Artiom Arutiunov

ID: 504597668

Professor Stahl

3/12/16

Project 7 Report

A) What I found most difficult about this project was working between the various classes that were provided for us to help write the Player and Game classes. This was the first time I got to work with more than two classes, which entailed keeping track of a lot of different methods and member variables of various classes. In addition, I encountered a roadblock when I realized that some members of the classes provided were private, so I had to use getter and mutator functions to reach those members. Another aspect of this project I found challenging was translating various elements of the actual game of blackjack to code form. For example, it took a while for me to come up with the correct algorithm for calculating when an ace should be worth eleven and when it should be worth only 1. The same case applied to the dealerStands method of class Game, as I had to come up with the correct combination of Player methods and methods within the Game class itself.

B)

I first tested my code by running it against the code Howard provided in the spec.

First, Howard declared four objects of the Card class.

// Howard's test code

Player p;

Card c( Face::ACE, Suit::DIAMONDS );

Card d( Face::TEN, Suit::CLUBS );

Card e( Face::THREE, Suit::SPADES );

Card f( Face::EIGHT, Suit::HEARTS );

He then ran some basic tests to ensure that methods of my Player class functions correctly:

assert( p.handcount() == 0 );

assert( p.hasBlackJack() == false );

assert( p.cardCount() == 0 );

p.acceptCard( c );

assert( p.handcount() == 11 );

assert( p.hasBlackJack() == false );

assert( p.cardCount() == 1 );

assert( p.getCard(0) == c);

p.acceptCard( d );

assert( p.handcount() == 21 );

assert( p.hasBlackJack() == true );

The compiler agreed with all of these assertions, meaning that the basic functions of my Player class are working correctly.

Next, Howard’s test cases tested basic methods of the Game class using the same set of cards.

p = Player();

Player dealer;

Game g( p, dealer );

g.playerStands();

g.dealerStands();

assert( g.dealerBusted()==false );

assert( g.playerBusted()==false );

assert( g.playerTied() == true );

assert( g.playerWon() == false );

assert( g.playerLost() == false );

assert( g.playerHasBlackJack() == false );

assert( g.dealerHasBlackJack() == false );

The compiler agreed with these assertions as well, indicating that the methods of my Game class were working appropriately.

His code then ran a combination of the two previous blocks of code, testing methods from both classes.

p = Player( );

p.acceptCard( c );

p.acceptCard( d );

dealer = Player( );

dealer.acceptCard( e );

dealer.acceptCard( f );

g = Game( p, dealer );

g.playerStands();

g.dealerStands();

assert( g.dealerBusted()==false );

assert( g.playerBusted()==false );

assert( g.playerTied() == false );

assert( g.playerWon() == true );

assert( g.playerLost() == false );

assert( g.playerHasBlackJack() == true );

assert( g.dealerHasBlackJack() == false );

The compiler agreed with all of these assertions as well, indicating that the overall project is working on an elementary level.

I then came up with some test cases of my own, which dealt with more intricate components of the project.

I first declared 10 objects of the Card class to have a broad range of Card objects to work with while testing my code.

Player Artiom;

Artiom = Player();

Card j (Face::KING, Suit::DIAMONDS );

Card j2 (Face::DEUCE, Suit::HEARTS);

Card j3 (Face::EIGHT, Suit::CLUBS);

Card j4 (Face::ACE, Suit::SPADES);

Card j5 (Face::ACE, Suit::HEARTS);

Card j6 (Face::ACE, Suit::CLUBS);

Card j7 (Face::ACE, Suit::DIAMONDS);

Card j8 (Face::FIVE, Suit::DIAMONDS);

Card j9 (Face::THREE, Suit::HEARTS);

Card j10 (Face::FOUR, Suit::SPADES);

I first declared 10 objects of the Card class to have a broad range of Card objects to work with while testing my code.

Artiom.acceptCard(j4);

Artiom.acceptCard(j5);

Artiom.acceptCard(j6);

Artiom.acceptCard(j7);

assert( Artiom.handcount() == 14);

Next, tested the exceptions I had to write for several methods of the Player class using a try-throw-catch threesome.

Player Artiom2;

try

{

for (int i = 0; i <= 12; i++)

{

Artiom2.acceptCard(j);

}

}

catch (std::logic\_error room )

{

cerr << "Not enough space for another card!" << endl;

}

try

{

Artiom2.getCard(12);

}

catch (std::logic\_error badindex)

{

cerr << "This index is invalid." << endl;

}

The program output both cerr statements, indicating that the code does in fact correctly recognize when there is a logic error.

Following this, I tested for various situations that could occur during the game.

First, I tested for defaulting to a player loosing if both the player and the dealer bust.

Player artiom3;

Player dealer2;

artiom3.acceptCard(j);

artiom3.acceptCard(j3);

artiom3.acceptCard(j8);

dealer2.acceptCard(j);

dealer2.acceptCard(j3);

dealer2.acceptCard(j8);

assert (artiom3.handcount() == 23);

assert(dealer2.handcount() == 23);

Game g2 (artiom3, dealer2);

assert( g2.playerBusted() == true);

assert( g2.dealerBusted() == true);

g2.playerStands();

g2.dealerStands();

assert( g2.playerLost() == true);

Initially, the compiler did not agree with the assertion that the player lost, which meant that I had to go back to the Game class and fix my code to account for this situation.

Next, I tested for a normal player wins scenario.

Player artiom4;

Player dealer3;

artiom4.acceptCard(j9);

artiom4.acceptCard(j);

artiom4.acceptCard(j3);

dealer3.acceptCard(j);

dealer3.acceptCard(j3);

Game g3 (artiom4, dealer3);

g3.playerStands();

g3.dealerStands();

assert(g3.playerWon() == true);

The compiler agreed with these assertions as well.

I then tested my code for situations dealing with blackjack.

First, that the dealer wins with blackjack.

Player artiom5;

Player dealer4;

artiom5.acceptCard(j);

artiom5.acceptCard(j9);

artiom4.acceptCard(j10);

dealer4.acceptCard(j);

dealer4.acceptCard(j7);

Game g4 (artiom5, dealer4);

g4.playerStands();

g4.dealerStands();

assert(g4.playerHasBlackJack() == false);

assert(g4.dealerHasBlackJack() == true);

assert(g4.playerLost() == true);

The compiler agreed with these assertions, meaning that my hasBlackJack function works correctly.

I then tested the same situation except with the player having blackjack.

Player artiom6;

Player dealer5;

artiom6.acceptCard(j);

artiom6.acceptCard(j4);

dealer5.acceptCard(j);

dealer5.acceptCard(j9);

dealer5.acceptCard(j10);

Game g5 (artiom6, dealer5);

g5.playerStands();

g5.dealerStands();

assert(g5.playerHasBlackJack() == true);

assert(g5.dealerHasBlackJack() == false);

assert(g5.playerWon() == true);

The compiler also agreed with these assertions, so both blackjack functions work correctly.

Then, I tested for both players getting blackjack, and the resulting tie.

Player artiom7;

Player dealer6;

artiom7.acceptCard(j);

artiom7.acceptCard(j4);

dealer6.acceptCard(j);

dealer6.acceptCard(j5);

Game g6 (artiom7, dealer6);

g6.playerStands();

g6.dealerStands();

assert(g6.playerHasBlackJack() == true);

assert(g6.dealerHasBlackJack() == true);

assert(g6.playerTied() == true);

The compiler agreed with these assertions, meaning that the code works correctly for identifying the tie.

Lastly, I tested the basic getCard method of the Player class.

Card i2 = p.getCard(0);

cerr << i2 << endl;

i2 = p.getCard(1);

cerr << i2 << endl;

i2 = dealer.getCard(0);

cerr << i2 << endl;

i2 = dealer.getCard(1);

cerr << i2 << endl;

The correct cards were outputted by the compiler for each of these as well.