CSU 670 Spring 2009 Sub Project 1
Karl Lieberherr and Ahmed Abdelmeged
January 6, 2009

Notes:
======
NOTE: This description is intentionally a bit underspecified.
Ask questions in class to clarify the requirements so that
you have a more complete picture.
In any software development project you have to dig for the requirements.
There is usually room for interpreting the requirements
in different ways.

Part 1
==

We are going to create an artificial world that we populate with little creatures that buy and sell services. Each creature gets the same initial life energy and they are responsible for the buy and sell decisions they make. The services are opportunities to win or lose energy. Each service that is bought consumes energy but there is the potential to get that energy back through clever use of a second resource: computational energy.

You have to take good care of those little creatures because you will be graded by what you "teach" them. A good dose of your computer science and mathematics knowledge will flow into those little creatures and if it does not, your grade will suffer. The little creatures will trade with each other using the knowledge you have given them. The life energy your creature will have left at the end of the game will influence your grade in a significant way.

Here are some trading examples: A summary of robot competitions. VodkaCurfewFest won the last competition. We will let VodkaCurfewFest and maybe other robots compete in our competitions. http://www.ccs.neu.edu/home/lieber/courses/csu670/sp09/competitions/f08/competitionsf08.htm.

The project of Spring 2009 CSU 670 is about designing and implementing an algorithmic derivative trading game, called the Specker Derivative Game (SDG). The objects traded are financial derivatives. (Financial derivatives are financial instruments whose value derives from the value of assets underlying them. Examples of financial instruments used in derivative transactions are options. Options are financial instruments that convey the right, but not the obligation, to engage in a future transaction on some underlying security.) Each derivative has a creator and can be bought by a player at the price offered by the creator. The creator of a derivative will not change when a derivative is sold. We say that a buyer owns a derivative. When you own a derivative with predicate T, you have the right to request raw material satisfying T and you have the right to sell back to the creator the finished product you produce from the raw material at a price that is equal to the quality of your product. The better quality you produce, the more you will be paid. On the other hand, the creator of the derivative has only the obligation to give you raw material satisfying predicate T, but he/she may make it hard for you to produce a high quality product.

The game has an administrator and a list of players. A player consists of a name, a list of offers (derivatives offered for sale), a list of sold derivatives (derivatives bought by other players), a list of bought derivatives (derivatives bought from other players), and a money field for the money owned by the player. Initially, each player gets 5 million dollars.

A derivative consists of a required name, a required predicate, a required creator (a player), a required price, and a few fields representing the state of the derivative as follows. If the optional boughtBy field is present, the derivative has been bought by another player. If the optional rawMaterial field is present, the buyer has requested raw material (currently unspecified). If the optional finished field is present, the buyer has delivered the finished product which is currently unspecified and it has a quality field which says how much the buyer has to be paid by the creator of the derivative.

Create an object-oriented design for The Specker Derivative Game. Include also the history, the list of moves that happened during a game (the administrator keeps track of this information). The moves that must be recorded include CreateDerivative, BuyDerivative, DeliverRawMaterial, DeliverFinishedProduct. CreateDerivative creates a derivative (unsold). BuyDerivative records who bought the derivative. DeliverRawMaterial records the raw material delivered by the creator of the derivative. DeliverFinishedProduct records the finished product delivered by the buyer and the quality (a real number in [0,1]) of the product. All prices are real numbers in [0,1] (fraction of a million $).

Here is an example of the kind of information you need to represent. This representation contains redundancy and you should attempt to reduce some or all of the redundancy in your design.

```
(player
   name Peter
   offers
     (derivative
        name d1
        predicate unspecified
        creator Peter
        price 0.75
      derivative
        name d2
        predicate unspecified
        creator Peter
        price 0.618
      )
   soldOffers
     (derivative
        name d3
        predicate unspecified
        creator Peter
        price 0.85
        boughtBy Paul
      derivative
        name d4
        predicate unspecified
        creator Peter
        price 0.618
        boughtBy Paul
      )
   boughtOffers
     (derivative
        name d5
        predicate unspecified
        creator Paul
        price 0.85
        boughtBy Peter
      derivative
        name d6
        predicate unspecified
        creator Paul
        price 0.6
        boughtBy Peter
      )
    money 0
)
history
  (create
     derivative
        name d1
        predicate unspecified
        creator Peter
       price 0.75
   create
     derivative
        name d4
        predicate unspecified
        creator Peter
        price 0.618
   buy
     derivative
        name d3
        predicate unspecified
        creator Peter
        price 0.85
        boughtBy Paul

   delivered
     derivative
        name d3
        predicate unspecified
        creator Peter
        price 0.85
        boughtBy Paul
        rawMaterial
          unspecified raw material
   finished
     derivative
        name d3
        predicate unspecified
        creator Peter
        price 0.85
        boughtBy Paul
        rawMaterial
          unspecified raw material
        finished
          unspecified finished product
          quality 0.25
```

)

The above example is intentionally incomplete and not from a real game. The purpose is only to show the kind of information that needs to be represented.

Turn in your object-oriented design, together with the corresponding Java or C# classes. The classes must be capable to parse a game history and print it back out. But you should not have to write any Java or C# code to define the details of the parsing and printing functionality. A tool should generate this information from a schema that you write for your object-oriented design.

I suggest that you use the DemeterF tool, a brand new tool developed in our college. See file: how-to-use-DemeterF in the course directory.

You may use any other Java or C# tool that has similar data-binding capabilities.

The details of the game will be introduced later. Basically, the players get a turn during which they create new derivatives or buy one from one of the players. It is explicitly allowed to create a derivative with predicate T and price p even if there is already a derivative of the same predicate T but with a higher price offered by some player.

If the derivatives are too expensive, nobody will buy them. For example, if all cost one (million). The rules of the game will make the players buy or lower the prices: Here is an important rule:

When it is a player's turn: The player must buy at least one derivative from another player or re-offer all derivatives that are for sale with a lower price. This will create new derivatives of an existing predicate, one for each predicate. In addition, a player may create new derivatives using new predicates. (The idea is that if a player does not want to buy, s/he must demonstrate that the prices are too high by lowering all of them. Several derivatives of the same predicate will be usually for sale and the clever buyer will normally choose the cheapest one.)

The game terminates after a fixed number of rounds (say 20). The winning player is the one who has the most energy (money) at the end of the game.

1. by careful thinking. Buy derivatives where you are guaranteed to make money.

2. by exploiting mistakes of others: (a) Sell derivatives where the buyer uses suboptimal techniques to create the finished product. This will lower the amount of money you have to pay for the finished product and will increase your profit. (b) Buy derivatives where you think the seller will make a mistake by giving you raw material from which you can produce a high quality product.

So in order to play the game well you must be good at: 1. spotting the best buys (which includes spotting the bad buys) 2. creating a high quality finished product out of the raw material.

In addition it helps, if you know the shortcomings of the other players but this is not needed to win in this game.

Of course, this all depends on the predicates used, and what it means to finish a raw product. This will be specified in project 2.

What to turn in for project 1/part 1: Turn in your object-oriented design, a DemeterF class dictionary or equivalent schema that can represent the required information. Turn in your parser and printer that has been generated from the schema.

The projects are done using pair programming. Find a partner and let me know if you cannot find one. Make sure you are a balanced team by comparing your answers to the questionnaire. We will be practicing pair programming: http://www.ccs.neu.edu/home/lieber/courses/csu670/sp09/project/ppp/pair-prog.html

You must turn in a development diary for each project: http://www.ccs.neu.edu/home/lieber/courses/csu670/sp09/project/ppp/development-diary.html

Part 2
==

Form a team of two students (the same pair as in part 1)
and play a few rounds of
SDG/classic/MAX-SAT (the raw materials are conjunctive normal forms) as described in

http://www.ccs.neu.edu/home/lieber/courses/csu670/sp09/requirements/two-player-param/two-player-sdg.pdf

This is an important document describing the requirements for the SDG.

Turn in the game history over 3 rounds.
In each round a player has one turn.

Follow the language definition for game histories in:

/home/lieber/.www/courses/csu670/sp09/software/abstract-histories

to turn in your history.

Turn in your answers electronically on Blackboard. Header to use:

Software Development CSU 670
Spring 2009
Project Number 1