**Assignment 4**

**Due, Wednesday, October 28, 2020 for maximum 100%**

**Thursday, October 29, 2020 for maximum 90%**

**Friday, October 30, 2020 for maximum 80%**

**Saturday, October 31, 2020 for maximum 70%**

**Assignment Scope**

1. Add member variables to classes
2. Generate getters/setters for member variables
3. Write custom constructors
4. Write methods to accomplish specific tasks for a class
5. Create enumerations as constants
6. Use class ArrayList
7. Use interface Set
8. Use interface List
9. Use class HashSet
10. Use static method from class Collections
11. Compile and run a project
12. Compress a project and submit to Webcourses
13. Decompress compressed project and verify it is a Netbeans project

**References**

1. Netbeans.docx
2. Setting up a project in Netbeans.docx
3. Netbeans right click menu help.docx

**Deliverables**

To complete this assignment you must submit your **compressed Netbeans project** to Webcourses.

**Tasks**

|  |  |
| --- | --- |
| Activity | |
| Casino project | Copy previous assignment to a new project (i.e. copy the project folder and modify the new project) |
| casino package |  |
| IPlayer interface | 1. Create interface **IPlayer** 2. Add method signature **hitOrStand**    1. Access level modifier **public**    2. Return type **int**    3. Empty parameter list |
| Player class | 1. Update class **Player**    1. Implement interface **IPlayer**    2. Add member variables       1. **ArrayList** for only class **Card** as **hand**       2. Integer **score**       3. Boolean **busted**       4. Boolean **play**    3. Generate getters/setters for added member variables    4. Add method **playAgain** to do the following       1. Access level modifier public       2. Return type boolean       3. Variable **choice** data type integer       4. Output to the console if the user wants to play again       5. Save the user’s input in variable **choice**       6. Evaluate variable **choice**          1. If the value is 1, return **true**          2. Else return **false**    5. Write interface method **hitOrStand** to do the following       1. Variable **choice** data type integer       2. Output to the console if the user wants to hit or stand       3. Save the user’s input in variable **choice**       4. Evaluate variable **choice**          1. If the value is **Constants.HIT**, return **Constants.HIT**          2. Else return **Constants.STAND**    6. Modify the getter **getScore** to do the following       1. Set member variable **score** to 0       2. Loop through the ArrayList **hand**, for each Card          1. Update member variable **score** to add the value of the card by calling method **getValue** in class **Card**          2. Return **score** |
| constants package |  |
| Constants class | 1. Add the following:    1. A constant for the players to HIT    2. A constant for the players to STAND    3. A constant for the dealer score to have to hit, value 16    4. A constant for the dealer score to have to stand, value 17    5. A constant for the value of 0 (i.e. ZERO)    6. A constant for the score for BUST, value 21 |
| blackjack package |  |
| BlackJack class | 1. Add member variables    1. Dealer dealer    2. Card card 2. Update the custom constructor to do the following    1. Instantiates class **Dealer**    2. Instantiates class **Card** 3. Update method play to do the following    1. Determine if the player’s cash balance is less than the black jack bet amount       1. If true          1. Output to the console the player does not have enough cash to play          2. Set loop control variable **play** to false    2. Loop while **play** is true       1. Output to the console the player’s cash balance       2. Determine if the player’s cash balance is less than the black jack bet amount          1. If true             1. Output to the console the player does not have enough cash to play             2. Set loop control variable **play** to false             3. Break out of the while loop       3. Update the player’s cash to deduct the black jack bet amount       4. Output to the console that the cards are being dealt       5. Reset the player’s **hand** member variable by passing as an argument a new **ArrayList** object to the setter       6. Reset the player’s **score** member variable by passing as an argument **Constants.ZERO** to the setter       7. Reset the player’s **busted** member variable by passing as an argument the value false to the setter       8. Reset the dealer’s **hand** member variable by passing as an argument a new **ArrayList** object to the setter       9. Reset the dealer’s **score** member variable by passing as an argument **Constants.ZERO** to the setter       10. Loop two times to deal the player two cards           1. Set member variable **card** equal to method **deal** in class **Dealer** passing member variable **deck** as an argument           2. Add the dealt card to the player’s hand by calling method **getHand** in class **Player** then method **add** in class **ArrayList** passing member variable **card** as an argument       11. Output to the console the player’s score       12. Loop two times to deal the dealer two cards           1. Set member variable **card** equal to method **deal** in class **Dealer** passing member variable **deck** as an argument           2. Add the dealt card to the dealer’s hand by calling method **getHand** in class **Dealer** then method **add** in class **ArrayList** passing member variable **card** as an argument       13. Write a **while** loop based on the condition that the player wants to continue to hit by calling method **hitOrStand** in class **Player**           1. Set member variable **card** equal to method **deal** in class **Dealer** passing member variable **deck** as an argument           2. Output to the console the player’s score           3. Evaluate the player’s score to determine if they busted (i.e. their score is greater than 21); if true              1. Output to the console that the player busted              2. Set the **busted** member variable in class **Player** to **true**              3. Break out of the loop       14. Evaluate the **busted** member variable in class Player           1. if true              1. Set member variable **play** equal to method call **playAgain** in class **Player**              2. Continue to the next iteration of the outer while loop           2. Else              1. Output to the console the dealer’s score              2. Write a **while** loop based on the condition that the dealer must continue hit by calling method **hitOrStand** in class **Dealer**              3. Output to the console the dealer’s score              4. Evaluate the dealer’s score to determine if they busted (i.e. their score is greater than 21); if true   Output to the console that the dealer busted  Break out of the loop   * + 1. Call method **results**     2. Set member variable **play** equal to method call **playAgain** in class **Player**  1. Write method **results** to do the following    1. Access level modifier **private**    2. Return type **void**    3. Empty parameter list    4. Output to the console the player’s score    5. Output to the console the dealer’s score    6. Evaluate the dealer and player scores       1. If the dealer’s score is greater than the value 21          1. Output to the console that the dealer busted and player wins $10          2. Update the player’s cash to add 10 to the current value       2. If the dealer’s score is equal to the player’s score          1. Output to the console that it was a push and player wins $10          2. Update the player’s cash to add 10 to the current value       3. If the player’s score is greater than the dealer’s score          1. Output to the console that the player won and player wins $20          2. Update the player’s cash to add 20 to the current value       4. Else, the dealer won          1. Output to the console that the dealer wins the hand |
| Card class | 1. Update class **Card** to do the following    1. Add member variable **value**, data type integer    2. Generate getter for member variable **value**    3. Modify method **getValue** to do the following       1. Set member variable **value** equal to 0       2. Evaluate the **face** member variable          1. For face 2 – 9, set member variable **value** equal to the integer value          2. For face 10, Jack, Queen and King, set member variable **value** equal to 10          3. For face Ace, set member variable **value** equal to 11       3. Return **value** 2. Write method **toString** to do the following    1. Access level modifier **public**    2. Return type **String**    3. Empty parameter list    4. Declare variable **card** data type **String**, set it equal to concatenation of the card face and suit (i.e. this will help with debugging and verifying the methods are functioning correctly) |
| Deck class | 1. Update class **Deck**    1. Update the custom constructor       1. Comment out all calls to System.out.println()       2. Comment out calls to method **displayDeck** |
| IDealer interface | 1. Create interface **IDealer** 2. Add method signature **hitOrStand**    1. Access level modifier **public**    2. Return type **int**    3. Empty parameter list 3. Add method signature **deal**    1. Access level modifier **public**    2. Return type **Card**    3. Parameter list includes class **Deck** **deck** |
| Dealer class | 1. Update class **Dealer**    1. Implement interface **IDealer**    2. Add member variables       1. **ArrayList** for only class **Card** as **hand**       2. Integer **score**    3. Generate getters/setters for added member variables    4. Write interface method **hitOrStand** to do the following       1. Evaluate member variable **score**          1. If the value less than or equal to **Constants.DEALER\_HIT**, return **Constants.HIT**          2. Else return **Constants.STAND**    5. Write interface method **deal** to do the following       1. Instantiate an instance of class **Card card**       2. Instantiate an instance of class **Iterator** for only data type class **Card** set equal to class **Deck**, method **getDeck**, method **iterator** in class **HashSet**       3. If the iterator **hasNext** is true          1. Set variable **card** equal to method **next** in class **Iterator**       4. Remove the dealt card from the **deck** by calling method **getDeck** in class **Deck** then method **remove** in class **HashSet** passing object **card** as an argument    6. Modify the getter **getScore** to do the following       1. Set member variable **score** to 0       2. Loop through the ArrayList **hand**, for each Card          1. Update member variable **score** to add the value of the card by calling method **getValue** in class **Card**          2. Return **score** |
| Casino application |  |
| Test Case 1 | Test Case 1 passes |
| Test Case 2 | Test Case 2 passes |
| Test Case 3 | Test Case 3 passes |
| Test Case 4 | Test Case 4 passes |
| Test Case 5 | Test Case 5 passes |
| Test Case 6 | Test Case 6 passes |
| Test Case 7 | Test Case 7 passes |
| Test Case 8 | Test Case 8 passes |
| Test Case 9 | Test Case 9 passes |
|  | Source compiles with no errors |
|  | Source runs with no errors |
|  | Source includes comments |

**Perform the following test cases**

|  |  |  |
| --- | --- | --- |
| Test Cases | | |
|  | **Action** | **Expected outcome** |
| Test Case 1 | **Project view** | Completed project view should look like **Figure 1** |
| Test case 2 | **Run application**  **User enters name and cash amount**  **User enters value 1 to select Black Jack** | The console window should look similar to Figure 2 Player’s cash balance and the first deal for the initial deal of cards |
| Test Case 3 | **User enters value 1 to HIT and clicks <Enter>**  **Player’s score busts** | The console window displays similar output to Figure 3 Player prompted to stand or hit and busted |
| Test Case 4 | **User enters value 1 and <Enter> to play another hand** | Output to the screen repeats **Figure 2 Player’s cash balance and the first deal** |
| Test Case 5 | **User enters value 0 to STAND and clicks <Enter>**  **Player wins** | The dealer’s score and results of the hand displays to the screen similar output to Figure 5 Player prompted to stand or hit and the player wins the hand |
| Test Case 6 | **User enters value 0 to STAND and clicks <Enter>**  **It is a push** | The dealer’s score and results of the hand displays to the screen similar output to Figure 6 Player prompted to stand or hit and it is a push |
| Test Case 8 | **User enters value 0 to STAND and clicks <Enter>**  **Dealer wins** | The dealer’s score and results of the hand displays to the screen similar output to Figure 7 Player prompted to stand or hit and the dealer wins the hand |
| Test Case 8 | **User enters value 1 and <Enter> to play another hand**  **Dealer busts** | The dealer’s score and results of the hand displays to the screen similar output to Figure 8 Player prompted to stand or hit and the dealer busts |
| Test Case 9 | **If the player does not have at least $10, they cannot play** | Output to the screen should be similar to Figure 9 Player is out of cash |

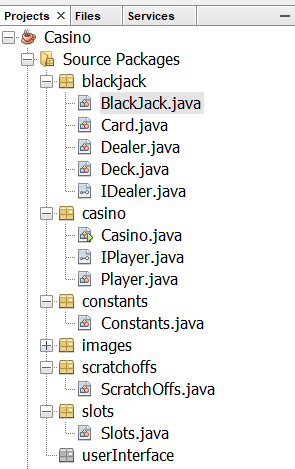


Figure 1 Project View

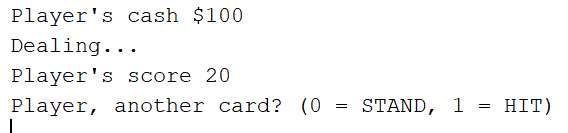


Figure 2 Player’s cash balance and the first deal

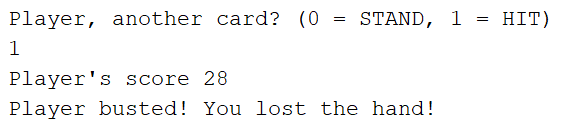


Figure 3 Player prompted to stand or hit and busted



Figure 4 Player prompted to play another hand

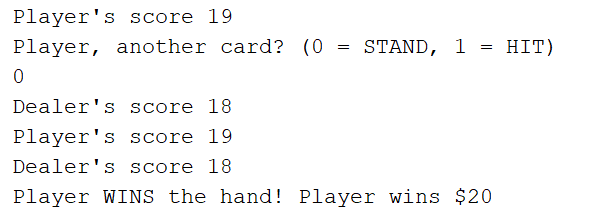


Figure 5 Player prompted to stand or hit and the player wins the hand

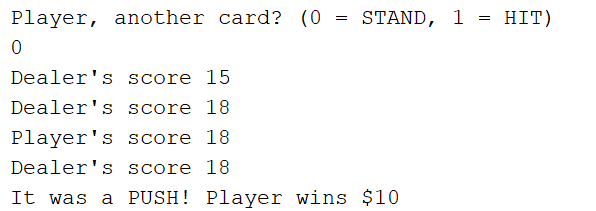


Figure 6 Player prompted to stand or hit and it is a push

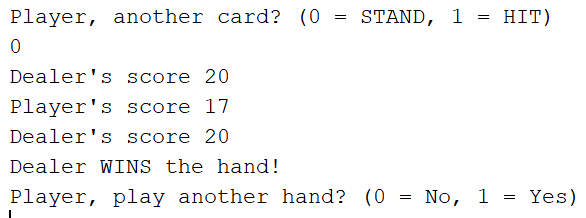


Figure 7 Player prompted to stand or hit and the dealer wins the hand

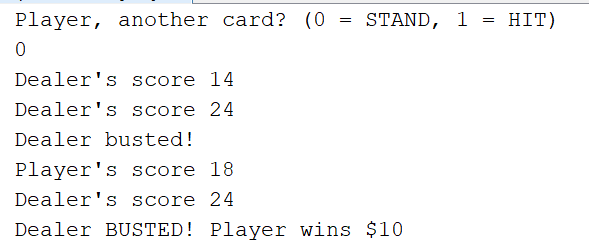


Figure 8 Player prompted to stand or hit and the dealer busts



Figure 9 Player is out of cash