Abbas Rizvi
100746798
March 3, 2022

Card Master: Blackjack

https://github.com/Abbas-Rizvi/card_master

Overview

The game that I have chosen to create is a simple card game application with the first game installment being a text based blackjack card game. I decided to create blackjack for the basis of this project as I am very familiar with the game and logic associated as well as its use of several different classes. Through creating and testing a game with a simple yet complex structure such as Blackjack, I hope to develop good testing and quality skills which can be extrapolated and used in larger projects in the future.

The Blackjack game will be housed in a larger application which makes use of the same base classes (deck, player, card, menu), to allow for the creation of additional card games reusing the same assets. This will also put greater emphasis on the design process and ensuring that quality is maintained throughout is imperative to the functionality of the software throughout its lifecycle.

Design Methodology

The design methodology employed for the creation for the Blackjack card game was the Agile methodology. An agile approach to the design process fit very well with the project and worked well in the creation of the several classes required in the game. With the intention of creating additional games based on the same deck and player classes, it is imperative that integration testing and maintenance is conducted throughout the lifecycle. The project can simply be thought of as the product of multiple sprints, of which the Blackjack game is just one of.

Development

1. Requirements

The initial requirements for the project are as follows:

- The timespan for which the project is to be developed is two weeks. This encompasses all aspects including development, testing and any modifications which are needed to be made
- The product should be good quality software and ready for deployment

The initial requirements for the blackjack game are as follows:

- The player player should be able to play a game of blackjack against a computer opponent
- The player should be able to wager (virtual) currency on the games
- There should be three outcomes to the game for the player
    - Player Win
        - When player hand is greater than dealer
        - When dealer hand is over 21
    - Player Lose
        - When dealer hand is greater than player
        - When player hand is over 21
    - Draw
        - When player hand is equal to dealer hand

Additional features and functionality can be added during the project design and implementation.

2. Design

Programming Language

The programming language to be used for the creation of the card master application is Java. This was selected due to an already established familiarity with the language, its portability due to running on the Java virtual machine, and its mature JUnit testing framework.

Design Approach

The game will be created following a composite design pattern. The functions of the game can be split up into different components which can then come together to form the completed project. The Card Master software will be divided into the following manner

- Menu
    - Blackjack
        - Player
            - Deck
                - Card

It is important to note that although following this design pattern does not have many benefits for the design of the blackjack game, as Card Master plans to implement additional games in the future employing a coherent plan for the design will allow the appropriate levels of abstraction and coupling to allow for the software to be maintained with the addition of additional classes using these classes.

Prior to implementation, a draft was created outlining the main functions in each of the classes to provide a guide for the development of each. It would be possible to create a comprehensive design but providing a looser design would allow for more flexibility in features and implementations. This could potentially allow for more efficient usage of other objects within the classes though is somewhat risky as the project could quickly become disorganized if consistency is not maintained throughout.

Testing Methodology

Another important factor which needs to be considered in the design phase is the testing. Each individual class within the Card master application will be tested individually using JUnit Unit tests. Given the composite nature of the application so long as sufficient testing code coverage is asserted for each class in the hierarchy, there should not be issues or bugs in higher order classes.

3. Implementation

Development

Implementation began first on the *Card* and *Deck* classes. This was decided to be conducted together because of the way the program is to be structured, with the Deck being composed of several Card Objects. This same pattern was repeated for the remaining classes, working incrementally on each higher level within the program and recursively changing the lower base classes, though major changes affecting more than one class were not required.

4. Testing

*All test cases are located in test package located at `src/test`

The testing was conducted following the methodology outlined in the design phase. Testing was conducted on four of the five classes in the game software. The *Menu* class was omitted from testing as it contained very little logic and relied heavily on Scanner input, something which is relatively challenging to test effectively.

The main focus of the tests is on the BlackJack class as it has the most intricate logic and was causing issues during development. Prior to testing there were several bugs which could be observed such as the dealer making incorrect decisions in regard to drawing cards, the inability for players to reset their hands and return cards to deck and a faulty wagering system. Despite these bugs, not all functions within the class need to be tested as some functions such as the *table, wager,* and *play* serve as setup for the game and are concerned mostly with handling Scanner input rather than providing any real functionality and are thus difficult to test effectively.

Code Refactorization

An immediate problem which was observed upon the completion of the development and prior to testing was the method in which testing would be conducted. With the *BlackJack* class, the program was very monolithic, having all the logic in primarily one function. While having a large block of software is functional, it is very difficult to both test for bugs and extend/modify, prompting a large refactoring of the classes. This was caused by negligence in the design phase and will be further discussed in the review portion.

This change resulted in the creation of the following new functions prior to beginning tests:

- ***returnCards*** function to return player cards
    - Called upon game completion
- ***wager*** function to be used to receive a player wager input and return whether or not it is valid with the player object.
    - Used as an exit flag prior to the creation of the game.
- ***dealCard*** function which allows the specification of a player and number of cards and deals the appropriate number of cards to the player.
    - Used in the *initialDeal* function
    - Used in the *play* function when player or dealer selects “hit” option
- ***playerOption*** function to manage logic for handling option selection input
    - Used when providing player with “hit” and “stay” options
- ***hit*** function to deal card to player/host and verify game exit condition not reached
    - Used when player selects “hit” option

    - Used when program decides dealer should draw card

- ***dealerOption*** to check if the dealer hand meets all conditions to hit. It should be noted that this function was created as a separate entity to allow for easier testing as this is a core game mechanic.

    - Used as flag to decide if dealer should hit

Major Testing Issues

- ***dealerOption***
  During testing an issue was identified as a result of a faulty logical operation. This function was altered to accept two *Player* objects to allow for better testing and consistency. Through running several tests on the function, the issue was identified and resolved.

- ***initialDeal***
  While testing it was found that the cards were not being dealt to the players as their deck size would remain at 0. It is unknown what caused this issue but it was discovered that the *dealCard* function could be expanded to allow for the dealings of more than one card and thus become a more robust method. *InitialDeal* would then become obsolete with no detrimental effect to the program.

- ***ReturnCards***
  A similar issue as to what was found in *dealerOption* was again found in *returnCards*. The cards were not being returned correctly from the two *Player* objects. This issue was resolved through again allowing the function to accept two *Player* objects along with a *Deck* to which the cards are to be returned.

  This bug was particularly difficult to resolve as there was some unexpected behavior in the *Deck.*numCards() function. Here testing was found very useful as it facilitated debugging and allowed simply running the test case to verify if the issue was resolved.

5. Review

Overall the goal of creating a good quality game software was completed. The Blackjack game functions as intended and without errors. Additionally it is very extensible and easy to maintain should any future revisions or improvements be needed. Throughout the process much was learned about software quality and software testing which would alter the way I would go about the project given a chance to restart.

Overall Design

Something which I should have better considered is managing the overall design of the software. I had come up with a loose guideline for how I would go about the design but I think a proper UML style diagram would have helped with organization immensely.

Had I had a better understanding of the program structure, I would have had an easier time separating the logic into different methods and functions. This was an extremely time consuming process and caused a lot of confusion as to what was to be expected of each one. Having a hard outline of the responsibilities of each method, I think would make structuring it in this manner a trivial task.

Testing Observations

There were no issues in testing the *Card, Deck* and *Player* classes however there were several issues encountered with the BlackJack class. Beyond the initial refactorization discussed above, each of the functions used objects from the class scope (deck, player, host). This generally works in development and as such I have continued to use it in the past however through the testing process I realized that it can cause many issues and bugs in the program. The main issue I found is that it can cause issues when working with sensitive data and variables with private data types as they cannot be specified through the function; it is essentially useless outside its own class. Building upon this, testing is essentially running the functions through a secondary class, meaning that none of these functions will be able to be tested properly. To combat this issue I had to revise all the functions and make them accept parameters for these objects and review each time they were called to ensure that the proper objects are passed through to the function.

Another consideration I should have made to allow for better testing was the creation of a Scanner method to allow for easier and more comprehensive testing of all functions. As it is difficult to simulate Scanner objects in test environments it became difficult to work with some of the menu classes which while less important are still a crucial part of the program. Structuring the program in such a way where everything comes together in a modular fashion would essentially allow the injection of data in place of these Scanner calls making testing easier.

Appendix

Sample Game Screenshots

Main menu and Blackjack Menus

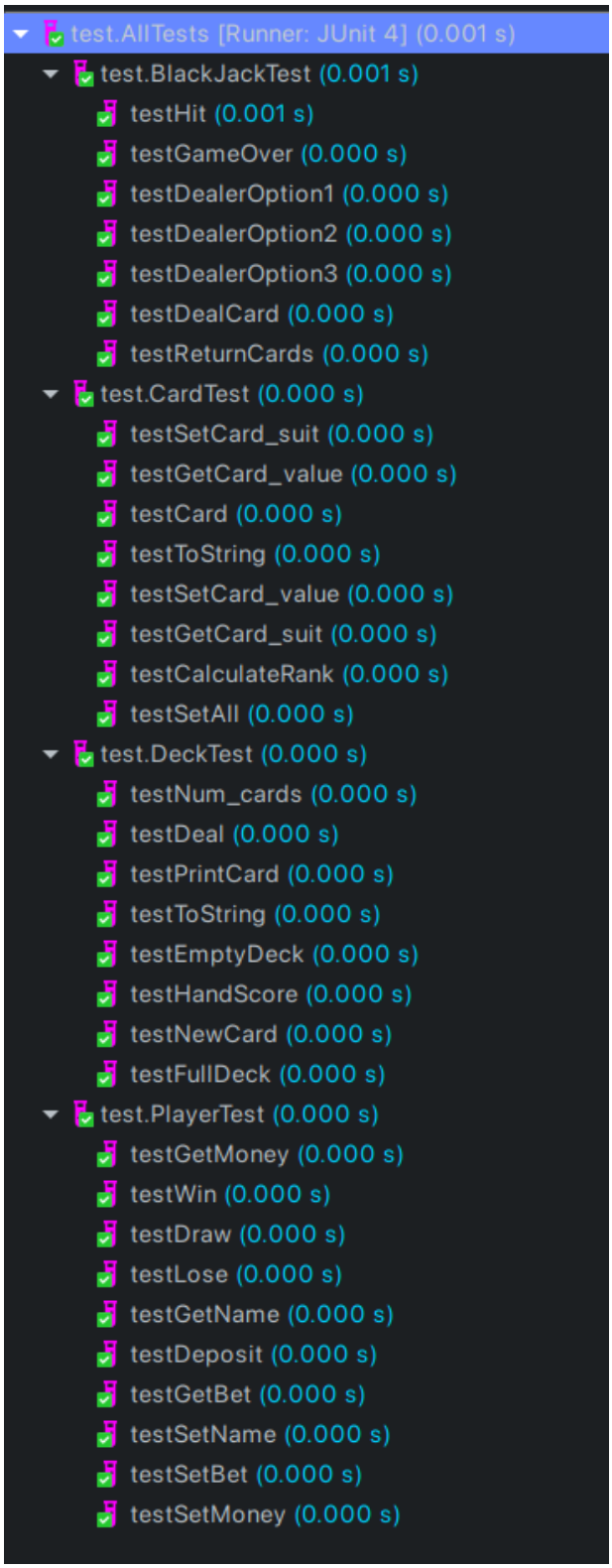
```
Welcome to Card Master


What game would you like to play today?
1: Blackjack
9: Quit
1
Welcome to BlackJack!
Enter Name: Abbas
Enter Abbas Balance: 1000


~~~~   BLACKJACK   ~~~~
---- Player Info ----
Abbas : $1000.0


Select an option
1: New Game
2: Deposit Additional Funds
9: Withdraw Funds and Exit
```

Wager and Gameplay

```
Welcome to the Blackjack table Abbas
Your Current Balance is $1000.0
Enter Amount You Wish to Wager:
100
Abbas Has Wagered $100.0!

---- Dealer Hand ----
HIDDEN
JACK OF CLUBS

---- Player Hand ----
SIX OF HEARTS
EIGHT OF HEARTS

Player Turn:
1: Hit
2: Stay
```

Test Screenshots (JUnit 4)

Test Suite



Sample Test Cases

```
@Test
public void testReturnCards() {

    // remove 10 cards from deck
    bj.dealCard(host, deck, 5);
    bj.dealCard(p1, deck, 5);

    // verify removal
    assertEquals(42, deck.num_cards());

    // return cards
    bj.returnCards(p1, host, deck);

    //verify retrieval
    assertEquals(0,p1.deck.num_cards());
    assertEquals(0,host.deck.num_cards());

    assertEquals(52, deck.num_cards());
}

// test dealer option if hand score is less than player and less than 17
@Test
public void testDealerOption1() {
    Player host = new Player();
    Player p1 = new Player();

    Card card1 = new Card(Card.Suit.DIAMONDS,Card.Value.QUEEN);

    // host hand is 10
    host.deck.newCard(card1);

    // player hand is 20
    p1.deck.newCard(card1);
    p1.deck.newCard(card1);

    // Should return true
    assertTrue(bj.dealerOption(p1, host));
}
```

Run Instructions

Extract from Readme.md at https://github.com/Abbas-Rizvi/card_master

Clone repository and enter directory

```
$ git clone https://github.com/Abbas-Rizvi/card_master.git
$ cd card_master
```

Run Menu.java file using JVM

src/game/Menu.java