FULL SAIL UNIVERSITY

PG2 – LAB: BLACKJACK OBJECTS

CONTENTS

FULL SAIL UNIVERSITY

OVERVIEW

You are going to create the classes and menu for the Blackjack project.

NOTE: Your lab must follow the specifications listed below. If you instead use code from the internet, you will get a 0 for the Lab.

LECTURE VIDEOS:

Classes Channel
Inheritance Channel
Polymorphism Channel
Unit Tests Channel
Good Programming Channel

FULL SAIL UNIVERSITY

PART A - CLASSES

Part A-1: Setup

A **C# .NET Core console application and class library** have been provided for you in your **GitHub repo**. **Use the provided solution.**

Create the classes and enums in the class library.

The menu should be handled in the console application.

GRADING: 5 POINTS

COMMON MISTAKES:

-3: you did not create the classes in the class library

Part A-2: enums

Create 2 enums to represent the data for a Suit and Face.

CardSuit: Spades, Hearts, Clubs, Diamonds

CardFace: A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K

LECTURE VIDEOS:

Classes Channel #21 Class Challenge (enums)

GRADING: 5 POINTS

COMMON MISTAKES:

-1: even though C# lets you, your enums should not have the same value. Jack, Queen and King should have different enum values. It's because you can't write code to distinguish between them, especially when you need to print the face.

FULL SAIL UNIVERSITY

Part A-3: The Card Class

Create a Card class. The properties should have private setters.

Add a **constructor** to the Card class that takes 2 parameters for face and suit. Also add a Print method that will print the card starting at the specified x,y position in the console window.

LECTURE VIDEOS:

Classes Channel #05 **Create Class**
Classes Channel #10 **Properties**
Classes Channel #11 **Properties Example**
Classes Channel #12 **Properties Challenge**
Classes Channel #13 **Constructors**
Classes Channel #14 **Constructors Example**
Classes Channel #15 **Constructors Challenge**
Classes Channel #16 **Methods**
Classes Channel #17 **Methods Challenge**

PROPERTIES

NAME	TYPE	COMMENTS
Suit	CardSuit	Make the setter private
Face	CardFace	Make the setter private

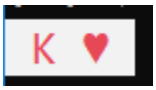
METHODS

NAME	RETURNS	PARAMETERS	COMMENTS
Card		face, suit	A parameterized ***constructor*** for the **Card** class. Set the properties of the class to the values passed in the parameters.
Print	void	x,y	Prints the card starting at the x,y position in the console.

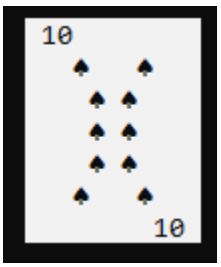
CARD APPEARANCE

Using the functions of the Console class, come up with a way of representing the cards as more than just a number and a single character for the suit. What about writing a white box to the screen, with red or black letters depending on the suit?

The ***minimum*** should be a different background color for the card (white?), a symbol for the suit (♣ ♠ ♦ ♥), and the card face.



Or you could go bigger!



GRADING: 15 POINTS

COMMON MISTAKES:

-5: no code for the Print method in Card

-2: you are not printing the symbols for the suits. to print the suit symbols, at the beginning of Main, you need to set the Console.OutputEncoding to something like UTF8 or Unicode. Then print the Unicode value for the suit symbol.

Part A-4: The Hand Class

You'll want a Hand class to hold the cards for a player or dealer. Each player is dealt cards. Those cards that the player has is consider the player's Hand. A Hand class can have **data** (list of cards) and **behavior** (AddCard).

AddCard should take a card as a parameter and add it to the list of cards for the Hand.

Print should take x,y parameters. They will serve as the starting top-left coordinates for where to start printing the cards. **NOTE: this method should call the Print method of each card in _cards.** It should not actually print the cards – that is the responsibility of the Card class. The Hand Print method should only determine ***where*** (the x and y) each card will be printed. The starting y position for each card would be the same but the starting x position of each card will be different (meaning the cards will be laid out horizontally).

FULL SAIL UNIVERSITY

LECTURE VIDEOS:

Classes Channel #06 **Fields**

Classes Channel #07 **Fields Example**

Classes Channel #08 **Fields Challenge**

FIELDS

NAME	TYPE	COMMENTS
_cards	List<Card>	Initialize in the constructor or in the declaration. Make this field ***protected***.

METHODS

NAME	RETURNS	PARAMETERS	COMMENTS
AddCard	void	Card	Adds the card to the list of cards
Print	void	x,y	Calls the Print method of each card in _cards. Prints the cards horizontally, the y would be the same for each card but the x would change.

GRADING: 10 POINTS

COMMON MISTAKES:

-1: in Hand, you need to initialize _cards (_cards = new List<Card>())

-2: the Hand Print method should loop over _cards and call the Print method of each card

-5: no code for the Print method in Hand

FULL SAIL UNIVERSITY

PART B – FACTORY, DECK, MENU

Part B-1: The Card Factory class

Create a static Factory class that will have a static method for creating cards.

LECTURE VIDEOS:

Inheritance Channel #02 **Static Classes**
Inheritance Channel #03 **Static Classes Example**
Inheritance Channel #04 **Static Classes Challenge**

METHODS

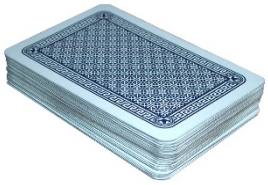
NAME	RETURNS	PARAMETERS	COMMENTS
CreateCard	Card	face, suit	A static method that creates a Card instance using the parameters and returns it.

USAGE EXAMPLE

```
Card card = Factory.CreateCard(CardFace.Ace, CardSuit.Hearts);
```

GRADING: 5 POINTS

Part B-2: The Deck Class



You can't play a card game without a deck of cards, so you'll want to add a Deck class. The Deck class has **data** (list of cards) and **behavior** (Shuffle, Deal).

The **constructor** for the Deck class should generate all 52 cards (4 suits * 13 cards per suit). Call the CreateAllCards method to fill the list of cards. **Call the Card Factory to create the card instances.**

Shuffle should reorder the cards in the list to mimic real-life shuffling.

Deal should return 1 card from the top of the deck. You'll need to consider what to do when the deck is empty. The dealer will use the Deal method to get a card from the deck and add it to the player's / dealer's hand.

FULL SAIL UNIVERSITY

FIELDS

NAME	TYPE	COMMENTS
_cards	List<Card>	Initialize in the constructor or in the declaration

METHODS

NAME	RETURNS	PARAMETERS	COMMENTS
Deck		(none)	A default **constructor** that initializes the list of cards. Call CreateAllCards to fill the list.
CreateAllCards	void	(none)	Creates all 52 cards and puts them in the list of cards.
Deal	Card	(none)	Returns a card from the list of cards. Recreate the deck if the list of cards is empty.
Shuffle	void	(none)	Reorders the cards in the list to mimic real-life shuffling

GRADING: 10 POINTS

COMMON MISTAKES:

-1: in the Deck's Deal method, when you run out of cards, you should recreate the list of cards and call Shuffle.

-1: _cards in Deck should not be public. Use the Deal method to get a card from the deck.

-1: in Deal, you need to remove the card from the list of cards or else you'll deal the same card every time.

-2: not calling CreateAllCards when you need to create the 52 cards.

-2: not calling the factory.

Part B-3: The Menu

Add a menu loop to the Main method in Program.cs of your Console application. Your game's main menu should give them 3 options: Play Blackjack, Shuffle & Show Deck, Exit.

1. Play Blackjack.
    a. This is the menu entry to start playing blackjack. (**Complete this part for the final Blackjack project**)
2. Shuffle & Show Deck.
    a. This option first shuffles the deck and then displays all the cards of the deck. You can use the Deal method of the Deck to get each card to print.
3. Sample Hands

a. This option should be completed in **Part C-4**.

4. Exit
    a. exits the app

```
1. Play Blackjack
2. Shuffle and Show Deck
3. Sample Hands
4. Exit
Choice?
```

GRADING: 5 POINTS

COMMON MISTAKES:

-1: the menu does not loop

-3: no code for the shuffle and show deck menu option

PART C – INHERITANCE, POLYMORPHISM

For Part C, you will add new classes that inherit from the classes in Part A. These classes provided specialized behavior for the Blackjack game.

Part C-1: BlackjackCard class

Create a BlackjackCard class that derives from the Card class from Part A. Add a Value property.

Value is the Blackjack value of the card: K = 10, Q = 10, J = 10, 10 = 10, 9 = 9, etc. Aces are the only cards whose value changes based on the other cards in the hand. Aces can either be valued at 11 or 1 depending on which gives the hand a better NON-BUSTING score. For aces, just decide what default value you want to give them: 1 or 11. Your choice will impact how you score the hand in the BlackjackHand class (see Part B-2).

LECTURE VIDEOS:

Inheritance Channel #05 **Derive From Base**
Inheritance Channel #06 **Constructors**
Inheritance Channel #07 **Derive Example**
Inheritance Channel #08 **Derive Challenge**

PROPERTIES

NAME	TYPE	COMMENTS
Value	Int	The Blackjack value of the card instance

METHODS

NAME	RETURNS	PARAMETERS	COMMENTS
BlackjackCard		face, suit	A parameterized ***constructor*** for the **BlackjackCard** class. Call the base constructor and calculate the Value based on the Face.

 GRADING: 10 POINTS

COMMON MISTAKES:

-2: in BlackjackCard, Value should be a property, not a field.

Part C-2: Update the Factory

Add a new method to the Card Factory that will create an instance of the BlackjackCard.

METHODS

NAME	RETURNS	PARAMETERS	COMMENTS
CreateBlackjackCard	Card	face, suit	A static method that creates a BlackjackCard instance using the parameters and returns it.

USAGE

```
Card card = Factory.CreateBlackjackCard(CardFace.Ace, CardSuit.Hearts);
```

 GRADING: 5 POINTS

COMMON MISTAKES:

-2: CreateBlackjackCard in the Factory should return a new BlackjackCard, not new Card.

-2: CreateBlackjackCard should not recursively call CreateBlackjackCard.

-1: the CreateBlackjackCard was not marked as static

FULL SAIL UNIVERSITY

Part C-3: BlackjackDeck class

Create a BlackjackDeck class that derives from the Deck class. **Override** the CreateAllCards method to call the **CreateBlackjackCard** method of the Factory. Fully-override the base method (do not call the base).

LECTURE VIDEOS:

Poly Channel #03 **Overriding**
Poly Channel #04 **Overriding Example**
Poly Channel #05 **Overriding Challenge**

METHODS

NAME	RETURNS	PARAMETERS	COMMENTS
CreateAllCards	void	(none)	**Fully Override** the base AddCard method. Creates all 52 cards and puts them in the list of cards (_cards). Call the **CreateBlackjackCard** of the factory.

GRADING: 5 POINTS

COMMON MISTAKES:

-2: not marking the method with override

-2: calling the base.CreateAllCards

Part C-4: BlackjackHand class

Create a BlackjackHand class that derives from the Hand class from Part A. A Blackjack Hand has a **score property** as it pertains to the Blackjack game. You will need to **override** the AddCard method to update the score of the hand after calling the base AddCard method. You will need to **override** the Print method: **print the score** for the player only and if it's a dealer hand, hide the first card.

The **Score** is the sum of the values of all the cards in the Hand. The Score should be the best score possible that is closest to 21. Aces make scoring tricky because the Aces value could change based on the other cards in the Hand. For instance, if the player has these cards in the Hand (Ace, 8), the score should be 19 (11 + 8). If a 6 card is added to the Hand, the Score would then become 15 (1 + 8 + 6), not 25 (11 + 8 + 6).

FULL SAIL UNIVERSITY

LECTURE VIDEOS:

Method Channel #14 **Optional Parameters**
Method Channel #15 **Optional Parameters Example**
Method Channel #16 **Optional Parameters Challenge**

PROPERTIES

NAME	TYPE	COMMENTS
Score	Int	The Blackjack score of the hand. Make the setter private.
IsDealer	Bool	True if the hand is the dealer's hand. Default the value to false.

METHODS

NAME	RETURNS	PARAMETERS	COMMENTS
BlackjackHand	(none)	isDealer	Make the isDealer parameter an optional parameter with the default value being false. Use isDealer to set the IsDealer property.
AddCard	void	Card	**Override** the AddCard method to update the score of the hand after calling the base AddCard method
Print	void	x,y	**Override** the Print method of the Hand class. In addition to printing the cards, print the score for the player only and if it's a dealer hand, hide the first card.

NOTE: to update the score, you'll have to cast the Card card to a BlackjackCard that has a Value property.

GRADING: 15 POINTS

COMMON MISTAKES:

-1: in BlackjackHand, the isDealer parameter in the constructor should be optional (ex: bool isDealer = false). Then you could remove the default constructor.

-1: for printing the blackjackhand, if the hand is a dealer, then hide the first card but print the rest of the cards.

-5: no code for the Print method in BlackjackHand

-2: BlackjackHand's Print method does nothing if IsDealer is false. It should call base.Print.

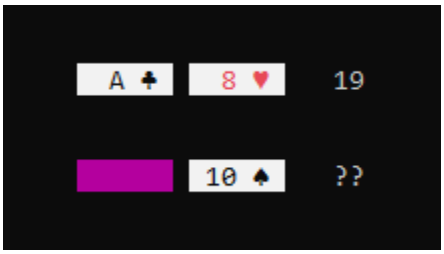
FULL SAIL UNIVERSITY

Part C-5: Sample Hands Menu

Add code for the “Sample Hands” menu option.

- Create an instance of **BlackjackDeck**
- Create 2 **BlackjackHand** instances (make one of them a dealer hand).
- **Add 2 cards** to each hand. Use the **BlackjackDeck** instance to get cards.
- Call **Print** on each hand so that they will print on the screen correctly.

Sample:



LECTURE VIDEOS:

Classes Channel #18 **Instances**
Classes Channel #19 **Instances Example**
Classes Channel #20 **Instances Challenge**
Classes Channel #21 **Class Challenge**

GRADING: 5 POINTS

COMMON MISTAKES:

-2: only printing 1 hand

-2: not creating a dealer hand

-2: not using a BlackjackDeck to get the cards

FULL SAIL UNIVERSITY

LAB 2: RUBRIC

Part A

FEATURE	VALUE
A-1: Setup	5
A-2: Enums	5
A-3: The Card Class	15
A-4: The Hand Class	10
TOTAL	35

Part B

FEATURE	VALUE
B-1: The Card Factory Class	5
B-2: The Deck Class	10
B-3: The Menu	10
TOTAL	25

Part C

FEATURE	VALUE
C-1: BlackjackCard Class	10
C-2: Update the Factory	5
C-3: BlackjackDeck Class	5
C-4: BlackjackHand Class	15
C-5: Sample Hands Menu	5
TOTAL	40

FULL SAIL UNIVERSITY

PROGRAMMER'S CHALLENGE

As with every programmer's challenge, remember the following…

1. Do the rubric first. Make sure you have something to turn in for the assignment.
2. When attempting the challenge, don't break your other code.
3. You have other assignments so don't sacrifice them to work on the challenges.

Unit Test Challenge

Use the provided unit test project to test the AddCard method of the BlackjackHand class.

The scoring of your Blackjack hand is ***critical*** to your game working correctly.

- Create an instance of the BlackjackHand
- Call AddCard twice and add 2 Blackjack cards: an Ace and an 8.
- **Test** that the score should be 19.
- Add a Ten card to the hand using the AddCard method.
- **Test** that the score should still be 19.