**Name:**

**Advanced Programming in Java**

**Lab Exercise 1.3.2022**

public interface Sports

{

void method1( );

void method2( );

int method3(double d);

}

public class Baseball implements Sports

{

public Baseball( )

{ . . . }

public void method1( )

{ //some code…}

public void method2( )

{ //some code…}

public int method3(double c )

{ //some code…}

public int statevar1;

}

public class Football implements Sports

{

public Football( )

{ . . . }

public void method1( )

{ //some code…}

public void method2( )

{ //some code…}

public int method3(double c )

{ //some code…}

public int statevar1;

}

public class Tester

{

public static void main(String[] args)

{

Sports x = new Baseball( );

Sports y = new Football( );

x.method2( );

y.method2( );

//more code

}

}

Read Lesson 38 in your textbook.

Use the above code in the following questions:

1. Which methods, if any, in the *Sports* interface are abstract?
2. public class Hockey implements Sports

{

//What methods, if any, **must** we implement here?

}

1. Look at the classes *Baseball* and *Football*. Both implement *method1*. Do both implementations have to have identical code? If so, why?
2. In the “more code” section of *Tester* what would the following return? (x instanceof Sports)
3. In the “more code” section of *Tester* what would the following return? (y instanceof Football)
4. The property of two classes being able to have methods of the same name (but with possibly different implementations) is known as \_\_\_\_\_\_\_\_\_\_\_\_\_\_.
5. Modify the following class so that it will simultaneously inherit the *Red* class and implement both the *Eagle* and *Bobcat* interfaces.

public class Austria

{

**. . .**

}

**Project… Linear Function**

You are a software engineer with the Blue Pelican Engineering Corporation. You’re immediate

supervisor has need of a class called *LinearFunction* and she knows exactly the methods that it

needs to include. Not having time to write it herself, she assigns the job to you. To insure that

you produce exactly the methods she wants, she is providing the interface below and requiring

that you implement this interface in the *LinearFunction* class you produce. When your project is

complete, she will simply look at your class signature and if she sees *implements*

*LinearFunctionMethods*, she will know for certain that you have implemented all the methods

she originally specified in the interface; otherwise, your code would not compile.

public interface LinearFunctionMethods

{

double getSlope( );

double getYintercept( );

double getRoot( );

double getYvalue(double x); //return the y value corresponding to x

double getXvalue(double y); //return the x value corresponding to y

}

For simplicity we will assume that the linear function’s graph can never be vertical or horizontal.

(This eliminates some complications with the math). In writing your methods, simply recall the *y*

*= mx + b* portion of your algebra studies. The constructor of your class should allow you to pass

the slope (m) and y-intercept(b) of the *LinearFunction* object you are instantiating.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Test your *LinearFunction* class with the *Tester* class below:**

import java.io.\*;

import java.util.\*;

public class Tester

{

public static void main(String args[])

{

Scanner kbReader = new Scanner(System.in);

System.out.print("What is the slope of your line? ");

double slope = kbReader.nextDouble();

System.out.print("What is the y-intercept of your line? ");

double yIntc = kbReader.nextDouble( );

LinearFunction line = new LinearFunction(slope, yIntc);

System.out.println("\nSlope of this line is: " + line.getSlope( ));

System.out.println("Y-intercept of this line is: " + line.getYintercept( ));

System.out.println("Root of this line is: " + line.getRoot( ));

System.out.print("\nWhat is an x value for which you wish to solve for y? ");

double x = kbReader.nextDouble( );

double yValue = line.getYvalue(x);

System.out.println("The y value corresponding to x = " + x + " is " + yValue);

System.out.print("\nWhat is a y value for which you wish to solve for x? ");

double y = kbReader.nextDouble( );

double xValue = line.getXvalue(y);

System.out.println("The x value corresponding to y = " + y + " is " + xValue);

}

}

**Below is a typical run:**

What is the slope of your line? -3

What is the y-intercept of your line? 2.5

Slope of this line is: -3.0

Y-intercept of this line is: 2.5

Root of this line is: 0.8333333134651184

What is an x value for which you wish to solve for y? -4.61

The y value corresponding to x = -4.610000133514404 is 16.329999923706055

What is a y value for which you wish to solve for x? 5.0

The x value corresponding to y = 5.0 is -0.833333313465118