constructor

1. Create a class names as Student
2. create a constructor of this class which takes rollno, course and name as parameter and assign them to variables of rollno, course and name respectively
3. Create another constructor which takes arguments: rollno, course, name and fee. Assign fee to fee variable and call first constructor to assign other three arguments.
4. Call class using three and four parameters.

static Keyword

A static method belongs to the class rather than object of a class.

A static method can be invoked without the need for creating an instance of a class.

Static method can access static data member and can change the value of it.

If you declare any variable as *static*, it is known *static* variable.

The *static* variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees, college name of students etc.

The *static* variable gets memory only once in class area at the time of class loading.

Task 6:

A. Calculate cube using a method

- Create a class named as calculate
- Write a method, count in which returns cube of the given variable.
- Write main method which calls the count method

```
class Counter{
int count=0;//will get memory when instance is created

public IncCounter(){
count++;
System.out.println(count);
}

public static void main(String args[]){

Counter c1=new Counter();
c1.IncCounter();
Counter c2=new Counter();
c2.IncCounter();
Counter c3=new Counter();
c3.IncCounter();
 }
}
```

B. Calculate cube using *static* method

Now change count method to *static* by adding *static* keyword

Call count method from main to calculate cube

Task 7: Implement following class

Now change the count variable to static then observe output.

QUESTIONS

Please Fill the blank space with respective answers to following questions:

Question 1: Can the constructor of a class be declared private? What will be consequences?

Question 2: What is the difference between private and protected data members of a class?

Question 3: What is 'default' access modifier?

Question 4: Write the content of the constructor whose parameter names are same as the member variables name.

```
class ClassConstructorOperations
{
    public static void main(String s[])
   {
      Student student = new Student( "Lavanya", 501, 95.6 );
      System.out.println(student.rollNumber + " is the roll number of " + student.name + "
and she secured " + student.percentage + " % in SSC" );
   }
}
class Student
{
    String name;
```

```
    int rollNumber;
    double percentage;
    //Write a constructor here to initialize the parameters.
```

```
}
```

Question 5: What will be the output of the program?

```
public class Karbon {
   static int i = 5;
   public static void main(String[] args) {
      Karbon test = null;
      System.out.print("i = " + test.i + ", ");
      System.out.print("i = " + Karbon.i);
```

```
   }}
```

THE END

KHWAJA FAREED UEIT RAHIM YAR KHAN | Faculty of Information Technology

JAVA FINALIZE, INNER & NESTED CLASSES

Lab-08

LAB 08	Finalize, Nested & Inner Classes

Lab Objectives:

1. Understanding the concept of garbage collection
2. Understanding and implementing Nested and Inner classes
3. Understanding and Implementing Encapsulation

Software Required:

JDK & Notepad/ Textpad

The finalize( ) Method

Sometimes an object will need to perform some action when it is destroyed. For example, if an object is holding some non-Java resource such as a file handle or character font, then you might want to make sure these resources are freed before an object is destroyed. To handle such situations, Java provides a mechanism called finalization. By using finalization, you can define specific actions that will occur when an object is just about to be reclaimed by the garbage collector.

TASK 1: See Output of the following Program

```
class A {
int i = 50;
@Overrideprotected void finalize() throws Throwable {
System.out.println("From Finalize Method"); }

}

public class Test
{
   public static void main(String[] args)
   {
      //Creating two instances of class A
```

```
            A a1 = new A();
            A a2 = new A();
            //Assigning a2 to a1
            a1 = a2;
            //Now both a1 and a2 will be pointing to same
object
             //An object earlier referred by a1 will become

             //abandoned

             System.gc();System.out.println("done");

             }

        }
```

final Keyword: The final keyword can be applied with the variables, a final variable that has no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only.

TASK 2: Value of final variable cannot be changed once declared. Run the following code and check output.

```
class Bike9{
        final int speedlimit=90;//final variable
        void run(){
        speedlimit=400;
        }
        public static void main(String args[])
        {
         Bike9obj=new Bike9(); obj.run();
        }
}//end of class
```

TASK 3: Try to initialize value of final variable in a constructorTASK 4: Do not initialize final variable on declaration and try to initialize it anywhere other than constructor.

TASK 4: Declare final variable as static and try to initialize its value in a method. Then create a static block and initialize the static final variable value.

Nested and Inner Classes

TASK 5: Run the following code and try to understand its working

```
// Demonstrate an inner class.
class Outer {
      int outer_x = 100;
            void test() {
            Inner inner = new Inner();
            inner.display();
            }
      // this is an inner class
      class Inner {
        void display() {        System.out.println("display:
outer_x = " + outer_x); }

    }

}
class InnerClassDemo {
      public static void main(String args[]) {
            Outer outer = new Outer();
            outer.test();
} }
```

TASK 6: Declare a local variable in inner class and try to display it in outer class.

TASK 7: Declare a private variable in outer class and try to display it in inner class.

TASK 8: Try to declare inner class within a method of outer class.

TASK 9: Run and Learn from following code:

```
public class Outer {
   static class Nested_Demo {
public void my_method() {

 System.out.println("This is my nested class");

} }

public static void main(String args[])

{

Outer.Nested_Demo nested = new Outer.Nested_Demo();
nested.my_method();

} }
```

QUESTIONS

Please Fill the blank space with respective answers to following questions:

Question 1: What is the purpose of garbage collector in java?

Question 2: If the final variable is initialized at the time of declaration then

_____

_____

can it be re-initialized in constructor?

Question 3: Can final variable be initialized somewhere other then

_____

_____

constructor?

Question 4: What will be the output of the program?

```
class Output
{
   final static short i = 2;
   public static int j = 0;
   public static void main(String [] args)
   {
      for (int k = 0; k < 3; k++)
      {
         switch (k)
         {
            case i: System.out.print(" 0 ");
            case i-1: System.out.print(" 1 ");
            case i-2: System.out.print(" 2 ");
         }
```

```
    }
  }
}
```

Question 5: What will be the output of the following program?

```
public class Final
{
  final int assign = 30;
  public static void main(String[] args)
  {
    final int result = 20;
    final int assign;
    Final f = new Final();
    assign = 20;
    System.out.println(assign);
    System.out.println(f.assign);
    System.out.println(f.process(result));
  }
  int process(int a)
  {
    return a + 5;
  }
```

```
}
```

THE END

KHWAJA FAREED UEIT RAHIM YAR KHAN | Faculty of Information Technology

JAVA INHERITANCE

Lab-09

LAB 09	Inheritance in Java

Lab Objectives:

1. Understanding the concept of inheritance
2. Implementation of Inheritance in java

Software Required:

Netbeans IDE

Inheritance in Java

TASK 1: Run and understand the following code

```
// A simple example of inheritance.
// Create a superclass.
clas
      s A {int i, j;
            void showij() {
            System.out.println("i and j: " + i + " " + j);
            }
}
// Create a subclass by extending class A.
class B extends A {
      int k;
            void showk() {
            System.out.println("k: " + k);
            }
            void sum() {
            System.out.println("i+j+k: " + (i+j+k));
                  }
}

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

```
        // The superclass may be used by itself.
        superOb.i = 10;
        superOb.j = 20;
        System.out.println("Contents of superOb: ");
        superOb.showij();
        System.out.println();

        /* The subclass has access to all public members of
        its superclass. */
        subOb.i = 7;
        subOb.j = 8;
        subOb.k = 9;
        System.out.println("Contents of subOb: ");
        subOb.showij();
        subOb.showk();
        System.out.println();
        System.out.println("Sum of i, j and k in subOb:");
        subOb.sum();
        }
}
```

TASK 2: Create a class A which has two data members one is public and other is private. Create another class B which extends class A. Try to display values of both data members of class A in class B.

TASK 3: Continuing with TASK 2. Display some statements in the constructors of both class A and class B and now try to create object of class B to observe which constructor was called first.

TASK 4: Imagine a publishing company that markets both book and audiocassette versions of its works. Create a class publication that stores the title (a string) and price (type float) of a publication. From this class derive two classes: book, which adds a page count (type int), and tape, which adds a playing time in minutes (type float). Each of these three

classes should have a getdata() function to get its data from the user at the keyboard, and a putdata() function to display its data.

Write a main() program to test the book and tape classes by creating instances of them, asking the user to fill in data with getdata(), and then displaying the data with putdata().

TASK 5: Drive a class called RegularEmployee and HourlyEmploee from the Employee class. RegularEmployee should add a type double data item called basicSalary. HourlyEmploee should add a type double data item called hourlyRate and another double data type item called noOfHours to calculate salary for hours worked. In both classes add appropriate functions to take their information from user and display it as well.

QUESTIONS

Please Fill the blank space with respective answers to following questions:

Question 1: What are different types of inheritance?

Question 2: Can child class inherit private data of parent class?

Question 3: What is the sequence of constructor calling in multi level

inheritance? Suppose A is parent of B and B is parent of class C.

Question 4: Define the required classes such that the program compiles without any errors.

```
class DefineClassHierarchy
{    public static void main(String s[])
```

```
{
    C c = new C();
    M m = new M();
    K k;
    O o = new C();
    H h = new H();
    k = o;
    m = h;
    c = m;
    System.out.println("All compilation errors resolved");
}
```

}

THE END

JAVA SUPER KEYWORD & POLYMORPHISM

Lab-10

LAB 10	*Super* Keyword and Polymorphism in Java

Lab Objectives:

1. Understanding the concept of polymorphism
2. Understand the super keyword in inheritance

Software Required:

Netbeans IDE

super Keyword in Java

Sample Code:

```
class Box {
        private double width;
        private double height;
        private double depth;
        // construct clone of an object
        Box(Box ob) { // pass object to constructor
                width = ob.width;
                height = ob.height;
                depth = ob.depth;
        }
        // constructor used when all dimensions specified
        Box(double w, double h, double d) {
                width = w;
                height = h;
                depth = d;
        }
        // constructor used when no dimensions specified
        Box() {
                width = -1; // use -1 to indicate
                height = -1; // an uninitialized
                depth = -1; // box
        }
```

```
        // constructor used when cube is created
        Box(double len) {
                width = height = depth = len;
        }
        // compute and return volume
        double volume() {
                return width * height * depth;
        }
}

// BoxWeight now fully implements all constructors.
class BoxWeight extends Box {
        double weight; // weight of box
        // construct clone of an object
        BoxWeight(BoxWeight ob) { // pass object to constructor
                super(ob);
                weight = ob.weight;
        }
        // constructor when all parameters are specified
        BoxWeight(double w, double h, double d, double m) {
                super(w, h, d); // call superclass constructor
                weight = m;
        }
        // default constructor
        BoxWeight() {
                super();
                weight = -1;
        }
        // constructor used when cube is created
        BoxWeight(double len, double m) {
                super(len);
                weight = m;
        }
}

class DemoSuper {
```

```
    public static void main(String args[]) {
        BoxWeight mybox1 = new BoxWeight(10, 20, 15, 34.3);
        BoxWeight mybox2 = new BoxWeight(2, 3, 4, 0.076);
        BoxWeight mybox3 = new BoxWeight(); // default
        BoxWeight mycube = new BoxWeight(3, 2);
        BoxWeight myclone = new BoxWeight(mybox1);
        double vol;
        vol = mybox1.volume();
        System.out.println("Volume of mybox1 is " + vol);
        System.out.println("Weight of mybox1 is " + mybox1.weight);
        System.out.println();
        vol = mybox2.volume();
        System.out.println("Volume of mybox2 is " + vol);
        System.out.println("Weight of mybox2 is " + mybox2.weight);
        System.out.println();
        vol = mybox3.volume();
        System.out.println("Volume of mybox3 is " + vol);
        System.out.println("Weight of mybox3 is " + mybox3.weight);
        System.out.println();
        vol = myclone.volume();
        System.out.println("Volume of myclone is " + vol);
        System.out.println("Weight of myclone is " + myclone.weight);
        System.out.println();
        vol = mycube.volume();
        System.out.println("Volume of mycube is " + vol);
        System.out.println("Weight of mycube is " + mycube.weight);
        System.out.println();
    }
}
```

TASK 1: *super* keyword is used to refer immediate parent class instance variable.

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields. Perform following steps

- Create a class and its sub class both having same data member.
- Use *super* in child class to print the value of parent class variable.
- create a test class to implement main method with demo functionality

TASK 2: *super* Keyword can be used to invoke parent class method.

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden. Perform following steps

- Create a class animal with a method eat() which prints "eating"
- Create subclass of animal class with method bark() which prints "barking" and method eat() same name as super class but it prints "eating meat"
- Write a print() method in subclass which call all the super and sub class methods.

TASK 3: *super* Keyword is used to invoke parent class constructor.

The super keyword can also be used to invoke the parent class constructor. Create a scenario in which above use of the *super* keyword is depicted.

MULTI LEVEL HIERARCHY:

TASK 4: Run the following code

```
class student
{
   int rollno;
   String name;
   student(int r, String n)
   {
      rollno = r;
      name = n;
   }
   void dispdatas()
```

```
    {
        System.out.println("Rollno = " + rollno);
        System.out.println("Name = " + name);
    }
}
class marks extends student
{
    int total;
    marks(int r, String n, int t)
    {
        super(r,n);   //call super class (student) constructor
        total = t;
    }
    void dispdatam()
    {
        dispdatas();    // call dispdatap of student class
        System.out.println("Total = " + total);
    }
}

class percentage extends marks
{
    int per;

    percentage(int r, String n, int t, int p)
    {
        super(r,n,t);  //call super class(marks) constructor
        per = p;
    }
    void dispdatap()
    {
        dispdatam();    // call dispdatap of marks class
        System.out.println("Percentage = " + per);
    }
}
class Multi_Inhe
```

```
{
   public static void main(String args[])
   {
      percentage stu = new percentage(102689, "RATHEESH", 350, 70); //call constructor percentage
      stu.dispdatap();  // call dispdatap of percentage class
   }
}
```

Polymorphism in Java

Polymorphism in java is a concept by which we can perform a single action by different ways. Polymorphism is derived from 2 greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

Usage of Java Method Overriding

Method overriding is used to provide specific implementation of a method that is already provided by its super class.

Method overriding is used for runtime polymorphism

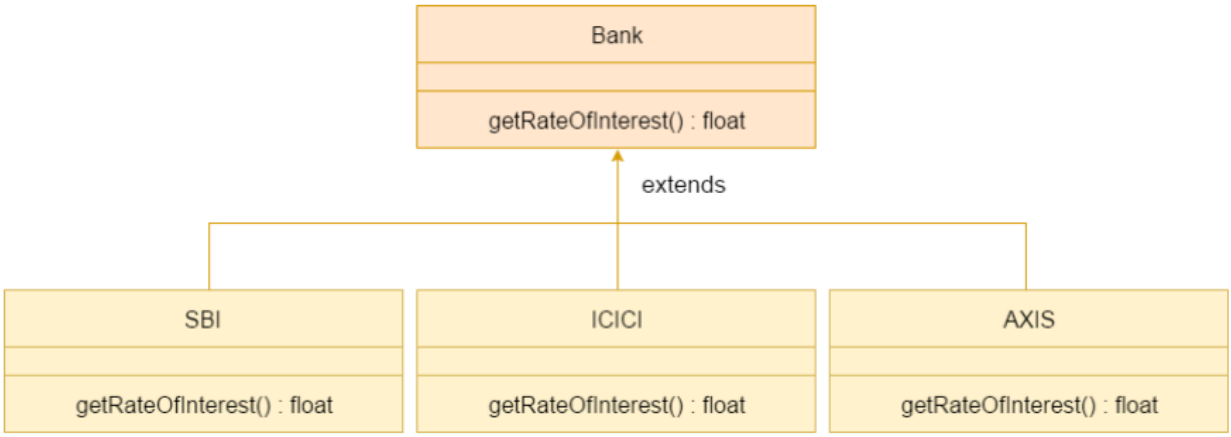
Rules for Java Method Overriding

1. method must have same name as in the parent class
2. method must have same parameter as in the parent class.
3. must be IS-A relationship (inheritance).

Sample Code:

```
class Vehicle{
void run(){System.out.println("Vehicle is running");}
}
class Bike2 extends Vehicle{
void run(){System.out.println("Bike is running safely");}

public static void main(String args[]){
Bike2 obj = new Bike2();
obj.run();
}
```

Task 05: Consider a scenario, Bank is a class that provides functionality to get rate of interest. But, rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7% and 9% rate of interest.



Task 06: Create Object in following manner:

Bank obj = new SBI();

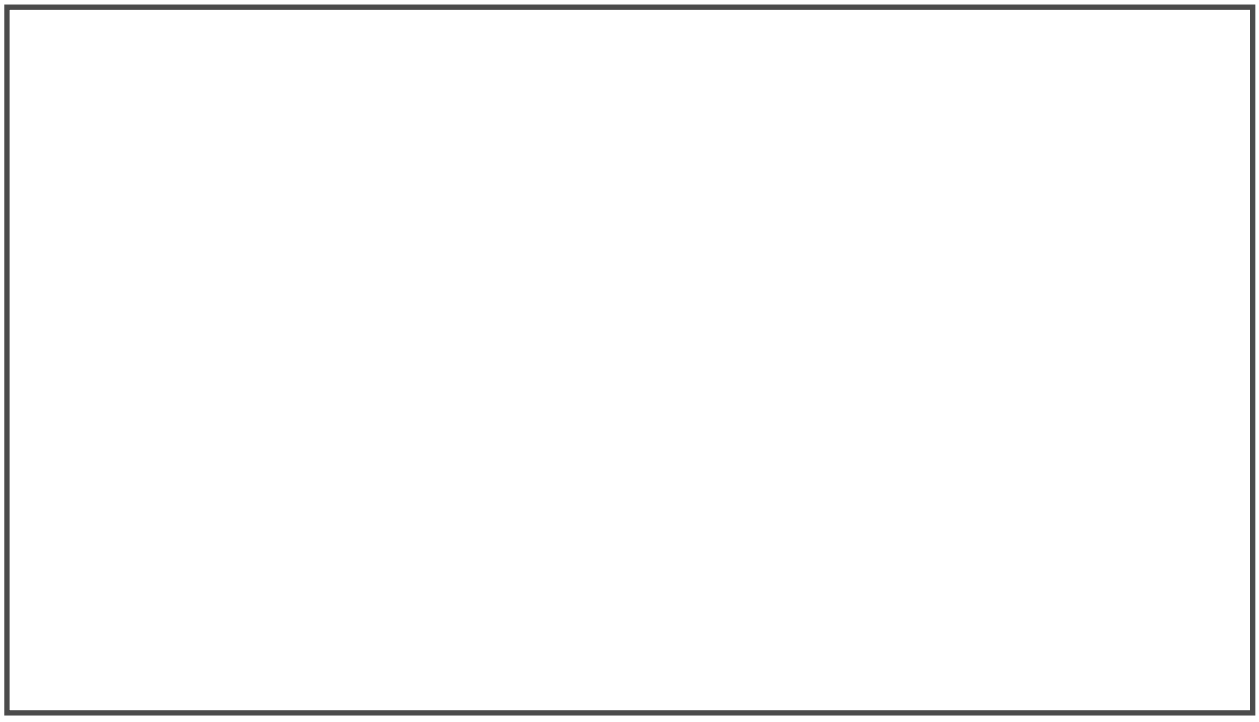
QUESTIONS

Please Fill the blank space with respective answers to following questions:

Question 1: What will be the output of the following program?

```
class Mobile {
   String str = "";
   Mobile() {
      str += "Mobile is : ";
   }
}
class Airtel extends Mobile {
   Airtel() {
      str += "Airtel";
```

```
    }
}
public class Docomo extends Mobile {
    Docomo() {
        str += "Docomo";
    }
    public static void main(String args[]) {
        Mobile d = new Docomo();
        System.out.println(d.str);
    }
```



```
}
```

Question 2: If parent class has a constructor animal which takes two parameters integer and a string. How will it be called in child class using

super? Write the calling line of code.

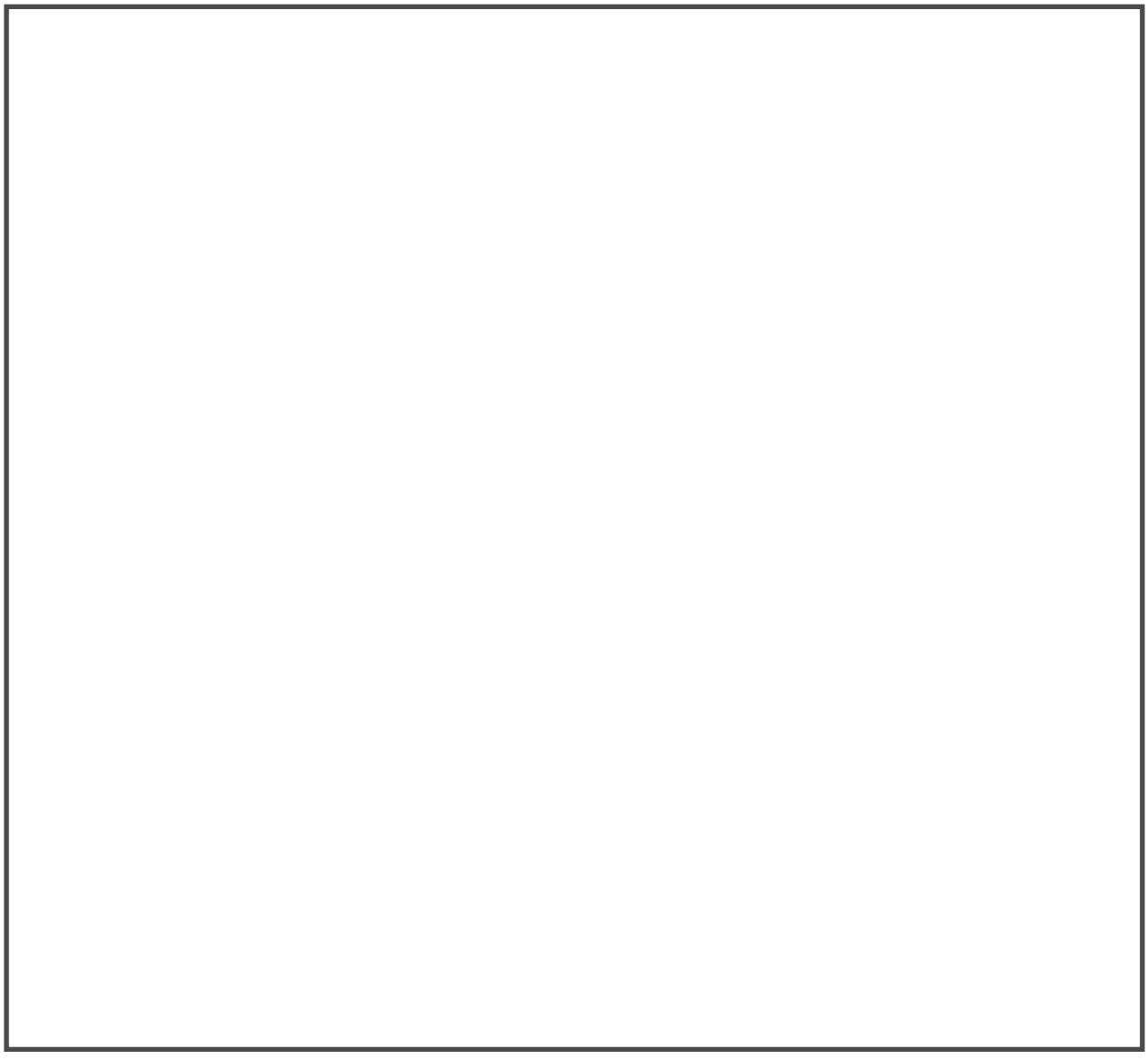
Question 3: What id difference between *this* and *super*?

Question 4: What is polymorphism? How to implement polymorphism in java?

Question 5: What will be the output of the following program?

```
public class MethodOverriding {
   public static void main(String args[]) {
      X x = new X();
      Y y = new Y();
      y.m2();
      x.m1();
      y.m1();
      x = y;
      x.m1();
   }
}
class X {
   public void m1() {
      System.out.println("m1 ~ X");
   }
```

```
}
class Y extends X {
    public void m1() {
        System.out.println("m1 ~ Y");
    }
    public void m2() {
        System.out.println("m2 ~ Y");
    }
}
```



THE END

JAVA ABSTRACTION

Lab-11

LAB 11	Abstraction in Java

Lab Objectives:

1. Understanding the concept of polymorphism
2. Understand the super keyword

Software Required:

Netbeans IDE

Abstract Classes

Sample Code

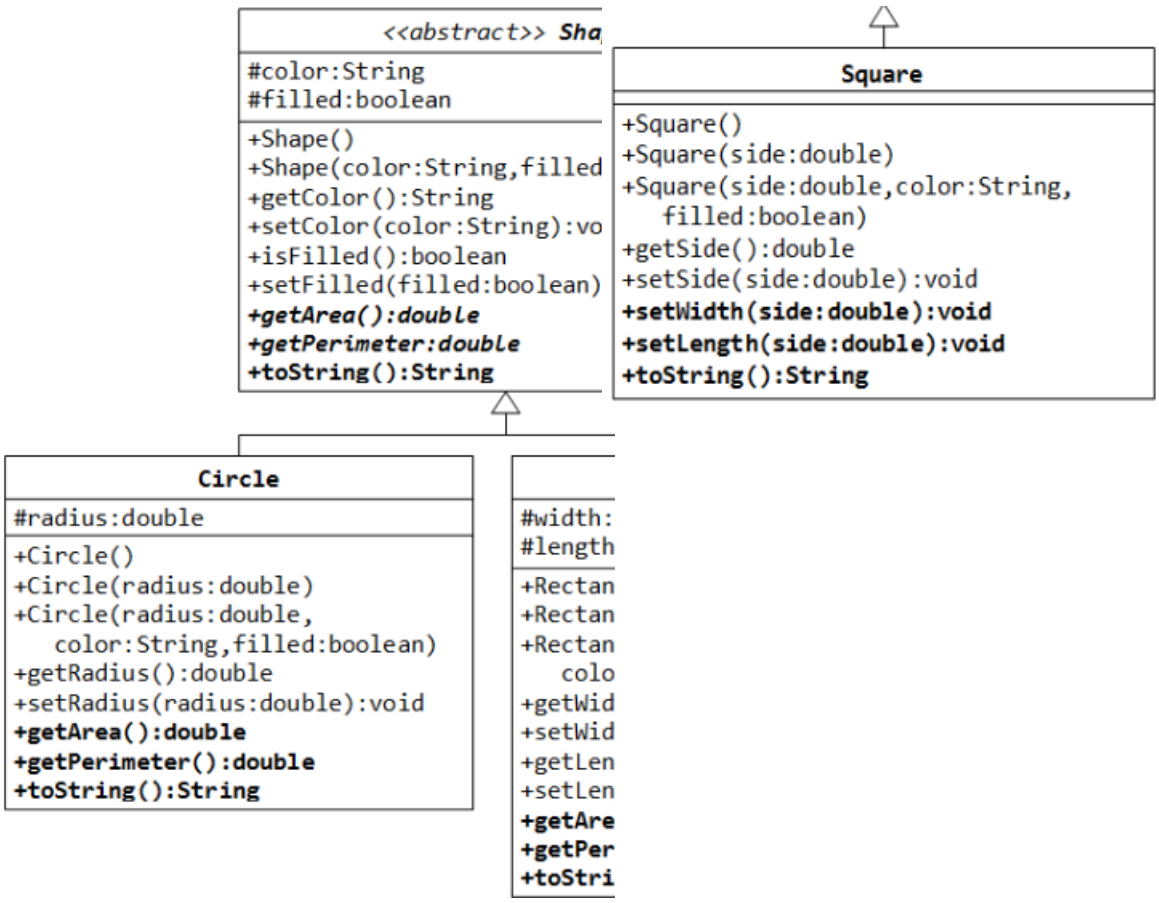
```
// A Simple demonstration of abstract.
abstract class A {
abstract void callme();
// concrete methods are still allowed in abstract classes
void callmetoo() {
System.out.println("This is a concrete method.");
}
}
class B extends A {
void callme() {
System.out.println("B's implementation of callme.");
}
}

class AbstractDemo {
public static void main(String args[]) {
B b = new B();
b.callme();
b.callmetoo();
}
```

}

Task 1: Implement the following Class diagram.

In this exercise, Shape shall be defined as an abstract class, which contains:

- Two protected instance variables color(String) and filled(boolean). The protected variables can be accessed by its subclasses and classes in the same package. They are denoted with a '#' sign in the class diagram.
- Getter and setter for all the instance variables, and toString().
- Two abstract methods getArea() and getPerimeter() (shown in italics in the class diagram).
- The subclasses Circle and Rectangle shall override the abstract methods getArea() and getPerimeter() and provide the proper implementation. They also override the toString().



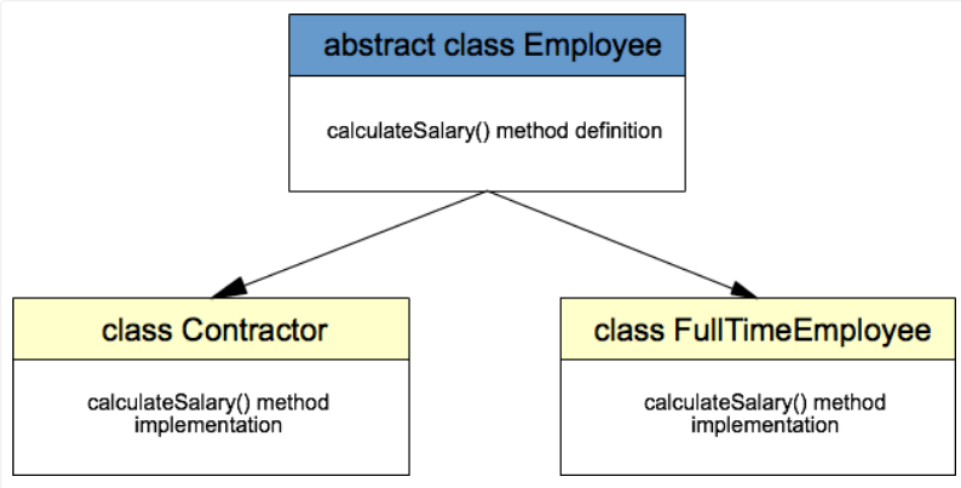
Task 02: Try to run following code in your main method after implementation of TASK 1

```
Shape s1 = new Circle(5.5, "RED", false);  // Upcast Circle to Shape
System.out.println(s1);                    // which version?
System.out.println(s1.getArea());          // which version?
System.out.println(s1.getPerimeter());     // which version?
System.out.println(s1.getColor());
System.out.println(s1.isFilled());
System.out.println(s1.getRadius());

Circle c1 = (Circle)s1;                    // Downcast back to Circle
System.out.println(c1);
System.out.println(c1.getArea());
System.out.println(c1.getPerimeter());
System.out.println(c1.getColor());
System.out.println(c1.isFilled());
System.out.println(c1.getRadius());

Shape s2 = new Shape();

Shape s3 = new Rectangle(1.0, 2.0, "RED", false);   // Upcast
System.out.println(s3);
System.out.println(s3.getArea());
System.out.println(s3.getPerimeter());
System.out.println(s3.getColor());
System.out.println(s3.getLength());
```

```
Rectangle r1 = (Rectangle)s3;   // downcast
System.out.println(r1);
System.out.println(r1.getArea());
System.out.println(r1.getColor());
System.out.println(r1.getLength());

Shape s4 = new Square(6.6);     // Upcast
System.out.println(s4);
System.out.println(s4.getArea());
System.out.println(s4.getColor());
System.out.println(s4.getSide());

// Take note that we downcast Shape s4 to Rectangle,
//  which is a superclass of Square, instead of Square
Rectangle r2 = (Rectangle)s4;
System.out.println(r2);
System.out.println(r2.getArea());
System.out.println(r2.getColor());
System.out.println(r2.getSide());
System.out.println(r2.getLength());

// Downcast Rectangle r2 to Square
Square sq1 = (Square)r2;
System.out.println(sq1);
System.out.println(sq1.getArea());
System.out.println(sq1.getColor());
System.out.println(sq1.getSide());
System.out.println(sq1.getLength());
```

TASK 03: Create one superclass called Employee and two subclasses – Contractor and FullTimeEmployee. Both subclasses have common properties to share, like the name of the employee and the amount of money the person will be paid per hour. There is one major difference between contractors and full-time employees – the time they work for the

company. Full-time employees work constantly 8 hours per day and the working time of contractors may vary.



QUESTIONS

Please Fill the blank space with respective answers to following questions:

Question 1: What will be the output of the following program?

```
abstract class Shape
{
   int b = 20;
   public void display()
   {
      System.out.println("This is display method");
      System.out.println(b);
   }
   abstract public void calculateArea();
}
class Rectangle extends Shape
{
   public static void main(String args[])
   {
      Rectangle obj = new Rectangle();
```

```
        obj.b = 200;
        obj.display();
    }
      public void calculateArea() {    }
```

```
}
```

Question 2: What will be the output of the following program?

```
class CalculateAreas
{
   public static void main(String arg[])
   {
      Square sq = new Square(5.25);
      System.out.println("Area of Square is " + sq.getArea());
   }
}

abstract class Shape
{
   abstract double getArea();
}

class Square extends Shape
{
```

```
    double side;

    double getArea()
    {
        return side * side;
    }
```

```
}
```

Question 3: What will be the output of the following program? Also explain the reason behind the output.

```
public class AbstractClass {
    public static void main(String[] args) {
        Shape shape1;  // LINE A
        Shape shape2 = new Shape(); // LINE B
        System.out.println("Compiles Successfully");
    }
}
abstract class Shape {}
```

class Rectangle extends Shape {}

THE END

JAVA ENCAPSULATION & PACKAGES

Lab-12

LAB 12	Encapsulation and Packages in Java

Lab Objectives:

1. Understanding the concept of Encapsulation
2. Implementation of encapsulation using public functions
3. Understanding of real world scenarios and application of encapsulation

Software Required:

Netbeans IDE

Encapsulation

The whole idea behind encapsulation is to hide the implementation details from users. If a data member is private it means it can only be accessed within the same class. No outside class can access private data member (variable) of other class. However if we setup public getter and setter methods to update (for example void setSSN(int ssn))and read (for example int getSSN()) the private data fields then the outside class can access those private data fields via public methods. This way data can only be accessed by public methods thus making the private fields and their implementation hidden for outside classes. That's why encapsulation is known as data hiding.

TASK 01: Run the following sample code

```
/* File name : EncapTest.java */

public class EncapTest {

   private String name;

   private String idNum;

   private int age;

   public int getAge() {

   return age;
```

```
   }

   public String getName() {

      return name;

   }

   public String getIdNum() {

      return idNum;

   }

   public void setAge( int newAge) {

      age = newAge;

   }

   public void setName(String newName) {

      name = newName;

   }

   public void setIdNum( String newId) {

      idNum = newId;

   }

}
```

Accessing of above class using following class

```
/* File name : RunEncap.java */

public class RunEncap {



   public static void main(String args[]) {

      EncapTest encap = new EncapTest();

      encap.setName("James");
```

```
        encap.setAge(20);

        encap.setIdNum("12343ms");



        System.out.print("Name : " + encap.getName() + " Age : " + encap.getAge());

    }

}
```

Task 02: Create a class Employee with age, name and ssn as private data members.

Write set ad get methods for the private data members. Methods must apply following conditions:

- Name is not null
- SSN is between 100 and 999
- Age is not a negative number
- Age is not 0.

Try to set the values of the Employee class data members using another class and also display the values of the data members.

Packages

The package is both a naming and a visibility control mechanism. You can define classes inside a package that are not accessible by code outside that package. You can also define class members that are exposed only to other members of the same package. This allows your classes to have intimate knowledge of each other, but not expose that knowledge to the rest of the world.

Because of the interplay between classes and packages, Java addresses four categories of visibility for class members:

1. Subclasses in the same package
2. Non-subclasses in the same package
3. Subclasses in different packages
4. Classes that are neither in the same package nor subclasses

Task 03: Create packages and try to access private, protected, public and default data members in another package. Also create subclasses a class outside its package and try to access private, protected, default and public data members.

	Private	No Modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

QUESTIONS

Please Fill the blank space with respective answers to following questions:

Question 1: Encapsulation is also called as?

Question 2: Encapsulation is implemented by using _____ access

_____

_____

modifier?

Question 3: What will be the output of the following program? Assume that we run LittleFlowerSchool class.

```
package com.mc.details;
import com.mc.Stud1.Student;
import com.mc.Stud2.StudentDetails;
public class LittleFlowerSchool {
    public static void main(String[] args) {
        Student std = new Student();
        System.out.println("Rank is " + std.rank);
        System.out.println("Marks is " + std.marks);
        System.out.println(StudentDetails.address);
    }
}
package com.mc.Stud1;
public class Student {
    public int rank = 10;
    public double marks = 89;
}
package com.mc.Stud2;
public class StudentDetails {
    public static String address = "Hyderabad, Andhra Pradesh, India";
```

}

Question 4: What will be the output of the following program? Assume that if we run XXL class.

```
package com.mc.size;
import com.mc.group.M;
import com.mc.type.XL;
public class XXL {
    public static void main(String[] args) {
        for (int i = 0; i < 3; i++) {
            System.out.print(XL.str + new XL().srt + XL.rts + new XL().rst);
            new M();
            System.out.print(M.a[i]);
        }
    }
}
package com.mc.type;
public class XL {
    public static String str = "~";
    public String srt = "&";
    public static String rts = "#";
    public String rst = "@";
    public static void main(String[] args) {
    }
```

```
}
package com.mc.group;
public class M {
   static public char[] a = {'a', 'b', 'c'};
   public static void main(String[] args) {
      System.out.print(a);
   }
}
```

THE END

JAVA INTERFACES

Lab-13

LAB 13	Interface in Java

Lab Objectives:

1. Understanding the concept of Interfaces
2. Implementation and real world usage of Interfaces

Software Required:

Netbeans IDE

Interface

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface. Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods. Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements.

Sample Code

```
/* Animal.java */
interface Animal {
  public void eat();
  public void travel();
}

/*MammalInt.java */
public class MammalInt implements Animal {
  public void eat() {
    System.out.println("Mammal eats");
  }
  public void travel() {
```

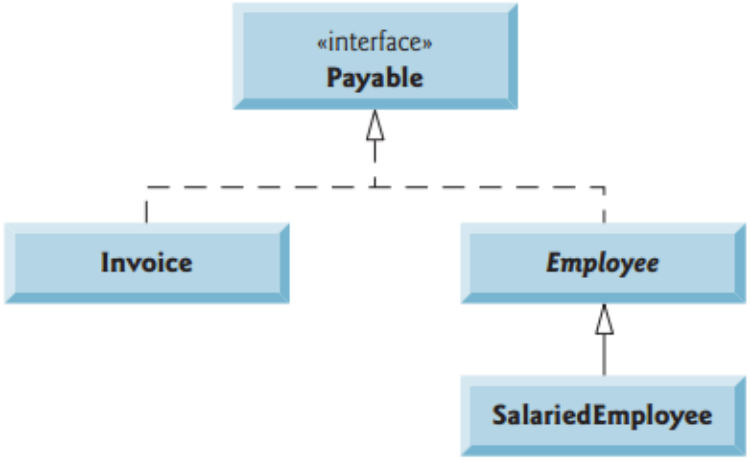
```
    System.out.println("Mammal travels");
  }
  public int noOfLegs() {
    return 0;
  }
  public static void main(String args[]) {
    MammalInt m = new MammalInt();
    m.eat();
    m.travel();
  }
}
```

Task 1: Developing a Payable Hierarchy

To build an application that can determine payments for employees and invoices alike, we first create interface Payable, which contains method getPaymentAmount that returns a double amount that must be paid for an object of any class that implements the interface. After declaring interface Payable, we introduce class Invoice, which implements interface Payable. We then modify class Employee such that it also implements interface Payable.

Classes Invoice and Employee both represent things for which the company must be able to calculate a payment amount. Both classes implement the Payable interface, so a program can invoke method getPaymentAmount on Invoice objects and Employee objects alike.

The UML class diagram below shows the hierarchy used in our accounts payable application. The hierarchy begins with interface Payable. The UML distinguishes an interface from other classes by placing the word “interface” in (« and ») above the interface name. The UML expresses the relationship between a class and an interface through a relationship known as **realization**. A class is said to “realize,” or implement, the methods of an interface. A class diagram models a realization as a dashed arrow with a hollow arrowhead pointing from the implementing class to the interface. The diagram indicates that classes Invoice and Employee each realize (i.e., implement) interface Payable. Class Employee appears in italics, indicating that it’s an abstract class. Concrete class SalariedEmployee extends Employee and *inherits its superclass’s realization relationship* with interface Payable.



Invoice Class:

Create class Invoice to represent a simple invoice that contains billing information for only one kind of item. The class declares private instance variables itemNumber, itemDescription, quantity and pricePerItem Class Invoice also contains a constructor, *get* and *set* methods that manipulate the class's instance variables and a toString method that returns a String representation of an Invoice object. Methods setQuantity and setPricePerItem ensure that quantity and pricePerItem obtain only nonnegative values.

Class Employee:

We now modify class Employee such that it implements interface Payable. It does not make sense to implement method getPaymentAmount in class Employee because we cannot calculate the getPaymentAmount owed to a general Employee—we must first know the specific type of Employee. This forced each Employee concrete subclass to override getPaymentAmount with an implementation. The Employee class will contain variables first name, last name and SSN. SalariedEmployee class will contain variable Salary. Both class will have get and set methods and constructors. toString method must also be implemented in both parent and child classes which display the values of the class data members.

Main Method:

```
// PayableInterfaceTest.java
// Tests interface Payable.
public class PayableInterfaceTest
{
    public static void main( String args[] )
```

```
{
   // create four-element Payable array
   Payable payableObjects[] = new Payable[ 4 ];

   // populate array with objects that implement Payable
   payableObjects[ 0 ] = new Invoice( "01234", "seat", 2, 375.00 );
   payableObjects[ 1 ] = new Invoice( "56789", "tire", 4, 79.95 );
   payableObjects[ 2 ] =
      new SalariedEmployee( "John", "Smith", "111-11-1111", 800.00 );
   payableObjects[ 3 ] =
      new SalariedEmployee( "Lisa", "Barnes", "888-88-8888", 1200.00 );

   System.out.println(
      "Invoices and Employees processed polymorphically:\n" );

   // generically process each element in array payableObjects
   for ( Payable currentPayable : payableObjects )
   {
      // output currentPayable and its appropriate payment amount
      System.out.println(currentPayable.toString() + “payment due” +
currentPayable.getPaymentAmount() );
   } // end for
} // end main
```

QUESTIONS

Please Fill the blank space with respective answers to following questions:

Question 1: What is the difference between interface and an abstract class?

Question 2: What will be the output of the following program?

```
public class CricketPlayersUsingInterfaces {
    public static void main(String s[]) {
        StrongBatsmen sachin = new StrongBatsmen("Sachin", 100, 326);
        sachin.makeCentury();
        sachin.takeWickets();
        StrongBatsmen gambhir = new StrongBatsmen("Gambhir", 25);
        gambhir.makeCentury();
    }
}
class StrongBatsmen implements IBatsmen, IBowler {
    int numberOfCenturies;
    String name;
    int wickets;
    StrongBatsmen(String name, int numberOfCenturies, int wickets) {
        this.numberOfCenturies = numberOfCenturies;
        this.wickets = wickets;
        this.name = name;
    }
    StrongBatsmen(String name, int numberOfCenturies) {
        this.numberOfCenturies = numberOfCenturies;
        this.name = name;
    }
    public void makeCentury() {
        System.out.println(name + " made " + numberOfCenturies + " centuries.");
    }
    public void takeWickets() {
        System.out.println(name + " taken " + wickets + " wickets.");
    }
}
interface IBatsmen {
```

```
    void makeCentury();
}
interface IBowler {
    void takeWickets();
```

```
}
```

Question 3: What will be the output of the following program?

```
public class King {
    public static void main(String[] args) {
        King k = new King();
        System.out.print("Output = " + k.new Elephant().step2(2, 3));
    }
    interface Queen {
        float step2(int low, int high);
    }
    interface Pawn {
        float step3(int a, int b, int c);
    }
    abstract class Knight implements Queen, Pawn {
    }
    class Elephant implements Queen {
        public float step2(int x, int y) {
            return (float)(x * 3);
        }
    }
```

}

THE END

JAVA EXCEPTION HANDLING

Lab-14

LAB 14	Exception Handling

Lab Objectives:

1. Understanding the concept of Exception Handling in Java
2. Implementing exception handling in java
3. Creating your own exceptions

Software Required:

Netbeans IDE

EXCEPTIONS:

Exception handling is a very important yet often neglected aspect of writing robust software. When an error occurs in a Java program it usually results in an exception being thrown. How you throw, catch and handle these exception matters. Runtime errors appear in Java as exceptions, exception is a special type of classes that could be thrown to indicate a runtime

error and provide additional data about that error. If a method is declared to throw an exception, any other method calls it should deal with this exception by throwing it (if it appears) or handle it. Three categories of errors (syntax errors, runtime errors, and logic errors):

- Syntax errors arise because the rules of the language have not been followed. They are detected by the compiler.
- You can use the debugging techniques to find logic errors.
- Exception handling to deal with runtime errors and using assertions to help ensure program correctness.

TASK 01: Run following code

```
12  public class Lab7 {
13
14      /**
15       * @param args the command line arguments
16       */
17      public static void main(String[] args) {
18          // TODO code application logic here
19          int a[] = new int[2];
20          try {
21              System.out.println("Access element three :" + a[3]);
22          } catch (ArrayIndexOutOfBoundsException e) {
23              System.out.println("Exception thrown  :" + e);
24          } finally {
25              a[0] = 6;
26              System.out.println("First element value: " + a[0]);
27              System.out.println("The finally statement is executed");
28          }
29      }
30  }
```

TASK 02: Run the following code

```
import java.util.*;
import javax.swing.*;
5   class ABC {
6       public static void main(String [] args)
7   {
8         try {
9             A();
10            B();
11        }
12        catch(Exception e){
13          System.out.println("in main...");
14          System.out.println(e);
15       }
16      }
17      public static void A(){
18        try {
19            B();
20        } catch(Exception e) {
21            System.out.println("in A...");
22          System.out.println(e);
23        }
24      }
25  public static void B()throws Exception
26  {
27          C();
28      }
29      public static void C()throws Exception{
30          throw new Exception("C is exceptional");
31  }
32  }
```

TASK 03: Run the following code// File Name InsufficientFundsException.java

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

// File Name CheckingAccount.java

```
import java.io.*;
public class CheckingAccount {
 private double balance;
private int number;
public CheckingAccount(int number) {
this.number = number;
}
public void deposit(double amount) {
balance += amount;
}
public void withdraw(double amount) throws InsufficientFundsException {
if(amount <= balance) {
balance -= amount; }
else {
```

```
double needs = amount - balance;
throw new InsufficientFundsException(needs);
}
}
public double getBalance() {
return balance;
}
public int getNumber() {
return number;
} }
```

// File Name BankDemo.java

```
public class BankDemo {
public static void main(String [] args) { CheckingAccount c = new CheckingAccount(101);
System.out.println("Depositing $500..."); c.deposit(500.00);
try {
System.out.println("\nWithdrawing $100...");
c.withdraw(100.00);
System.out.println("\nWithdrawing $600...");
c.withdraw(600.00);
} catch (InsufficientFundsException e) {
System.out.println("Sorry, but you are short $" + e.getAmount());
e.printStackTrace();
} }
}
```

TASK 04: Write a program that counts even numbers between minimum and maximum values. Maximum and minimum values are provided by user. If minimum value is greater than or equal to maximum value, the

program should throw an InvalidRange exception and handle it to display a message to the user on the following format: Invalid range: minimum is greater than or equal to maximum.

QUESTIONS

Please Fill the blank space with respective answers to following questions:

_____

_____

_____

Question 1: What is the difference between throw and throws?

Question 2: What is the output of the following program?

```
public class ErrorTest {
   public static void main(String[] args) {
      float f = 90;
      char i = 0;
      try {
         float d = f / i;
         System.out.println(d);
      } catch (Exception e) {
         System.out.println("Exception");
      }
   }
```

}

Question 3: What is the output of the following program?

```
public class MarksValidation {
    public static void main(String[] args) throws Exception {
        int marks = 102;
        if (marks < 100) {
            System.out.println("You marks are Valid");
        } else {
            try {
                throw new MyException("You marks are not Valid");
            } catch (MyException e) {
                System.out.println(e.getMessage());
            }
        }
    }
}
class MyException extends Exception {
    public MyException(String string) {
        super(string);
    }
```

}

THE END

KHWAJA FAREED UEIT RAHIM YAR KHAN | Faculty of Information Technology

JAVA FILE HANDLING

Lab-15

LAB 15	File Handling

Lab Objectives:

1. Understand java IO Streaming
2. Understanding and use of java.io
3. Reading and writing files in java

Software Required:

Netbeans IDE

Java I/O Tutorial

Java I/O (Input and Output) is used *to process the input* and *produce the output*. Java uses the concept of stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations. We can perform file handling in java by Java I/O API. Stream

OutputStream class: OutputStream class is an abstract class. It is the super class of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Java FileOutputStream Class : Java FileOutputStream is an output stream used for writing data to a file. If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use FileWriter than FileOutputStream.

FileOutputStream class declaration

Let's see the declaration for Java.io.FileOutputStream class:

```
public class FileOutputStream extends OutputStream
```

Java FileOutputStream Example 1: write byte

Task 1: Try to write string into a file using FileOutputStream

Hint:
String s="Welcome to javaTpoint.";
byte b[]=s.getBytes();//converting string into byte array

Java FileInputStream Class: Java FileInputStream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character- stream data. But, for reading streams of characters, it is recommended to use FileReader class.

```
import java.io.FileOutputStream;
public class FileOutputStreamExample {
   public static void main(String args[]){
        try{
         FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
         fout.write(65);
         fout.close();
         System.out.println("success...");
        }catch(Exception e){System.out.println(e);}
    }
}
```

Java FileInputStream class declaration

Let's see the declaration for java.io.FileInputStream class:

```
public class FileInputStream extends InputStream
```

Java FileInputStream Example 2: read single character

```
import java.io.FileInputStream;
public class DataStreamExample {
    public static void main(String args[]){
        try{
          FileInputStream fin=new FileInputStream("D:\\testout.txt");
          int i=fin.read();
          System.out.print((char)i);

          fin.close();
        }catch(Exception e){System.out.println(e);}
       }
      }
```

Task 02: Try to read all characters from file

Java BufferedOutputStream Class : Java BufferedOutputStream class is used for buffering an output stream. It internally uses buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast. For adding the buffer in an OutputStream, use the BufferedOutputStream class. Let's see the syntax for adding the buffer in an OutputStream:

OutputStream os= new BufferedOutputStream(new FileOutputStream("D:\\IO Package\ \testout.txt”));

Java BufferedOutputStream class declaration

public class BufferedOutputStream extends FilterOutputStream

Example 3 of BufferedOutputStream class:

```
1       FileInputStream fin=new FileInputStream("D:\\testout.txt");
2               int i=0;
3               while((i=fin.read())!=-1){
4                System.out.print((char)i);
5               }
```

```
package com.javatpoint;
import java.io.*;
public class BufferedOutputStreamExamp
public static void main(String args[])thr
    FileOutputStream fout=new FileOutput
    BufferedOutputStream bout=new Buff
    String s="Welcome to javaTpoint.";
    byte b[]=s.getBytes();
    bout.write(b);
    bout.flush();
    bout.close();
    fout.close();
    System.out.println("success");
}
}
```

```
package com.javatpoint;

import java.io.*;
public class BufferedInputStreamExample{
 public static void main(String args[]){
  try{
    FileInputStream fin=new FileInputStream("D:\\testout.txt");
    BufferedInputStream bin=new BufferedInputStream(fin);
    int i;
    while((i=bin.read())!=-1){
     System.out.print((char)i);
    }
    bin.close();
    fin.close();
  }catch(Exception e){System.out.println(e);}
 }
}
```

Example 4 of Java BufferedInputStream

Java Writer: It is an abstract class for writing to character streams. The methods that a subclass must implement are write(char[], int, int), flush(), and close(). Most subclasses will override some of the methods defined here to provide higher efficiency, functionality or both.

```
import java.io.*;
public class WriterExample {
    public static void main(String[] args) {
        try {
            Writer w = new FileWriter("output.txt");
            String content = "I love my country";
            w.write(content);
            w.close();
            System.out.println("Done");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Java Reader: Java Reader is an abstract class for reading character streams. The only methods that a subclass must implement are read(char[], int, int) and close(). Most subclasses, however, will override some of the methods to provide higher efficiency,

```
package com.javatpoint;

import java.io.FileReader;
public class FileReaderExample {
   public static void main(String args[])throws Exception{
        FileReader fr=new FileReader("D:\\testout.txt");
        int i;
        while((i=fr.read())!=-1)
        System.out.print((char)i);
        fr.close();
   }
}
```

additional functionality, or both.

Task 03: Read following line from a file and calculate the sum of last two digits

“Sameen Fatima”, 2323, 30.99, 20.23

Hint: use function readline() of BufferredReader to read the line and then use lineRead.split(“,”) to split the line read from file.

Task 04: Read following paragraph from a file and write it on another file using Filewriter and Filereader.

Developing writers can often benefit from examining an essay, a paragraph, or even a sentence to determine what makes it effective. On the following pages are several paragraphs for you to evaluate on your own, along with the Writing Center's explanation. Note: These passages are excerpted from actual student papers and retain the original wording. Some of the sentences are imperfect, but we have chosen these passages to highlight areas in which the author was successful.

Thanks to the students who provided their writing for this page of our website. Do you have a discussion post, paper, or other writing assignment that you are particularly proud of? We would love to use it as a sample.

QUESTIONS

Please Fill the blank space with respective answers to following questions:

Question 1: What will be the output of following code ?

```
public static void main(String[] args){
String parent = null;
File file = new File(parent, "myfile.txt");
try {
file.createNewFile();
} catch (IOException e) {
e.printStackTrace();
}
}
```

Question 2: What are flush() and close() used for ?

Question 3: What is the difference between the Reader/Writer class hierarchy and the InputStream/OutputStream class hierarchy?

Question 4: What will be the output of the following program?

```
import java.io.*;
import java.util.*;
public class FileTest {
    public static void main(String args[]) throws IOException
{
        FileWriter fw = new FileWriter("ScanText.txt");
        fw.write("1 9.1 3 4 5.1 7.1 so on 2.6");
        fw.close();
        FileReader fr = new FileReader("ScanText.txt");
        Scanner scanner = new Scanner(fr);
        while (scanner.hasNext()) {
            if (scanner.hasNextDouble()) {
                System.out.print(scanner.nextDouble() + ",");
            } else
                break;
        }
        fr.close();
    }
}
```

THE END