Adam Corbin COP 5330 - 002

1

Fib.java

```
public class Fib {

    // constructor
    public Fib(int f0, int f1) {
        this.f0 = f0;
        this.f1 = f1;
    }


    /**
     * Here are the following steps this function will do:
     * 1. This main function expects to read in 3 arguments from the command
line which will be f(0), f(1), and n.
     * 2. Fib object will be created passing in f(0) and f(1)
     * 3. Then using the f method, the fibonacci number will be computed for
0..n and printed out to the screen
     * 4. Then using the fRec method, the fibonacci number will be computed for
0..n and printed out to the screen
     *
     * @param args Expecting a list of 3 integers.
     *             1. will be f(0) which should be 0
     *             2. will be f(1) which should be 1
     *             3. will be n for the nth item that is desired to be computed
     */
    public static void main(String[] args) {

        // get numbers F(0) and F(1) from args[0] and args[1].
        // use either the Scanner class or Integer.parseInt(args[...])
        // you must handle possible exceptions !
        if (args.length != 3) {
            System.out.println("ERROR: Expecting to have 3
arguments[f(0),f(1),n]. You have only provided " + args.length + " arguments");
            System.exit(0);
        }

        int n = 0;
        int f0 = 0;
        int f1 = 0;
        try {
            f0 = Integer.parseInt(args[0]);
            f1 = Integer.parseInt(args[1]);

            // get n from args[2]:
            n = Integer.parseInt(args[2]);
        } catch (Exception e) {
```

```
            System.out.println("ERROR: Parsing Arguments error. Please be sure
to enter 3 integers as inputs. Exiting");
            System.exit(0);
        }

        System.out.println(f0 + " " + f1 + " " + n);

        // create a Fib object with params F(0) and F(1)
        Fib fib = new Fib(f0, f1);


        // calculate F(0), ..., F(n) and display them with
System.out.println(...) using
        // the iterative methode f(i)
        for (int i = 0; i < 11; i++) {
            System.out.println("Iterative fib for " + i + " is:" + fib.f(i));
        }
        System.out.println("---------------------");


        // calculate F(0), ..., F(n) and display them with
System.out.println(...) using
        // the recursive methode fRec(i)
        for (int i = 0; i < 11; i++) {
            System.out.println("Recursive fib for " + n + " is:" + fib.fRec(i));
        }
    }

    /**
     * Computes F(n) using an ***iterative*** algorithm, where F(n) = F(n-1) +
F(n-2) is the recursive definition.
     * Uses instance variables that store F(0) and F(1).
     *
     * @param n The nth Fib sequence that is desired to be computed
     * @return result of the nth fib sequence
     * @throws ArithmeticException when n less than 0
     */
    //
    // use instance variables that store F(0) and F(1).
    // check parameter and throw exception if n < 0. Don't worry about
arithmetic overflow.
    public int f(int n) throws ArithmeticException {
        if (n < 0) {
            throw new ArithmeticException("n less than 0");
        }

        if (n == 0) {
            return f0;
        } else if (n == 1) {
            return f1;
        }

        int[] fibArray = new int[n + 1];
        fibArray[0] = f0;
        fibArray[1] = f1;
        for (int i = 2; i <= n; i++) {
            fibArray[i] = fibArray[i - 1] + fibArray[i - 2];
        }
```

```
        return fibArray[n];
    }

    /**
     * Computes F(n) using the ***recursive*** algorithm, where F(n) = F(n-1) +
F(n-2) is the recursive definition.
     * Uses instance variables that store F(0) and F(1).
     *
     * @param n The nth Fib sequence that is desired to be computed
     * @return result of the nth fib sequence
     * @throws ArithmeticException when n less than 0
     */
    //
    // check parameter and throw exception if n < 0. Don't worry about
arithmetic overflow.
    public int fRec(int n) throws ArithmeticException {
        if (n < 0) {
            throw new ArithmeticException("n less than 0");
        }
        if (n == 0) {
            return f0;
        } else if (n == 1) {
            return f1;
        } else {
            return fRec(n - 1) +fRec(n - 2);
        }
    }

    // instance variables store F(0) and F(1):
    public int f0;
    public int f1;
}
```

2

Greeter.java

```
/**
 * A class for producing simple greetings.
 */

public class Greeter {
   private String name;

   /**
    * Constructs a Greeter object that can greet a person or
    * entity.
    *
    * @param aName the name of the person or entity who should
    *              be addressed in the greetings.
    */
   public Greeter(String aName) {
      name = aName;
   }
```

```
    /**
     * Greet with a "Hello" message.
     *
     * @return a message containing "Hello" and the name of
     * the greeted person or entity.
     */
    public String sayHello() {
        return "Hello, " + name + "!";
    }

    /**
     * Swaps out the member "name" between the current Greeter object and the
other Greeter object passed in.
     *
     * @param other Greeter object that will be used to swap out the name
     */
    public void swapNames(Greeter other) {
        String tempName = other.name;
        other.name = this.name;
        this.name = tempName;
    }

    /**
     * Creates a new Greeter object with its name being the qualifier string
followed by
     * " " and the executing greeter's name (i.e. this.name).
     * For example:
     * Greeter g = new Greeter("world");
     * Greeter g2 = g.createQualifiedGreeter("beautiful");
     *
     * @param qualifier The string used in the creation of the new Greeter object
     * @return New Greeter object with the new name
     */
    public Greeter createQualifiedGreeter(String qualifier) {
        return new Greeter(qualifier + " " + this.name);
    }
}
```

GreeterTester.java

```
import org.junit.Assert;
import org.junit.Test;

public class GreeterTester {
    /**
     * This main method will test the createQualifiedGreeter & swapNames methods
     * <p>
     * For createQualifiedGreeter one Greeter object will be created with a name,
then the createQualifiedGreeter
     * method will called with string. Then the sayHello will be used to ensure
that we have added the new string before
     * the name. Some caveat is since name is private we need to use sayHello to
read the name from the built string.
     * that means the string test has the "Hello,.*!" built in.
     * <p>
```

```
    * For the swapNames we take a similar approach to createQualifiedGreeter as
we will use the sayHello to test the name
    * 2 Greeter objects will be created with different names. Then one of the
objects will call swapNames with passing
    * in the other Greeter object. To ensure the names are swapped, the sayHello
values are tested.
    *
    * @param args Not expected to use any incoming arguments
    */
   public static void main(String[] args) {

      Greeter worldGreeter = new Greeter("World");
      Greeter helloWorldGreeter = worldGreeter.createQualifiedGreeter("Hello");
      if (helloWorldGreeter.sayHello().equals("Hello, Hello World!")) {
         System.out.println("createQualifiedGreeter passes");
      } else {
         System.out.println("createQualifiedGreeter Fails");
      }

      Greeter jamesGreeter = new Greeter("James");
      Greeter adamGreeter = new Greeter("Adam");
      jamesGreeter.swapNames(adamGreeter);

      if (jamesGreeter.sayHello().equals("Hello, Adam!") &&
adamGreeter.sayHello().equals("Hello, James!")) {
         System.out.println("swapNames passes");
      } else {
         System.out.println("swapNames Fails");
      }

   }

   /**
    * Using JUnit to create 1 Greeter object and call the createQualifiedGreeter
to create a new Greeter object.
    * This new Greeter object is tested using the sayHello with the
Assert.assertEquals method
    */
   @Test
   public void createQualifiedGreeter_Test() {
      Greeter worldGreeter = new Greeter("World");
      Greeter helloWorldGreeter = worldGreeter.createQualifiedGreeter("Hello");
      Assert.assertEquals(helloWorldGreeter.sayHello(), "Hello, Hello World!");
   }

   /**
    * Using JUnite to create 2 Greeter objects, then using swapNames with these
2 objects. To test swapNames by using
    * sayHello on both objects to test the returned string that names are
swapped
    */
   @Test
   public void swapNames_Test() {
      Greeter jamesGreeter = new Greeter("James");
      Greeter adamGreeter = new Greeter("Adam");
      jamesGreeter.swapNames(adamGreeter);

      Assert.assertEquals(jamesGreeter.sayHello(), "Hello, Adam!");
```

```
        Assert.assertEquals(adamGreeter.sayHello(), "Hello, James!");
    }
}
```

`

3

DataAnalyzer.java

```
import java.util.LinkedList;

/**
 * The DataAnalyzer class is expected to compute some simple statistics given a
list of integers.
 */
public class DataAnalyzer {
    private LinkedList<Integer> numList;

    DataAnalyzer(LinkedList<Integer> list) {
        numList = list;
    }

    /**
     * This method will loop over the numList trying to find the smallest number
which will then be returned
     *
     * @return min number in numList
     */
    public Integer min() {
        Integer minVal = Integer.MAX_VALUE;
        for (Integer num : numList) {
            if (num < minVal) {
                minVal = num;
            }
        }
        return minVal;
    }

    /**
     * This method will loop over the numList trying to find the largest number
which will then be returned
     *
     * @return max number in numList
     */
    public Integer max() {
        Integer maxVal = Integer.MIN_VALUE;
        for (Integer num : numList) {
            if (num > maxVal) {
                maxVal = num;
            }
        }
        return maxVal;
    }
```

```
    /**
     * This method will loop over numList to compute the total, divide by the
size to compute the average which will
     * then be return
     *
     * @return average of the numList
     * @throws ArithmeticException when the list is empty
     */
    public Double average()throws ArithmeticException  {
        if (numList.size() != 0) {
            Integer total = 0;
            for (Integer num : numList) {
                total += num;
            }
            return total / (double) numList.size();
        }
        else{
            throw new ArithmeticException("The list is empty. Please be sure to
enter a list first");
        }
    }
}
```

DataAnalyzerTest.java

```
import java.io.FileWriter;
import java.io.IOException;
import java.util.LinkedList;
import java.util.Scanner;

class DataAnalyzerTest {
    /**
     * a. reads from the terminal a sequence of numbers (integers)
     * b. saves them to a file with the name given from the command line
     * c. calculates,then displays on the terminal, and also saves to that file
     * the maximum,  minimum, and average.
     *
     * @param args expect the file name to be entered in args[0]
     */
    public static void main(String[] args) throws IOException {

        // Retrieve the filename
        String fileName = "";
        if(args.length != 1 || (args.length == 1 && !args[0].contains(".txt"))){
            System.out.println("ERROR: Expected to have \".txt\" in the first
argument");
            System.exit(0);
        }else{
            fileName = args[0];
        }

        //Parse the integers from the scanner
        Scanner sc = new Scanner(System.in);
        String input = "";
        System.out.println("Please a list of numbers separated by spaces");
        input = sc.nextLine();
```

```
        System.out.println(input);
        String[] list = input.split(" ");

        if (list.length < 1) {
            System.out.println("Expecting at a list of integers and a file name.
At minimum 1 int. Exiting");
            System.exit(0);
        }

        LinkedList<Integer> inputList = new LinkedList();

        try {
            for (int i = 0; i < list.length - 1; i++) {
                Integer num = Integer.parseInt(list[i]);
                inputList.add(num);
            }
        } catch (Exception e) {
            System.out.println("ERROR: Parsing integers error. Please be sure to
enter integers as inputs. Exiting");
            System.exit(0);
        }

        //Compute stats, output to screen and file.
        DataAnalyzer data = new DataAnalyzer(inputList);
        FileWriter out = new FileWriter(fileName);
        out.write("Max: " + data.max().toString() + "\n");
        out.write("Min: " + data.min().toString() + "\n");
        out.write("Average: " + data.average().toString() + "\n");
        out.close();

        System.out.println("Max: " + data.max().toString());
        System.out.println("Min: " + data.min().toString());
        System.out.println("Average: " + data.average().toString());
    }

}
```

`

4

The answer should be 11. The reason is the != comparison is comparing the address values which is not desired.Then g2 gets set to null. Then x gets set to 1, and then when g2 calls sayHello we get a NullPointerException which sets x to 10. Finally will then be run and incrementing x by 1 which will make x 11. There are a few ways to correctly compare strings and here is one for example g1.sayHello().equals(g2.sayHello())

5

PrimeFactorizer.java

```
import java.util.ArrayList;

public class PrimeFactorizer {
```

```
    private int n = 0;
    private ArrayList<Integer> internalPrimes = new ArrayList();
    private ArrayList<Integer> internalExponents = new ArrayList();

    /**
     * Initialize the object with target number n.
     */
    public PrimeFactorizer(int n) {
        this.n = n;
    }

    /**
     * Return n, the target number.
     */
    public int getN() {
        return n;
    }

    /**
     * Compute factorization. Do not repeat operation if it was called before.
     * Algorithm:
     * The objective here is to find all the prime numbers that multiple
together to to reach n. To do this
     * we will start with 2 and try to find if it can evenly divide into N with
only hole numbers remaining.
     * When then happens we will save that number off and reset this divisor
back to 2 and try over again.
     * When the number doesnt meet the division criteria then this divisor gets
incremented by 1.
     */
    public void compute() {

        //Start with 2 and keep dividing until we have whole numbers. then
restart at 2 again
        int left = n;
        while (left > 0) {
            for (int i = 2; i <= left; i++) {
                //Check to see if we have ant decimal places. If no decimal
places then we have found a valid number
                double test = ((double) left / (double) i);
                if (test % 1 == 0) {
                    //Corner case where we have come to the end of the for loop
and we can evenly divide whats left
                    // meaning we have found the last possible divider
                    if (left == i) {
                        addPrimeToList(left);
                        left = 0;
                    } else {
                        addPrimeToList(i);
                        left = left / i;
                    }
                    break;
                }
            }
        }
    }
```

```
    /**
     * This helper function is used to effectively store the factors and
ensuring that the exponents have been correctly
     * set or incremented.
     *
     * @param factor prime factor to be added to lists
     */
    private void addPrimeToList(int factor) {
        int primeIndex = internalPrimes.indexOf(factor);
        if (primeIndex == -1) {
            internalPrimes.add(factor);
            internalExponents.add(1);
        } else {
            internalExponents.set(primeIndex, internalExponents.get(primeIndex)
+ 1);
        }
    }


    /**
     * Return the factors and exponents in two arraylists. Call compute() first,
if necessary.
     * For instance, if n=60, primes=[2,3,5], and exponents=[2,1,1].
     * This function overwrites the 'n' parameter passed to the constructor.
     *
     * @param n the number that is requested to find the prime factorization
     * @param primes list of primes that can evenly be multiplied to create N.
This list is populated and returned
     * @param exponents list of exponents that that would be used in conjunction
with the primes list to create N.
     *                  This list is populated and returned
     */
    public void getFactorsAndExponents(int n, ArrayList<Integer> primes,
ArrayList<Integer> exponents) {
        //Only compute if n is different or if Compute hasnt happen yet
        if (n != this.n || this.internalPrimes.size() <= 0) {
            this.n = n;
            internalPrimes.clear();
            internalExponents.clear();
            compute();
        }
        primes.addAll(this.internalPrimes);
        exponents.addAll(this.internalExponents);
    }


    /**
     * Return a string with the "pretty" representation of the prime
factorization.
     * For instance, if n is 60, then toString() for the PrimeFactorizer(60)
object
     * should be "60 = 2^2*3*5". Call compute() if not done before.
     */
    public String toString() {
        StringBuilder primeFactorStr;
        primeFactorStr = new StringBuilder();
        primeFactorStr.append(n).append(" = ");
        for (int i = 0; i < internalPrimes.size(); i++) {
```

```
            primeFactorStr.append(internalPrimes.get(i).toString());
            if (internalExponents.get(i) > 1) {

 primeFactorStr.append("^").append(internalExponents.get(i).toString());
            }

            if (i != internalPrimes.size() - 1) {
                primeFactorStr.append("*");
            }

        }
        return primeFactorStr.toString();
    }

    // other code, helper functions, etc.
}
```

PrimeFactorizerTest.java

```
import java.util.ArrayList;
import java.util.Scanner;

class PrimeFactorizerTest {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Integer input = null;
        try{
            System.out.print("Please enter an integer to compute the Prime
Factorization: ");
            input = sc.nextInt();
        }catch (Exception e) {
            System.out.println("ERROR: Parsing input error. Please be sure to
enter integers as inputs");
            System.exit(0);
        }
        PrimeFactorizer pf = new PrimeFactorizer(input);
        pf.compute();
        System.out.println(pf.toString());
        System.out.println("---");
        try{
            System.out.print("Please enter an integer to exercise
getFactorsAndExponents: ");
            input = sc.nextInt();
        }catch (Exception e) {
            System.out.println("ERROR: Parsing input error. Please be sure to
enter integers as inputs");
            System.exit(0);
        }
        ArrayList<Integer> p = new ArrayList();
        ArrayList<Integer> e = new ArrayList();
        pf.getFactorsAndExponents(input,p,e);
        System.out.println(pf.toString());
        System.out.println("Primes: ");
        for (Integer integer : p) {
            System.out.print(integer + " ");
        }
```

```
        System.out.println("\nExponents");
        for (Integer integer : e) {
            System.out.print(integer + " ");
        }
    }
}
```