Recitation #3, Worksheet - C++ Blackjack Solutions

In groups of 3-4, compare and contrast source code #1 and source code #2 with our assignment #2. What do you understand and not understand about these solutions?

**Source #1:** Taken from [http://www.it2051229.com/cppbjack.html](http://www.it2051229.com/cppbjack.html)

```cpp
#include "BlackJackGame.h"

// Initialize the properties of the black jack game
BlackJackGame::BlackJackGame() {
    // At the moment there is nothing to initialize
}

// Entry point of the program
void BlackJackGame::run() {
    displayInstructions();

    while(true) {
        player.getHand().reset();
        dealer.getHand().reset();

        if(player.getFundsAvailable() == 0) {
            cout << "You have no funds left." << endl;
            break;
        }

        // make a bet
        cout << endl;
        makePlayerBet();

        // terminate the game if the bet amount placed is
        if(player.getBetAmount() == 0) {
            break;
        }

        // make a hand
        cout << endl;
        makeDealerHand();

        // check if player has lost
        cout << endl;
        checkIfPlayerLost();

        // you can also use cout to print statistics
        cout << endl;
        cout << "Statistics:
        cout << "Hand Size: " << handSize << endl;
        cout << "Total: " << total << endl;
        cout << "Number of Aces: " << numberOfAces << endl;

        if(handSize > MAXHANDSIZE) {
            cout << "Max. hand size exceeded!" << endl;
        }

        cout << endl;
        cout << "Player wins: " << playerWins << " | Dealer wins: " << dealerWins << " | Draws: " << draws << endl;

        // replay and start again
        cout << endl;
        cout << "Do you want to replay? (y/n) ";
        cin >> replayInput;
        if(replayInput == 'n') {
            break;
        }
    }
}
```
#include "Deck.h"

// Upon creating the deck, fill it with 52 shuffled cards
Deck::Deck() {
    refill();
    shuffle();
}

// Shuffles the deck to randomly place them
void Deck::shuffle() {
    for(int i = 0; i < MAXSIZE; i++) {
        int j = i + (rand() % (MAXSIZE - i));

        Card tempCard = deck[i];
        deck[i] = deck[j];
        deck[j] = tempCard;
    }
}

// Returns the next card from the deck
Card Deck::nextCard() {
    // If there are no more cards left, then refill the deck
    if(empty()) {
        refill();
        shuffle();
    }

    Card card = deck[next];
}

#include "Hand.h"

// Initialize the hand properties
Hand::Hand() {
    reset();
}

// Make the hand empty
void Hand::reset() {
    handSize = 0;
}

// Calculate the score of the hand
int Hand::getPoints() {
    int points = 0;
    int acesFound = 0;

    // for the mean time we calculate all aces as one and:
    for(int i = 0; i < getHandSize(); i++) {
        if(hand[i].getValue() == 1)
            acesFound++;

        points += hand[i].getValue();
    }

    // Convert each ace as 11 to reach the maximum points:
    // exceed the 21 points
    for(int i = 0; i < acesFound; i++)
```cpp
#include "Player.h"

// Initialize the default properties of like the funds
Player::Player() {
    fundsAvailable = 1000;
}

// Places a bet
bool Player::makeBet(int paramBetAmount) {
    if(fundsAvailable - paramBetAmount < 0)
        return false;

    betAmount = paramBetAmount;
    return true;
}

// When a player wins the bet amount he or she have placed
void Player::addBetAmountToFunds() {
    fundsAvailable += betAmount;
    betAmount = 0;
}

// When a player loses the bet amount he or she have placed
void Player::removeBetAmountFromFunds() {
    fundsAvailable -= betAmount;
    betAmount = 0;
}
```

Welcome to Black Jack 1.0
There are only 2 players in this game (you and the dealer).
As a start, you are allotted $1000 funds available in your account.
Once your funds are depleted or if you have bet $0 then the game stops.

You have $1000
Place your bet: 50
Your bet has been successfully placed.

Dealing 2 cards for you...
You have
- 10-spade
- 7-clover

Your hands score is: 17
What would you like to do?
[1] Hit another card
[2] Stay
Option: 1

You have a new card: 10-heart
Your cards are:
- 10-spade
- 7-clover
- 10-heart

Your hands score is: 27
What would you like to do?
[1] Hit another card
[2] Stay
Option:
Source #2  Taken from: http://forum.devmaster.net/t/black-jack-in-c/11100

card.h

enum suit {clubs, diamonds, hearts, spades};

class Card {
public:
    Card(suit = clubs, int = 1);
    suit su() {return f; }
    int value() {return v; }
    void write_out();

private:
    suit f;
    int v;
    int s;
};

card.cpp

#include "card.h"
#include <cassert>
#include <iostream>
using namespace std;

Card::Card(suit ss, int vv)
{
    assert(vv >= 1 && vv <= 13);
    s = ss;
    v = vv;
}

void Card::write_out()
{
    const char *tab1[] = {"Clubs", "Diamonds", "Hearts", "Spades"};
    const char *tab2[] = {"Jack", "Queen", "King"};
    cout << tab1[ s ] << ' ';  
    if (v == 1)
        cout << "Ace";
    else if (v <= 10)
        cout << v;
    else
        cout << tab2[v - 11];
}

cardstack.h

#include "card.h"
#include <vector>

using namespace std;

class Cardstack {
public:
Cardstack() {stack.reserve(52);}  
void throw_cards() {stack.clear();}  
int number_cards() { return stack.size();}  
Card look_at(int no);  
Card deal_top();  
void lay_top(Card k);  
void new_pack();  
void shuffle();
private:  
    vector<Card> stack;
};

cardstack.cpp

#include "cardstack.h"
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

Card Cardstack::look_at(int no)
{
    return stack.at(stack.size() - no);
}
Card Cardstack::deal_top()
{
    Card top = stack.back();
    stack.pop_back();
    return top;
}
void Cardstack::lay_top(Card k)
{
    stack.push_back(k);
}
void Cardstack::new_pack()
{
    stack.clear();
    for (suit s=clubs; s<spades; s=suit(s+1))
    for (int v=1; v<=13; v++)
        stack.push_back(Card(s,v));
}
void Cardstack::shuffle()
{
    srand(time(0));
    for (int i=1; i<1000; i++)
    {
        int n1 = rand() % stack.size();
        int n2 = rand() % stack.size();
        Card temp = stack[n1];
        stack[n1] = stack[n2];
        stack[n2] = temp;
    }
}
player.h

#include "cardstack.h"

class Player {

public:
    Player(Cardstack& cardpack, bool is_computer)
        : pack(cardpack), computer(is_computer) {}
    int play();

private:
    Cardstack hand;
    Cardstack& pack;
    const bool computer;
    int points();
};

player.cpp

#include "player.h"
#include <iostream>

using namespace std;

main()
{
    Cardstack pack;
    Player you (pack, false);
    Player I (pack, true);
    char answer[10];
    cout << "Welcome to Black Jack, the game of twenty-one!" << endl;

    while (true)
    {
        cout << "New game? "; cin >> answer;
        if (answer[0] != 'y') break;
        pack.new_pack();
        pack.shuffle();
        int p1 = you.play();
        if (p1 > 21)
            cout << "Sorry, you busted. You lose."
                 << endl;
        else if (p1 == 21)
            cout << "You Win!" << endl;
        else
        {
            //the computer must play
            int p2 = I.play();
            if (p2 <= 21 && p2 >= p1)
                cout << "You lost!" << endl;
            else
                cout << "You won!" << endl;
        }
    }
}
points.cpp

#include "player.h"
#include "card.h"
#include <iostream>

using namespace std;

int Player::points()
{
    int p = 0, number_aces = 0;

    for (int i = 1; i <= hand.number_cards(); i++)
    {
        int v = hand.look_at(i).value();
        if (v == 1) {
            p+=14;
            number_aces++;
        } else
        p += v;
    }

    for (int j=1; j <= number_aces && p > 21; j++)
        p -= 13; // counts an ace as 1

    return p;
}

play.cpp

#include "points.cpp"
#include <iostream>

using namespace std;

int Player::play()
{
    bool continue = true;
    int p;

    while (continue)
    {
        Card k = pack.deal_top();
        hand.lay_top(k);
        p = points();
        if (computer) {
            cout << "The computer got "; k.write_out();
            cout << endl;
            if (p >= 16)
            {
                cout << "The computer has " << p << " points" << endl;
                continue = false;
            }
        } else
        { //Person
            cout << "You got "; k.write_out();
        }
    }
}
cout << " and have " << p << " points" << endl;
if (p < 21)
{
    char answer[10];
    cout << "One more card? Indicate 'y' for yes or 'n' for no ";
    cin >> answer; // this needs to be fixed
    continue = answer[0] == 'y';
}
else
    continue = false;
}
hand.throw_cards();
return p;

Pre-Recitation #4 Design – Due Sunday, April 17, 2016
5:00pm on Canvas

Design Plan:

The process of object-oriented analysis typically includes the following steps:

1. **Identify the classes and objects to be used in the program.**
   Remember, a class is a package that consists of data and procedures that perform operations on the data. In order to determine the classes that will appear in a program, the programmer should think of the major data elements and decide what procedures or actions are required for each class.

2. **Define the attributes for each class.**
   A class’s attributes are the data elements used to describe an object instantiated from the class. They are the values needed for the object to function properly in the program.

3. **Define the behaviors for each class.**
   Once the class’s attributes have been defined, the programmer must identify the activities, or behaviors, each class must be capable of performing. In C++, a class’s behavior is its member functions.

4. **Define the relationships between classes.**
   The last step in our object-oriented analysis phase is to define the relationships that exist between and among the classes in a program. The possible relationships may be formally stated as
   - Access
   - Ownership (Composition)
   - Inheritance
   Informally, these three relationships can be described as
   - Uses-a
• Has-a (opposite of Part-of)
• Is-a

The first relationship, access, allows an object to modify the attributes of another object. Normally, an object has attributes not accessible to parts of the program outside the object. These are known as private attributes. An access relationship between two objects means that one object will have access to the other object’s private attributes. When this relationship exists, it can be said that one object uses the other.

The second relationship, ownership, means that one object has another object as one of its members. In OOP terminology, this type of relationship is also called composition. When this relationship exists, it can be said that one object has the other (or one object is part of the other).

The third relationship is inheritance. Sometimes a class is based on another class. This means that one class is a specialized case of the other. When this relationship exists, it can be said that one object is from the other object.

**Testing Plan** – Create a test plan with the test cases (bad, good, and edge cases). What are the expected results?

<table>
<thead>
<tr>
<th>Values</th>
<th>Expected</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>You must have a positive number of players</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Ask the initial playing amount for player 1</td>
<td></td>
</tr>
<tr>
<td>…</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

You must submit your design and test plan to your recitation group on Canvas by Sunday, April 17, 2016 5:00pm, and you must provide 3 peer-to-peer reviews before your recitation date and time.