Recursion and Tower of Hanoi

Overview

For this assignment, you will be implementing a few Recursive Algorithms. In addition to this, you will be implementing a Tower of Hanoi puzzle game.
Due: February 21st 2014 @ 5PM

Files to complete

You are expected write an implementation for each of the classes listed in **config.Configuration**. You may call them anything you like but they must meet the requirements stated in the comments of that file (as well as any addition restrictions mentioned in the next sections).. In addition to this, you **MUST** specify which class we should use in the **config.Configuration** class; you will see an example of this for the **BasicMath** interface in the **config.Configuration** class.

In addition to this, you **MUST** include a String field in the **config.Configuration** file with your UMass student ID number. This is the 8 digit value found on your student ID card and is used to identify you in moodle. This will allow us to get your grade and feedback to you quickly.

It may be useful for you to write additional class files. Any class file you write that is used by your solution **MUST** be in the provided **src** folder. When we test your assignment, all files not included in the **src** folder will be ignored.

Note: You may not use any of the provided Collection elements provided by the java standard API.

Test files

In the test folder, you are provided with several JUnit test cases that will help you keep on track while completing this assignment. We recommend you run the tests often and use them as a checklist of things to do next. You are not allowed to modify these files. If you have errors in these files, it means the structure of the files found in the **src** folder have been altered in a way that will cause your submission to lose points.

Support Code API

The Support Code's comments have been generated into a nicely formatted API that can be found here: http://people.cs.umass.edu/~jcollard/cs187/s14/assignments/4/doc/

It is highly recommended that you spend a day simply reading over the comments in each of the interfaces and classes provided:

algorithms.BasicMath - An interface with several math functions

hanoi.HanoiBoard - An interface representing a 3 pegged Tower of Hanoi puzzle
hanoi.HanoiMove - A class representing a move on a **HanoiBoard**
hanoi.HanoiPeg - An interface representing a peg in a Tower of Hanoi puzzle
hanoi.HanoiRing - A class representing a ring in a Tower of Hanoi puzzle
hanoi.HanoiSolution - An interface representing a solution for a Tower of Hanoi puzzle
hanoi.HanoiSolver - An interface representing a strategy for solving the Tower of Hanoi puzzle.
hanoi.IllegalHanoiMoveException - An exception that is thrown if an illegal move is attempted

structures.ListInterface - A List interface specifying the minimum requirements to test your solution.

Part One: Importing Project into Eclipse

Begin by downloading the provided project from Moodle and importing it into your workspace.

You should now have a project called **recursion-student** it is very important that you do not rename this project as it is used during the grading process. If the project is renamed, your assignment may not be graded.

By default, your project should have no errors and contain the following root items:
src - The source folder where all code you are submitting must go. You can change anything you want in this folder, you can add new files, etc...
support - This folder contains support code that we encourage you to use (and must be used to pass certain tests). You are not allowed to change or add anything in this folder
test - The test folder where all of the public unit tests are available
JUnit 4 - A library that is used to run the test programs
JRE System Library - This is what allows java to run

If you are missing any of the above or errors are present in the project, seek help immediately so you can get started on the project right away.

Part Two: Basic Recursion Warmup

Start off by creating a class that implements the **algorithms.BasicMath** interface (an example of this is in the **src** folder by default). Once you have created this class, be sure to specify that you want to use it in the **config.Configuration** class file in your **src** folder.

Note: For your implementation in this class, you may not use any loops. The private test will be smart enough to detect this.

isEven and isOdd

For your implementation of the isEven and isOdd methods, you may not use the % operator.

Hint:
What does isEven(0) return? What does isEven(1) return?
What does isEven(2) return? What does isEven(3) return? 4? 5? 6?
Is there a way to reduce n such that we get to one of these?
What about negative values?

sumN

For your solution, you may not use the * or / operator.

Hints: Read about factorial at the end of section 4.1. The solution to this one is pretty similar!

Factorial

Read about factorial at the end of section 4.1

Note: A bug in the specification was found stating that you should throw an IllegalArgumentException for values less than 1. This has been changed to say less than 0. However, when you submit your assignment, you will **NOT** be tested on factorial(0) as this was a mistake in the assignment.

biPower

For your solution, you may not use the **Math** class provided in **java.lang**.

Hints: This one is also very similar to factorial! Take a look at the non recursive solution in **algorithms.BasicMathExamples**

Part Three: Implement ListInterface

Take a mental break from recursion and work on something you should be quite familiar with Lists!
You will need to implement the **structures.ListInterface** provided. Make sure to read over the comments carefully; you are being tested on how well you implement the stated contract.

You will notice that the **size()** and **append()** methods require that they execute in O(1) time. This means you will not pass the tests if you iterate down your links to calculate the size / append. You will have to find another way to do this.

You are now adding a remove method. The method says it should take O(n) time. That means, you will be iterating to the specified location, doing some patch work on your list and then returning the element.

Note: Don't forget to specify the implementation you are using in the **config.Configuration** class!

Part Four: Implement the Tower of Hanoi Framework

In this section, you will be implementing a framework for the creating a Tower of Hanoi puzzle game. The Tower of Hanoi puzzle is presented in Chapter 4.3. There is also a pretty good wikipedia article here: http://en.wikipedia.org/wiki/Tower_of_Hanoi

Start by reviewing how the game works and then reading the interfaces / classes provided. Draw some pictures to see how each of the classes relates to one another before you start. This will help you become familiarized with their functionality.

We recommend you complete your implementations in this order: **HanoiPeg, HanoiBoard, HanoiSolver**

HanoiPeg

Does this class look familiar in some way? Perhaps you can utilize some code you've written previously to implement it.

HanoiBoard

This class should be pretty straight forward to implement. Imagine if you built an actual Tower of Hanoi puzzle. How would you interact with it in real life?

HanoiSolver

This is by far the hardest part of this assignment. Perhaps, it even deserved its own Part Five section! You may not use loops in your implementation (I don't think you want to anyway).

In this section, you must implement an algorithm to solve the Tower of Hanoi puzzle. You should definitely read (and then re-read) section 4.3 of the book. Make sure you understand how to solve the puzzle in and out.

You will notice that the solve method actually returns an interface called HanoiSolution. You will have to implement this class to complete this section. This class is simply a wrapper containing a value representing how one should set up the Tower of Hanoi puzzle and then a List of moves to make to complete the puzzle.

Best of Luck!

Note: Make sure you specify which implementation you want us to test in the **config.Configuration** class!

Part Five: Export Project and Submit

When you have finished your solution and are ready to submit, export the entire project. Be sure that the project is named **recursion-student**. Save the exported file with the zip extension (any name is fine).

Log into Moodle and submit the exported zip file.z