Project 1 - Due

Mar 27, 2016 11:59 PM

CMSC 335 7980 Object-Oriented and Concurrent Programming (2162)

The more your use this conference, the more you will enjoy and learn in this class!

* See sample data file below.
* You are encouraged to add entries and share them with the rest of the class.
* You can also used the attached jar file to create other (much larger) data files.
* The source code is included in the jar file, which you should find helpful in your own programs for this class.

NOTE: Your instructor will have many suggestions for how to implement this program, but you should think about the program yourself and try some stuff before your instructor just overwhelms you with ideas and suggestions. Thus, you should think about the program and its requirements, try some code, and then  you will find that you have questions. You should post questions about clarifying what your instructor has in mind, as well as asking for comments about your code, as replies to THIS conference. That way, everyone can benefit from both the questions and the answers.

We want to emphasize that if you have a question or problem, there is an excellent chance that others are having, or will shortly have, the same issue.

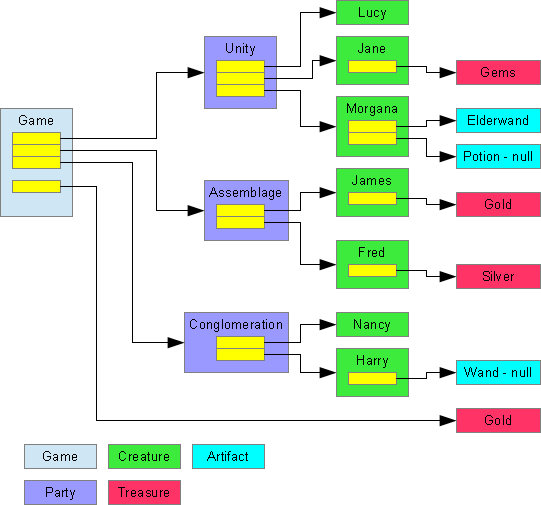
Implement a program to:

* use a GUI interface to control and display the results of the program,
* read a text data file,
* instantiate appropriate classes,
* create an the specified internal (multi-tree) data structure,
* display the structure in a convincing way using a JTextArea, and
* search the structure.

The project will be graded according the criteria for the final project - see below.

Elaboration:

The scenario is inspired by a game call [Sorcerer's Cave](http://en.wikipedia.org/wiki/The_Sorcerer%27s_Cave), and can be represented by the following conceptual diagram:



The yellow boxes represent java.util.ArrayList's.

We have a number of parties, each consisting of a variety of creatures, and each creature may be able to posses some artifacts and carry some treasure. There may also be some creatures specified in the data file that are not members of any party, and artifacts and treasures that may not currently be held by any creature.

Eventually (projects 3 and 4), the creatures will have jobs to perform (threads!), and those jobs will generally take time and require artifacts. The artifacts will be shared resources (project 4), which the creatures will have to share according to some algorithm.

In the game, the focus is on the willingness of creatures to join the party as it enters a cavern, thus the importance of empathy and fear. We will simply use these fields for sorting examples, but you may wish to make this more like a game as we go along.

1. Required data structure - the advanced data structure I want you to implement is a multi-tree with the following levels:
   1. Cave - level 0
   2. Party - Level 1
   3. Creature - Level 2
   4. Artifacts - Level 3
   5. Treasures - Level 3
2. Use at least the following classes, you are free to define others in the spirit of this scenario:  
   1. Cave
      * ArrayList <Party> as an instance variable data structure
   2. Party - index, a name, and list of a number of creatures (accessed by links to instances of the Creature class).
      * ArrayList <Creature> as an instance variable data structure
      * Eventually we would like to add a location attribute to this class, but not for any of the projects in this course.
   3. Creature - index, a type, a name, party by index, empathy value, fear value, list of treasures, list of artifacts
      * ArrayList <Treasure> as an instance variable data structure
      * ArrayList <Treasure> as an instance variable data structure
   4. Treasure - index, type, creature by index, weight, value
   5. Artifact - index, type, creature by index, other fields
3. Use the ArrayList class to hold instances of the classes defined in item 1. Be sure to use generics appropriately.
4. Use a text data file with the following format - which you may extend, and may be extended in the later projects. As a minimum, each item should be specified by a single line in the data file. The initial fields should be used during the instantiation of objects and your program should ignore later fields that are not used. For simplicity, let us assume that the fields are separated by colons. The specifications look something like the following, where the angle brackets are not part of the actual data file. I think we can assume that empathy and fear are integers (perhaps negative), while carrying capacity, weight and value will be doubles.
   * p:<index>:<name>
   * c:<index>:<type>:<name>:<party>:<empathy>:<fear>:<carrying capacity>
   * t:<index>:<type>:<creature>:<weight>:<value>
   * a:<index>:<type>:<creature>

Your program should

* work if there are additional fields on some lines,
* ignore blank lines,
* ignore lines starting with a / (as comments),
* accept spaces within names and types, and
* ignore extra spaces around the :'s nicely.

1. All the classes should be provided with appropriate:
   * constructors
   * toString implementations
   * additional methods
   * instance variables connecting the class to appropriate instances of other classes
2. A GUI that will allow the user to do at least the following operations:
   * Select the data file at run time, using JFileChooser starting at dot.
   * Show the internal data structures in a text area
   * Search for an item by index, name or type. For example, the creature whose name is "Jane".
   * Resize nicely in a JFrame.
3. The GUI class(es) should be distinct from the other classes in the program.

**Deliverables**

1. Java source code files
2. Data files used to test your program
3. Configuration files used
4. a well-written document including the following sections:
   1. Design: including a UML class diagram showing the type of the class relationships
   2. User's Guide: description of how to set up and run your application
   3. Test Plan: sample input and ***expected*** results, and including test data and results, with screen snapshots of some of your test cases
   4. (optionally) Comments: design strengths and limitations, and suggestions for future improvement and alternative approaches
   5. Lessons Learned
   6. Use one of the following formats: MS Word doc, docx, OpenOffice odf, pdf, rtf.

Your project is due by 8 am, EST, on the day following the date posted in the class schedule. We do not recommend staying up all night working on your project - it is so very easy to really messed up a project at the last minute by working when one was overly tired.

Your instructor's policy on late projects applies to this project.  
  
Submitted projects that show evidence of plagiarism will be handled in accordance with UMUC Policy 150.25 — Academic Dishonesty and Plagiarism.

**Format**

(The following is from the standard syllabus, but I really don't care about any of these details. Your document should be easy to read and convey the information about your project in a reasonable manner. If you wish to follow a style, I would suggest IEEE or ACM style rather than APA.) Documentation format and length. The documentation describing and reflecting on your design and approach should be written using Microsoft Word, and should be no more than five pages in length and no less than two pages. The font size should be 12 point. The page margins should be one inch. The paragraphs should be double spaced. All figures, tables, equations, and references should be properly labeled and formatted using APA style.

* Code format:
  + header comment block, including the following information in each source code file:
    - file name
    - date
    - author
    - purpose
  + appropriate comments within the code
  + appropriate variable and function names
  + correct indentation

**Grading**

This activity is awarded 15 percent of the total grade in the course. In the grade book, the total number of points will be set to 100. The project elements will be assessed as follows:

|  |  |
| --- | --- |
| Attributes | Value |
| Project design | 20 points |
| Project functionality | 40 points |
| Test data | 20 points |
| Approach documentation | 15 points |
| Approach documentation grammar and spelling | 5 points |
| Total | 100 points |

**Sample data file:**

// sample data file for CMSC 335, Spring 2012  
// Nicholas Duchon  
// Jan 23, 2012  
// Parties format:  
//    p:<index>:<name>  
p : 10001 : Unity  
p : 10002 : Assemblage  
p : 10003 : Conglomeration

// Creatures format:  
//    c:<index>:<type>:<name>:<party>:<empathy>:<fear>:<carrying capacity>  
c : 20001 : Woman   : Lucy   :10001 : 17 : 22 : 20  
c : 20002 : Woman   : Jane   :10001 : 22 : 15 : 25  
c : 20003 : Woman   : Nancy  :10003 : 80 : 44 : 20  
c : 20004 : Man     : James  :10002 : 35 : 13 : 50  
c : 20005 : Troll   : Fred   :10002 : 03 : 05 : 150  
c : 20006 : Warlock : Harry  :10003 : 05 : 06 : 0  
c : 20007 : Witch   : Morgana:10001 : 10 : 15 : 0

// Treasures format:  
//    t:<index>:<type>:<creature>:<weight>:<value>  
// creature = 0 means noone is carrying that treasure at the moment  
t : 30001 : Gold : 20004 : 50 : 2000  
t : 30002 : Gold :     0 : 75 : 5000  
t : 30003 : Gems : 20002 : 10 : 10000  
t : 30004 : Silver : 20005 : 120 : 1000

// Artifacts format:  
//    a:<index>:<type>:<creature>[:<name>]  
a : 40001 : Wand : 20007 : ElderWand  
a : 40002 : Wand : 20006  
a : 40003 : Potion : 20007

**Some more thoughts you might want to consider:**

Generally, the project should be what call "in the Java spirit" - by which I mean defining and using classes, variables and methods appropriately. Some examples:

1. You should ask yourself if it makes sense to instantiate a class more than once. This has implications for which variables should be local to a method, which methods should be instance vs class (static) and  which variables should be instance vs class.
2. For example, if you are going to use a class like a Game class to contain everything, it makes sense to have more than one game, and each game could have its own data file.
3. In a Game, some students have pointed out the there might be artifacts, treasures and creatures that are not part of any party. How should we handle these? I suggest a single data structure to handle the lot - an ArrayList is most appropriate, but that structure is generic, so what kind of generic declaration should we use, perhaps ArrayList <Object>, but that seems like a too general structure since we don't really want stuff like JLabel or JButton this structure. Thus, we are lead to proposing another class, say GameElement, and the other classes we have seen so far, Creature, Treasure and Artifact should extend this class, and then we can use the following declaration: ArrayList <GameElement> to hold stuff that is just lying around.
4. Links? This word is being used as shorthand for references to instances of other objects. To give you some idea of what we have in mind, you might consider the following example, particularly the graphics with it:  
   <http://sandsduchon.org/duchon/Musings/a/index.html#Clones_of_Linked_Lists>