From: **Alan Liang** alan.5100@yahoo.com
Subject: assignment07
Date: October 30, 2016 at 6:52 PM
To: Winnie Liang winnieliang27@yahoo.com

AL

Assignment 7

Due Tuesday November 1 at 11:59pm

Download the file Nim.zip from Blackboard Content->Code->Assignment 7 Game of Nim.

This is simple illustration of the Strategy Design Pattern. Change line 20 of the class Nim and you chance the behavior of your computer opponent in the game. Each behavior is an algorithm that implements the interface Play in the file Play.java.

What is the game? There are N sticks on the board. When it is the player's turn, the player picks up between 1 and N/2 sticks, or the remaining stick if there is only 1 left. The expression N/2 is whole-number division. The game is described in the textbook as Problem P6.6.

Implement all the strategy classes in different ways. `Frugal` is correctly implemented. Modify `Greedy` to always return N/2, which means `currentState/2` if `currentState > 1` and to return 1 if `currentState` is 1. Modify `Guess` to return a random value between 1 and `currentState/2` (**INCLUSIVE**) when `currentState > 1` and to return 1 if `currentState` is 1. Modify Smart as described in the textbook in P6.6--a simple while loop will generate the values 2, 2*2, 2*2*2, 2*2*2*2 until you pass (N+1)/2 and then that number minus 1 is what you want to ***leave*** on the board. This only fails if N is itself a power of 2 minus 1, in which case the optimal choice to leave on the board is equal to N (so you pick up 0--not allowed) and so Smart picks randomly, the way Guess does.

Play the game by running Nim.

==========

There is probably a known algorithm to systematically solve this problem but all I found from a quick look is a reference to https://en.wikipedia.org/wiki/Ford%E2%80%93Fulkerson_algorithm from http://softwareengineering.stackexchange.com/questions/311829/how-to-match-up-courses-with-requirements
I thought we could extend the undergraduate class from Assignment 6 to include general education analysis. I am "*only*" going to look at the requirement for CS!! I think this brute force works but it may need corrections as the week goes by!

The Gened requirement for CS is to complete the A, C, F1, G, H, L, M, N, O, P, S, Y requirements. B gives both S and Y, J gives both C and O. Course with C, O, J, or F1 can satisfy that requirement as well one of the other Gened requirements. Otherwise only one Gened can be used from any specific course. The Degree Audit will make choices for you to get the most Gened's out of the course you take. We are going to see how that could be done. It is a challenge.

The reason that we are making this CS specific is that Computer Science and Nursing only require one semester of foreign language (F1). Represent Geneds with an enum:

```
package assignment07;
public enum CSGened {
    A, B, C, F1, G, H, J, L, M, N, O, P, S, Y
}
```

NOTE: to test for equality of enums, we use ==, since they are singletons. For example, there can only be one `CSGened.A` object in the whole program.

You need all the classes concerning Course and Student form assignment06. We can import what we need using `import assignment06.Course;` and so on.

However we need a subclass

```
package assignment07;
import assignment06.UndergradCourse;
public class CSGenedCourse extends UndergradCourse {
// has a field that is an array of CSGened called geneds
// provide a constructor that has an arry as a paramter to set the field
// and a getter method for the array field
}
```

Go back to the class assignment06.Student class with a field
`private ArrayList<Course> allCourses = new ArrayList<>();`
with its own *getter* method

Copy assignment06 UndergraduateStudent over to assignment07, this is where all the work is.

```
package assignment07;
```

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

import assignment06.Course;
import assignment06.Student;

public class UndergradStudent extends Student {
        public boolean isUndergrad() {
                return true;
        }
...
```

Simply stated the challenge is to write a method `boolean satifiesCSGened()` that returns `true` if the course in `getAllCourses()` combine to satisfy the general education requirements.

Also we need a Driver to test that `satifiesCSGened` does what it should.

AN OUTLINE OF HOW I WROTE A BRUTE FORCE ALGORITHM.

The idea is that an UndergradStudent has taken a set of all courses taken (obtained by calling `getAllCourses()`). Among these, some are `Gened`. We need find a bunch of sets of `CSGeneds` to collect all the different possible combinations of Geneds that are possible (apart from C, O and F1, only one Gened can count at a time). After collecting all the sets, we go through them to see if one set contains all possible geneds.

To have some way to see what the method is doing we can have it print to a text file as it progresses. As we go along, keep in mind that the following are possible `CSGenedCourses` in the student's record:

```
UndergradStudent ugstu = new UndergradStudent();
ugstu.getAllCourses().add(new CSGenedCourse(new CSGened[] {CSGened.B}));
ugstu.getAllCourses().add(new CSGenedCourse(new CSGened[] {CSGened.H}));
ugstu.getAllCourses().add(new CSGenedCourse(new CSGened[] {CSGened.C, CSGened.P, CSGened.N}));
ugstu.getAllCourses().add(new CSGenedCourse(new CSGened[] {CSGened.G, CSGened.C}));
ugstu.getAllCourses().add(new CSGenedCourse(new CSGened[] {CSGened.G}));
```

There will be a few redundant sets.

For the method `boolean satifiesCSGened()` the outer structure is:

```
try (PrintWriter out = new PrintWriter("test.log")) {

   ...// lots of code coming here

       out.close();
} catch (FileNotFoundException e) {
       e.printStackTrace();
}
return false;
```

In the `...` make an array list of `Set<CSGened>`, instantiated to a new array list, I called my list `waysToSatisfy`. Add a single new `HashSet` of `CSGened` to `waysToSatisfy`. Introduce three `booleans Csatisfied`, `Osatisfied`, `F1satisfied` set to `false`.

Next there is an outer for loop: `for(Course crs : getAllCourses())`, if crs is a Gened course--use `if(crs instanceof CSGenedCourse)`--cast crs to a gened course, so we can call `getGeneds()`: CSGenedCourse gecrs = (CSGenedCourse)crs;Call `out.println(Arrays.toString(gecrs.getGeneds()));` make the variables `sz` and `len` equal to `waysToSatisfy.size();`and `gecrs.getGeneds().length;` respectively.

At this point we will assume that we have already accumulated a list of sets of CSGeneds that can be extracted from the courses we already looked at. If len > 1, then for each CSGened in gecrs we have to add it to a *separate copy* of the sets we have so far, since only one gened can be used (e.g. on one of P or N). This is where some redundancy creeps in, since we will make copies for any C, O, J, or F1, since that will be the easiest way to proceed.

How to make the copies: use the nested loops `for(int i = 0; i < len-1; i ++) { for(int j = 0; j < sz; j++) ...` The loop body makes a copy of the j-th set in waysToSatisfy: (i) make a new CSGened HashSet, (ii) populate the HashSet by calling `addAll(waysToSatisfy.get(j))` on the HashSet you just made, (iii) add this new HashSet to `waysToSatisfy.` Note we used `<tt> i < len-1`, since no copying is needed if `len` is 1.

Just to see what happened put in the line `for(Set<CSGened> s : waysToSatisfy) out.println(s);`

Now we add each of the Geneds individually to our previous sets. First let's explain: Suppose we had previously build 5 possible sets of geneds `waysToSatisfy.get(0)`, through `waysToSatisfy.get(4)` and suppose the next course has the geneds `CSGened.C`, `CSGened.P`, `<tt>CSGened.N`. Since `len` is 3, we made 2 copies of the elements of `waysToSatisfy` in the nested loop above to get 15 sets `waysToSatisfy.get(0)`, through `waysToSatisfy.get(14)`. We plan to add `CSGenEd.C` to the first 5, `CSGenEd.P` to the next 5, and `CSGenEd.N` to the last 5.

Note we not actually going to add any `CSGenEd.C`, `CSGenEd.O`, `<tt>CSGenEd.J`, or`<tt>` `<tt>CSGenEd.F1` to any sets, since they can double count with another gened but it is too much work to eliminate this source of redundancy in the list`<tt><tt>` `waysToSatisfy`.

This is a nested loop, the first is for `i` from 0 to `len-1` and the inner loop take j from `i*sz` to `(i+1)*sz-1` (check in your mind that if `len` is 3 and `sz` is 5, then the inner loops are form 0 to 4, 5 to 9, and 10 to 14). Inside the loop:
if `gecrs.getGeneds()[i] == CSGened.C`, just change `Csatisfied` to `true`
else if `gecrs.getGeneds()[i] == CSGened.O`, just change `Osatisfied` to `true`
else if `gecrs.getGeneds()[i] == CSGened.J`, just change both `Csatisfied` and `Osatisfied` to `true`
else if `gecrs.getGeneds()[i] == CSGened.F1`, just change `F1satisfied` to `true`
else add `gecrs.getGeneds()[i]` to `waysToSatisfy.get(j)`

Just to see what happened put in the line `for(Set<CSGened> s : waysToSatisfy) out.println(s);`
Note that C, O, J and F1 are not added to any sets.

That ends the outer loop `for(Course crs : getAllCourses())`

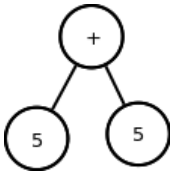
Cleaning up B's and making a decision:
For each set `s` in `waysToSatisfy`, if `s.contains(CSGened.B)`, remove `CSGened.B` from `s` and add `CSGened.S` and `<tt>CSGened.Y`.
In the same loop keep track of the current set and the C, O and F1:
`out.println(s + " " + Csatisfied + " " + Osatisfied + " " + F1satisfied);`
Finally if the size of `s` is 9 (NINE: A, G, H, L, M, N, P, S, Y) and `Csatisfied`, `Osatisfied`, and `F1satisfied` are all true, return true out of the method.

Add more gened courses to the the undergraduate students record so you can check true and false return values from the `satisfiesCSGened` method.

=========
[This part will likely make more sense after Thursday's lab]

This last part of assignment07 will deal with recursion, specifically for evaluating simple arithmetic expressions. No for/while loops are allowed, just recursion. Points will be deducted for failure to comply.

For instance, the simple expression *5 + 5* can be viewed as a tree where the add operation is the parent node of each 5 node.



To evaluate the result of 5 + 5, we would sum the results of evaluating the two subexpressions (both 5), to get 10. This is recursion: solving a problem from the solutions to smaller problems. This can scale to other operations like multiplication, division, negation, and so on. Negation over integers evaluates to 1 if the integer is 0 or 0 if the integer is non-zero. If we represented the addition as a class `Add`, we could construct the addition expression like so

```
new Add(new Num(5), new Num(5));
```

where add accepts two arguments, the left and right expressions. However, we want to allow arbitrarily complex expressions like *(4 * 2 - 1) + 5*. To handle this, the operations should accept any expression, not just `Nums`. The same goes for any other operations, except `Nums`. `Nums` just represent numbers, they cannot have subexpressions. Therefore, the result of evaluating a Num is just the integer it holds.

Here is a JUnit tester class with several tests. Read these tests carefully to understand how the tests relate to the above discussion. From what the tests use, you can know what classes and methods you need to create. This portion of the assignment will be complete when the code compiles and all the tests pass. There is not a lot of code in this portion, but it will require some thinking on your part.

One hint is to make the `Expr` class abstract with one abstract method.

Put the following in ExprTester.java

```
package assignment07;

import static org.junit.Assert.*;
import org.junit.Test;

public class ExprTester {
```

```
    @Test
    public void simpleAdd() {
        assertEquals(10,
                     new Add(
                         new Num(5),
                         new Num(5)
                         ).eval());
    }

    @Test
    public void simpleNeg() {
        assertEquals(0, new Neg(
                         new Num(5)
                         ).eval());
        assertEquals(1, new Neg(
                         new Num(0)
                         ).eval());
        assertEquals(0, new Neg(
                         new Num(1)
                         ).eval());
    }

    @Test
    public void complexAdd() {
        assertEquals(26,
                     new Add(
                         new Add(
                             new Num(1),
                             new Add(
                                 new Num(10),
                                 new Num(1)
                                 )
                             ),
                         new Add(
                             new Add(
                                 new Add(
                                     new Num(3),
                                     new Num(3)
                                     ),
                                 new Num(7)
                                 ),
                             new Num(1)
                             )
                         ).eval());
    }

    @Test
    public void complexMixed() {
        Expr a = new Add(
            new Add(
                new Num(1),
                new Add(
                    new Num(10),
                    new Num(1)
                    )
                ),
            new Add(
                new Add(
                    new Add(
                        new Num(3),
                        new Num(3)
                        ),
                    new Num(7)
                    ),
                new Num(1)
                )
            );
        Expr b = new Mul(
            new Add(
                new Num(2),
                new Add(
                    new Num(3),
                    new Mul(
                        new Num(2),
                        new Num(3)
```

```
                    )
                ),
            new Mul(
                new Neg(
                    new Num(0)
                    ),
                new Num(10)
                )
            );
        Expr c = new Mul(
            new Neg(
                new Neg(
                    new Num(1)
                    )
                ),
            new Num(10)
            );
        assertEquals(146,
                     new Add(
                         new Add(a,
                                 b),
                         c
                         ).eval());
    }
}
```

assignment7

Updated 4 hours ago by Leslie Lander and Matthew Hems

followup discussions *for lingering questions and comments*

Resolved
Unresolved

Anonymous 5 days ago
I wrote an implementation of the Ford-Fulkerson algorithm based off the wiki page, should I scrap it and use your way?

Leslie Lander 5 days ago No that is great. test it and check that it works

Reply to this followup discussion

Resolved
Unresolved

Anonymous 3 days ago
If we have one course that satisfies all the geneds (obviously unrealistic) should it pass the test? With the code that you've described it doesn't work for me.

Anonymous 3 days ago Oh, and is it okay to put each part of this assignment in a sub-package? e.g. assignment07.part1 etc

Matthew Hems 3 days ago A course that offers the geneds can contribute a maximum of two of those geneds to the student's graduation requirements. If a course has the geneds {A, G, M, N}, only one of those can count towards the degree. If a course has the geneds {A, C, N}, the C and one of {A, N} can count to the degree. So no, one course with every gened should not satisfy the requirements.

Sub-packages are fine, just make a note of it when you submit to blackboard.

Leslie Lander 3 days ago I am on the committee that oversees approval of new geneds (university undergraduate

curriculum committee) and there is a limit of 3 on the number of geneds a course can have, although this should not have any impact on the code we write.

Reply to this followup discussion

Resolved
Unresolved

Anonymous 3 days ago
Doesn't CSGenedCourse need a string passed to its constructor in order to pass a string when calling super() for UndergradCourse?

Anonymous 3 days ago And similarly for UndergradStudent calling the constructor of Student?
-For this part I was confused whether we are supposed to copy and paste our UndergradStudent class from assignment06, or to copy and paste the code for UndergradStudent on the piazza post above?

Anonymous 3 days ago You can just pass a hardcoded string into super(), because you will never see the title of any course.

Matthew Hems 3 days ago You are copying the UndergradStudent class from assignment06 and tweaking it.

Anonymous 1 hour ago So it is ok to hardcode a string into the super constructor?

Reply to this followup discussion

Resolved
Unresolved

Anonymous 3 days ago
"In the `...` make an array list of `Set<CSGened>`, instantiated to a new array list, I called my list `waysToSatisfy`. Add a single new `HashSet` of `CSGened` to `waysToSatisfy.`"

Is this what the statement is saying:
Set<CSGened> waysToSatisfy = new HashSet<CSGened>();

Anonymous 3 days ago

```
...make an array list of Set<CSGened>
```

If you want to make an arraylist of Set<CSGened>, its type is ArrayList<Set<CSGened>>. What you have now is just a single Set<CSGened>, not any type of collection type that takes them.

Anonymous 1 day ago Do we use the ".addAll" method when adding the single HashSet to the ArrayList?

Anonymous 1 day ago ignore the above post, i figured it out.

Reply to this followup discussion

Resolved
Unresolved

Anonymous 3 days ago Thanks, I completely misread it.

Reply to this followup discussion

Resolved
Unresolved

Anonymous 20 hours ago
If "Neg" is an expression on our tree, it should only have one node

If "Neg" is an expression on our tree, it should only have one node stemming from it, correct? Because you'd only be negating one number or expression.

If so, do we make an assumption that the node stemming from a negation will either be left or right?

Or if there happens to be two nodes stemming from a negation should we return some kind of error?

Anonymous 20 hours ago Pretty sure Neg should only take one Express in its constructor. I might be wrong as I'm very fucked up right npw

Leslie Lander 4 hours ago You can make it the left child or the right child or even have its own expression field with null left and right children.
Be liberal in making constructors--whatever works well.

Anonymous 1 hour ago Does "Neg" just represent if the expression (e.g. 5 + -5) resulted in 0, return 1 otherwise the expression resulted in a non-zero value?

Leslie Lander 41 minutes ago Neg has nothing to do with the "-" negation sign. As an analogy, the C language allows you to use numbers as if they are true/false and Neg is related to the ! boolean operation.

Reply to this followup discussion

Resolved
Unresolved

Anonymous 1 hour ago
When I created the project in Eclipse, I got an error message when I tried to import from assignment06. I added assignment06 to the build path and the error went away. So, if I submit assignment07, will the code bomb when run on another system?

Leslie Lander 43 minutes ago The two packages assignment06 and assigment07 have to live in the same Project. Eclipse allows you to cut an past packages to achieve that.

Nim.zip