Linnaeus University | Faculty of Technology | 1DV607

Objektorienterad analys och design med UML

Workshop 3 – Design Using Patterns

*Senast ändrad 2017-10-20 09:11 av **Tobias Ohlsson***

This describes the task for the third workshop, perform all the steps in order. Be sure to document all assumptions and changes you are making. Also, be sure to specify who participated in the work and be prepared to answer any questions about your model. You must be a course participant and logged in to see the information below.

Please use English in your models and documents. You may get international students as reviewers.

Introduction

In this workshop you will continue implementation and documentation of a Black Jack game. The design i similar to what we have done on the lectures but it is not exactly the same. Save models and implementation in your portfolio. The focus is not usability or a nice user interface but to have a robust and well documented design that can handle change. In the zip files you find a started but not playable version of the game in C# or Java. As there are many different variants of the rules one of the aims is to create a flexible design that supports different combinations of rules. There are also a class diagram that describes the packages, classes and the relations in the implementation. There is also a sequence diagram for one not implemented part of the game.

Read more about Black Jack on Wikipedia

Diagrams and Code

C# Black Jack Code GitHub repository (Fork or download)
Java Black Jack Code GitHub repository (Fork or download)

BlackJack_Stand_sequencediagram.png	67.62 kB	2011-10-20
BlackJack_class_diagram.png	65.14 kB	2011-10-20

Grade 2 Requirements

- Download, compile and run the game in your development environment (remember that it should run, but it's not playable).
- Study the class diagram and the source code to understand the design of the game.
- Implement the operation Game::Stand using the sequence diagram Game_Stand sequencediagram. The game should now be playable.
- Remove the bad, hidden, dependency between the controller and view (new game, hit, stand)
- Design and implement a new rule variant for when the dealer should take one more card. The new variant is "Soft 17″, use the same design pattern already present for Hit. Soft 17 means that the dealer has 17 but in a combination of Ace and 6 (for example Ace, two, two, two). This means that the Dealern can get another card valued at 10 but still have 17 as the value of the ace is reduced to 1. Using the soft 17 rule the dealer should take another card (compared to the original rule when the dealer only takes cards on a score of 16 or lower).
- Design and implement a variable rules for who wins the game. This variation could for example vary who wins on an equal score (in one implementation the Dealer wins, in the other the Player). The design should make it easy to add other variants without changing the Dealer.
- The code for getting a card from the deck, show the card and give it to a player is duplicated in a number of places. Make a refactoring to remove this duplication and that supports low coupling/high cohesion. The code that is duplicated i similar to this:

  ```
  Card c = deck.GetCard();
  c.Show(true/false)
  player.DealCard(c);
  ```

- Use the Observer-pattern to send an event to the user interface that a player (human or dealer) has got a new card in his hand. When the event is handled the user interface should be redrawn to show the new hand (with the new card) and the game should be briefly paused, to make the game a bit more exciting, the pausing code should be in the user interface (view or controller) and not in the model. The pause should be when any player (dealer or player) gets a card.
  The start game procedure should be something like:

  Dealer:
  Player: c1
  pause

Dealer: c1
Player: c1
pause
Dealer: c1
Player: c1, c2
pause
Dealer: c1, c2
Player: c1, c2

- Update the class diagram to reflect the changes you make (it is not necessary to recreate the whole diagram only the parts that have been affected by your changes).

You should have the following in your portflio and all parts should match

- Easily executable version alt. instructions on how to execute.
- Source code for the entire application
- An updated class diagram

Requirements for Grade 3

- Perform the requirements for grade 3
- Expand the design and implementation with:
- Use the design pattern Abstract Factory to offer some “finished” rule factories for different rule combinations. Parameterize the Game class in the constructor with the factory to use.
- Use the design pattern Visitor to enable the view to print what rules are used at the start of the game. The model classes shoul **not** have any form of “type”-attrubute (string, enum, int etc).
- Hand in the assignment before the deadline.

Requirements for Grade 4

- Perform the requirements for grade 3
- Perform the requirements for grade 4
- Expand the design and implementation with:
    - Design and implement a new (not console based) user interface (view/controller) to the game, for example using Swing, WPF, OpenGL, XNA or a web based variant (you are free to convert it to another language than C#/Java if you want). Use as much of the present design as possible but change it if it is needed. The following video shows an XNA version of the game:

    - Write a short report on the conversion, what went well, what needed to be changed, why and how was the change made.
- Hand in the assignment before the deadline.