



**Assignment 1: Choosing the right software process to build good-quality software, and ensuring quality using test automation**

Project: Black Jack (Java)

Yash Patel 100746810

SOFE 3980U: Software Quality

Dr. Akramul Azim

February 17, 2022

## **Software Quality Assignment 1: Black Jack Game Report**

### **Program Description**

The program I have chosen is for this assignment is to recreate the card game, Black Jack. The premise of the game is to get 21 or the closest to it with the values of the cards that are given to you. The way the game works is that the dealer will shuffle the deck and hand cards one at a time to each player starting from the dealer. Each player will get 2 cards, and each player has to hit to be dealt another card. However, if their total score goes over 21 they will bust and lose the game.

### **Program History**

The base of the game will be taken from an assignment made in Object-Oriented Programming SOFE2710U. The assignment required us to create a poker game with many different functions. For this assignment, I will be stripping away many unnecessary functions and converting poker to blackjack. Essentially, due to the previous implementation of the methods and classes, I will be required to work from scratch but will use some of the logic that previously existed. Therefore, the majority of the application will be refactored to support new implementation.

### **Software Process**

The software process chosen for this project will be the Agile Process Model. The reason I have chosen this process model is that is perfect considering the limited time to work on the project as well as allowing later versions of the software to be easily developed. The main focus of this project will be functionality since I would like to create a working game and the chosen model provides just that.

#### **1. Planning**

- Create the card game BlackJack.
- Use Java as the program's language
- User will launch the game and play one round and it will finish and will require relaunching of the program to play again.

#### **2. Design**

- The game will be created using object-oriented programming
- The program will have a card class that will consist of getters and setters (getSuit, getValue, setCard, etc)
- The program will also have a main game class that will create a deck, shuffle the deck, assign cards, create a score counter, check for a blackjack, and return score.
- The program will have a driver class that instructs and initializes the game as well as the cards.

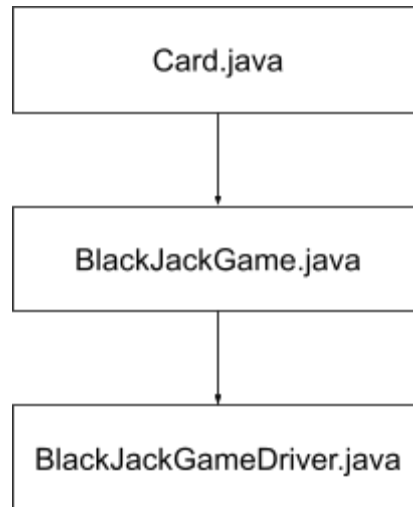


Figure 1: Game design model

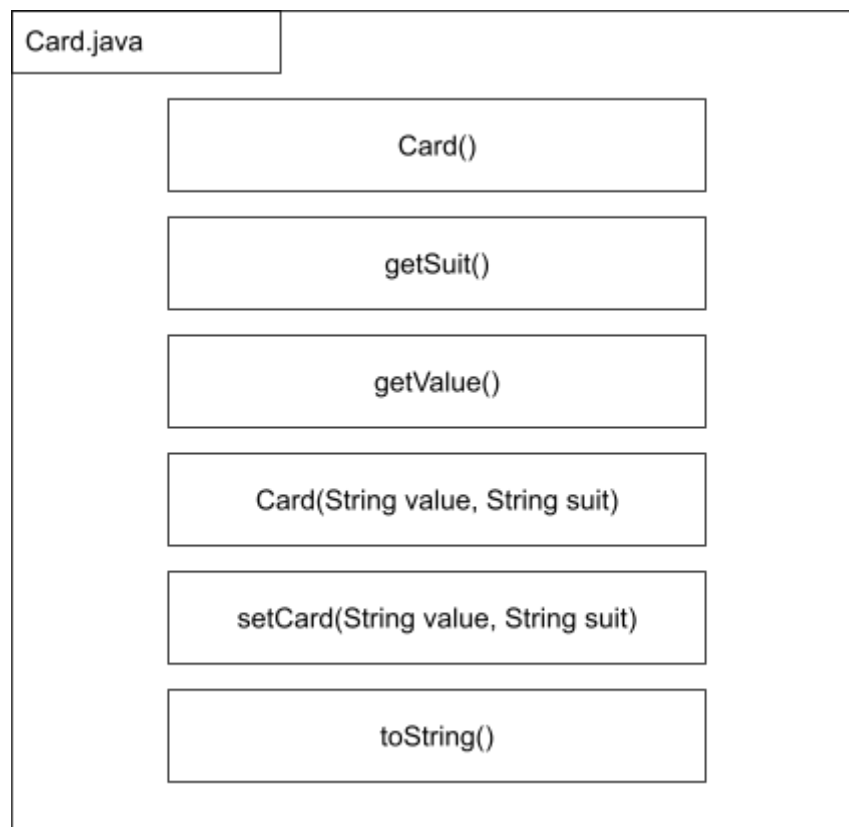


Figure 2: Card.java with methods



Figure 3: Card.java with methods

### 3. Development

Within the development phase of the game, it went through several attempts of trial and error, as well as many iterations of development to ensure functionality. The process started with developing the Card class and ensuring that the proper methods were created for ease of use. Developing BlackJackGame class was quite difficult as it required proper planning and execution. During this process, it was required to change methods within the Card class as they wouldn't work as intended. The final process was to create BlackJackGameDriver class which would encapsulate the previous methods to create a running environment. This process had the most iterations of development as Card and BlackJackGame classes had to be reconstructed in many events to allow for proper execution.

### 4. Quality Assurance

The goal created for the testing phase for the game was to have over 80% coverage for the methods created in Card and BlackJackGame classes. The library chosen for this game was Junit as it provides the essential functions as well as the coverage percent

which is key for this phase. The testing will consist of unit tests to see if the actual values returned matched the theoretical values.

The tests created are special for certain aspects of the game and are to ensure correct functionality. The first test created is to determine if player hand and dealer hands are different values by using *assertNotEquals()*. The second test was created to determine if two player hands are the same by using *assertEquals()*. The third test was created to determine if the scores were actually functional, this was done by assigning cards and checking if they were equal or not equal, and using *assertEquals()* or *assertNotEquals()* respectively. The fourth test was created to determine if cards were properly set by manually assigning a suit an value and using *assertEquals()*. The fifth test was created to determine if card values were being assigned correctly by manually assigning a card with a value and using the getter method within *assertEquals()*. The final and sixth test was created to determine if card suits were being assigned correctly by manually assigning a card with a suit and using the getter method within *assertEquals()*.

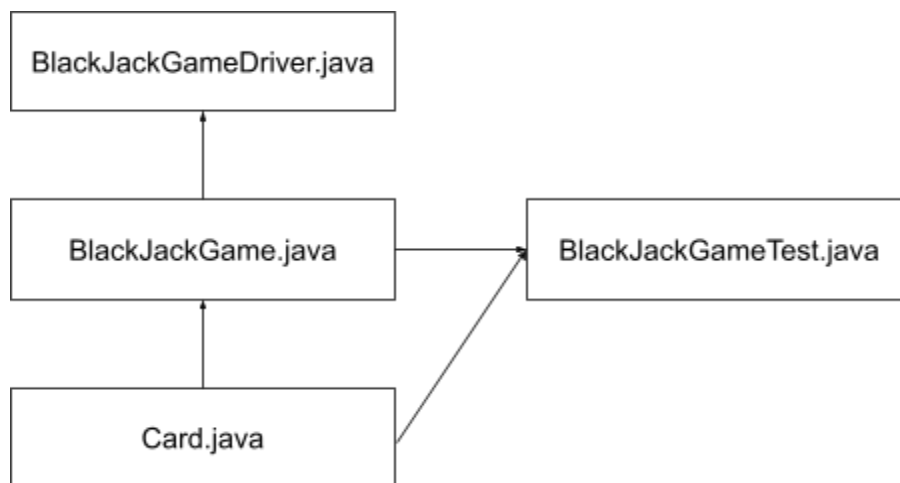


Figure 4: Game design model with the test class

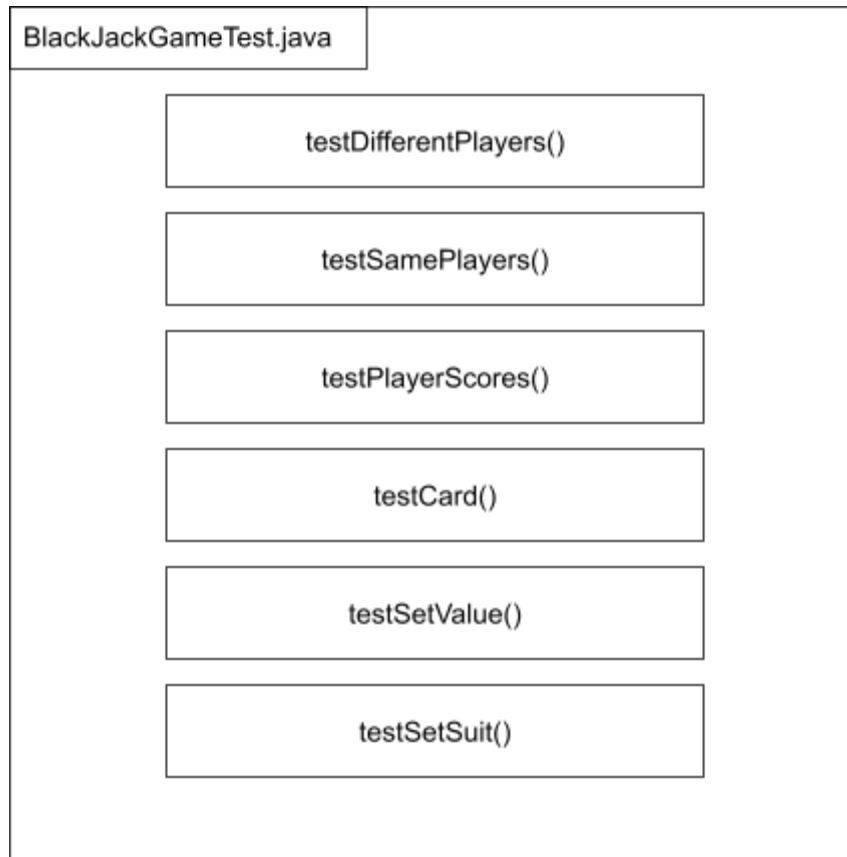


Figure 5: BlackJackGameTest.java with methods

## 5. Deployment

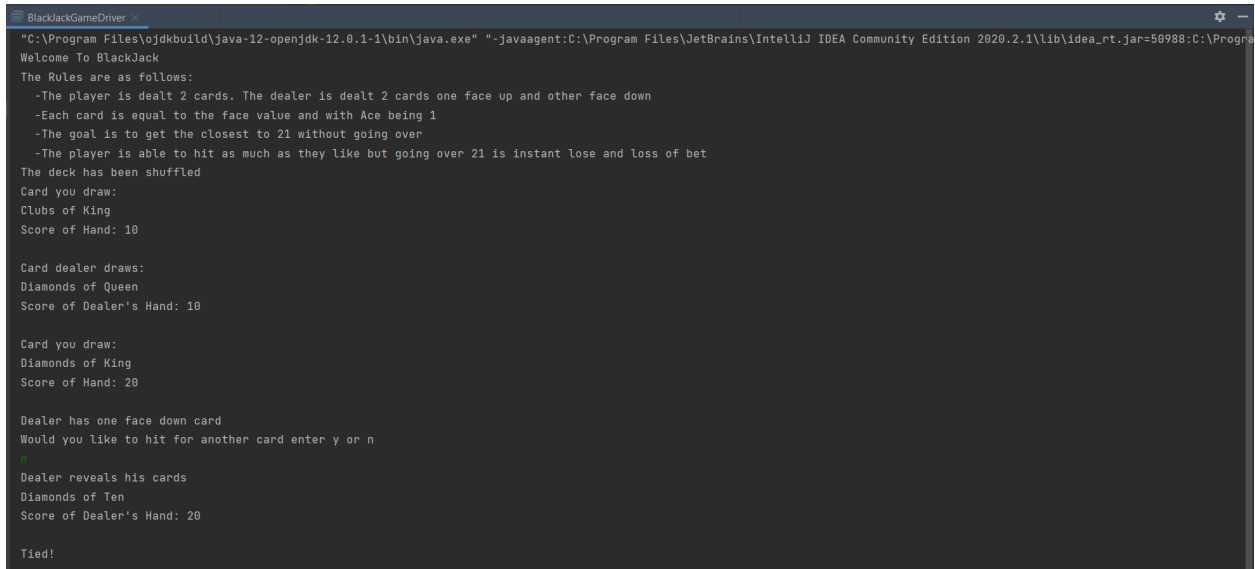
The final phase of the Agile Process Model is the deployment phase, within this stage, a functional version of the game has been created. Due to this game being a Java application there can not be a live instance due to the language, however, there is a GitHub repository with all the required files that can be downloaded and used. By providing an open-source repository and due to the application being created with the agile process the game can be further developed, and supports frequent change.

GitHub repo: [yash-patel268/Black-Jack-Game \(github.com\)](https://github.com/yash-patel268/Black-Jack-Game)

### Challenges Encountered

Some of the challenges encountered during the development of this game include creating proper setters and getters, creating different user types (player, dealer), matching hit times of player to the dealer, and making sure the win conditions are proper and functional. The challenges encountered during the testing of this game include adapting to the new version of Junit, not receiving the expected return value, and creating quality test cases.

## Screenshots



```
"C:\Program Files\jdkbuild\java-12-openjdk-12.0.1-1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.2.1\lib\idea_rt.jar=50988:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.2.1\bin" -Didea.config.path=C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.2.1\conf -Didea.copyright=C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.2.1\copyright -Didea.home.path=C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.2.1\bin -Didea.platform.prefix=Java12 -Didea.vendor.id=idea -Didea.version=2020.2.1 -jar C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.2.1\bin\idea_rt.jar=50988:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.2.1\bin

Welcome To BlackJack

The Rules are as follows:
- The player is dealt 2 cards. The dealer is dealt 2 cards one face up and other face down
- Each card is equal to the face value and with Ace being 1
- The goal is to get the closest to 21 without going over
- The player is able to hit as much as they like but going over 21 is instant lose and loss of bet

The deck has been shuffled

Card you draw:
Clubs of King
Score of Hand: 10

Card dealer draws:
Diamonds of Queen
Score of Dealer's Hand: 10

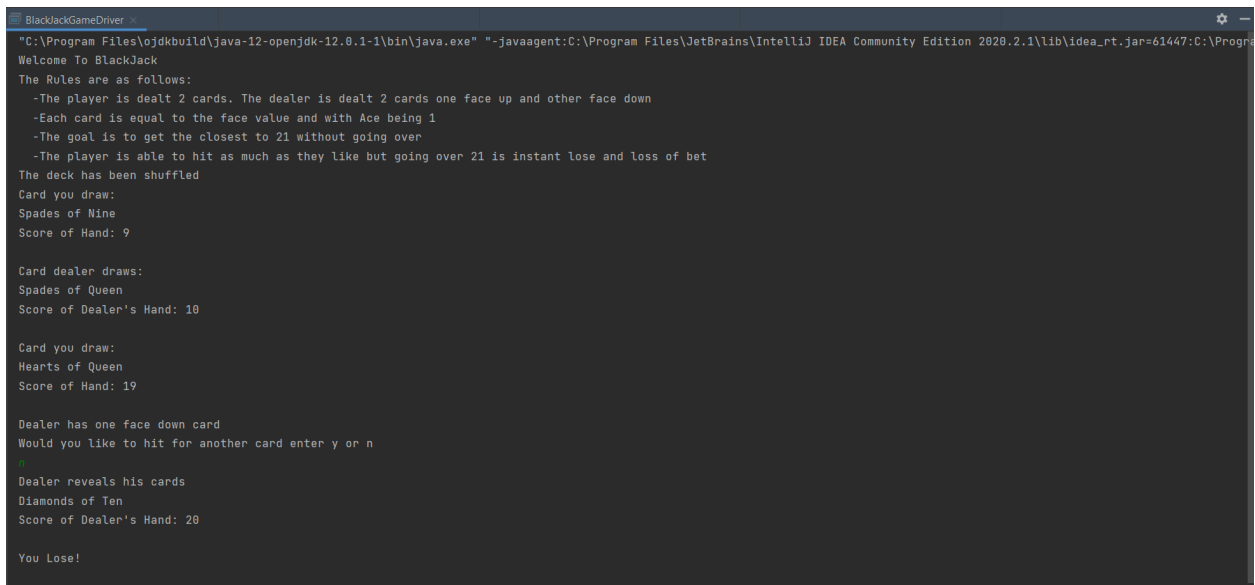
Card you draw:
Diamonds of King
Score of Hand: 20

Dealer has one face down card
Would you like to hit for another card enter y or n
n

Dealer reveals his cards
Diamonds of Ten
Score of Dealer's Hand: 20

Tied!
```

Figure 6-1: Game run time with a tied outcome



```
"C:\Program Files\jdkbuild\java-12-openjdk-12.0.1-1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.2.1\lib\idea_rt.jar=61447:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.2.1\bin" -Didea.config.path=C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.2.1\conf -Didea.copyright=C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.2.1\copyright -Didea.home.path=C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.2.1\bin -Didea.platform.prefix=Java12 -Didea.vendor.id=idea -Didea.version=2020.2.1 -jar C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.2.1\bin\idea_rt.jar=61447:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.2.1\bin

Welcome To BlackJack

The Rules are as follows:
- The player is dealt 2 cards. The dealer is dealt 2 cards one face up and other face down
- Each card is equal to the face value and with Ace being 1
- The goal is to get the closest to 21 without going over
- The player is able to hit as much as they like but going over 21 is instant lose and loss of bet

The deck has been shuffled

Card you draw:
Spades of Nine
Score of Hand: 9

Card dealer draws:
Spades of Queen
Score of Dealer's Hand: 10

Card you draw:
Hearts of Queen
Score of Hand: 19

Dealer has one face down card
Would you like to hit for another card enter y or n
n

Dealer reveals his cards
Diamonds of Ten
Score of Dealer's Hand: 20

You Lose!
```

Figure 6-2: Game run time with losing outcome

```

BlackJackGameDriver
"C:\Program Files\jdkbuild\java-12-openjdk-12.0.1-1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.2.1\lib\idea_rt.jar=61474:C:\Progr
Welcome To BlackJack
The Rules are as follows:
- The player is dealt 2 cards. The dealer is dealt 2 cards one face up and other face down
- Each card is equal to the face value and with Ace being 1
- The goal is to get the closest to 21 without going over
- The player is able to hit as much as they like but going over 21 is instant lose and loss of bet
The deck has been shuffled
Card you draw:
Diamonds of Nine
Score of Hand: 9

Card dealer draws:
Diamonds of Queen
Score of Dealer's Hand: 10

Card you draw:
Spades of Queen
Score of Hand: 19

Dealer has one face down card
Would you like to hit for another card enter y or n
y
Dealer reveals his cards
Spades of Eight
Score of Dealer's Hand: 18

You Win!

Process finished with exit code 0

```

Figure 6-3: Game run time with winning outcome

Test Case	Execution Time
BlackJackGameTest (com.company)	20 ms
testDifferentPlayers	18 ms
testSamePlayers	0 ms
testCard	1 ms
testPlayerScores	1 ms
testSetSuit	0 ms
testSetValue	0 ms

Figure 7: All test cases passing

Package/Class	Methods Covered	Lines Covered
com.company	75%	68%
BlackJackGame	100%	87%
BlackJackGameDriver	0%	0%
BlackJackGameTest	100%	97%
Card	100%	100%

Figure 8: Test coverage percentages



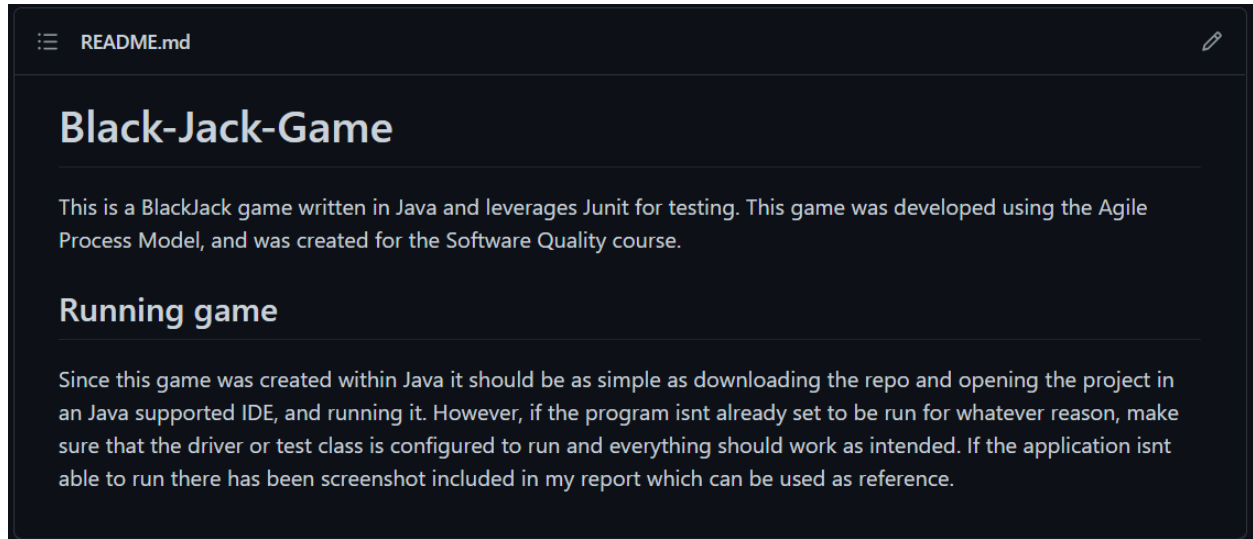


Figure 9: GitHub repo README.md