

# PG2 – LAB: BLACKJACK OBJECTS

## CONTENTS

Overview .....	2
Part A - Classes .....	2
Part A-1: Setup .....	2
Part A-2: enums .....	2
Part A-3: The Card Class .....	3
Part A-4: The Menu .....	4
Part B – Factory, Deck, Menu .....	6
Part B-1: The Hand Class .....	6
Part B-2: The Card Factory class .....	7
Part B-3: The Deck Class .....	7
Part C – Inheritance, Polymorphism .....	9
Part C-1: BlackjackCard class .....	9
Part C-2: Update the Factory .....	10
Part C-3: BlackjackDeck class .....	10
Part C-4: BlackjackHand class .....	11
Part C-5: Sample Hands Menu .....	12
Lab 3: Rubric .....	13
Part A .....	13
Part B .....	13
Part C .....	14
Programmer’s Challenge .....	14
Unit Test Challenge .....	14

## 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.**

## PART A - CLASSES

### Part A-1: Setup

#### Lab Overview Video

[Part A-1 Overview](#)

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 BlackjackClassLibrary project.**

The menu should be handled in Main in Program.cs in the Lab3 project.



GRADING: 5 POINTS

#### COMMON MISTAKES:

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

### Part A-2: enums

#### Lecture Videos

[Class Challenge](#) (enums)

#### Lab Overview Video

[Part A-2 Overview](#)

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



#### 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.

## Part A-3: The Card Class

### Lecture Videos

[Create Class](#)

[Properties](#)

[Properties Example](#)

[Properties Challenge](#)

[Constructors](#)

[Constructors Example](#)

[Constructors Challenge](#)

[Methods](#)

[Methods Challenge](#)

### Lab Overview Video

[Part A-3 Overview](#)

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.



#### 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 <b>constructor</b> for the <b>Card</b> 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.

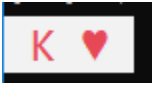
#### CARD APPEARANCE



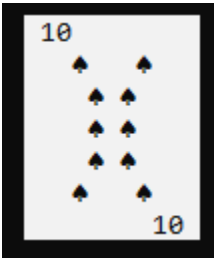
FULL SAIL  
UNIVERSITY

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 Menu

### Lab Overview Video

[Part A-4 Overview](#)

Add a menu loop to the Main method in Program.cs of your Console application. Your game's main menu should give them 5 options: Sample Card, Shuffle & Show Deck, Sample Blackjack Hands, Play Blackjack, Exit.

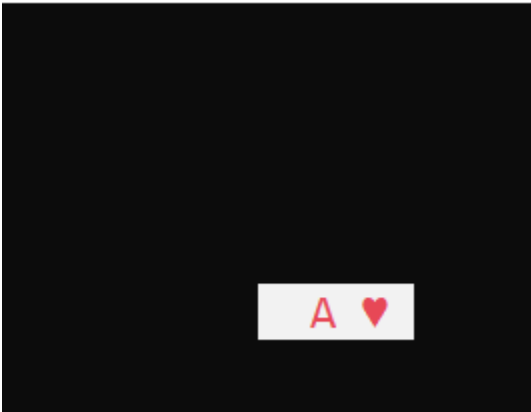
1. Sample Card
  - a. Create a card and call the Print method to show the card in the console.
2. Shuffle & Show Deck.
  - a. This option first shuffles the deck and then displays all the cards of the deck. **Call the Print method of the deck.**  
This option should be completed in **Part B-3.**
3. Sample Blackjack Hands
  - a. This option should be completed in **Part C-4.**
4. Play Blackjack.
  - a. This is the menu entry to start playing blackjack. **(You will complete this part for Lab 4 - the Blackjack game)**
5. Exit
  - a. exits the app



FULL SAIL  
UNIVERSITY.

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

Example output for Sample Card menu option:



GRADING: 5 POINTS

COMMON MISTAKES:

- 1: the menu does not loop
- 3: no code for the shuffle and show deck menu option

## PART B – FACTORY, DECK, MENU

### Part B-1: The Hand Class

#### Lecture Videos

[Fields](#)

[Fields Example](#)

[Fields Challenge](#)

#### Lab Overview Video

[Part B-1 Overview](#)

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).



#### FIELDS

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

#### METHODS

NAME	RETURNS	PARAMETERS	COMMENTS
<b>AddCard</b>	void	Card	Adds the card to the list of cards
<b>Print</b>	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.
<b>Clear</b>	Void	(none)	Clears the list of cards.



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

## Part B-2: The Card Factory class

### Lecture Videos

[Static Classes](#)

[Static Classes Example](#)

[Static Classes Challenge](#)

### Lab Overview Video

[Part B-2 Overview](#)

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

#### 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

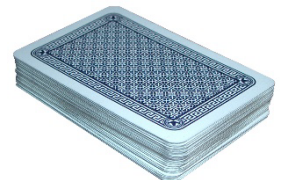
## Part B-3: The Deck Class

### Lab Overview Video

[Part B-3 Overview](#)

You can't play a card game without a deck of cards, so you'll want to add a Deck class.

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 sufficiently shuffle the cards.

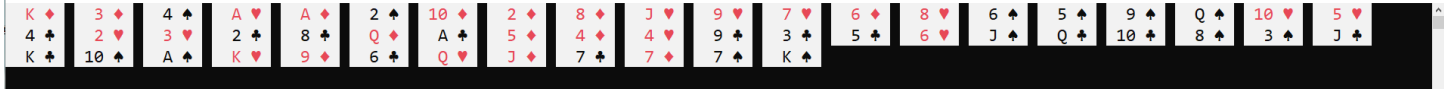
**Deal** should return 1 card from the top of the deck. You'll need to consider what to do when the deck is empty. The dealer will use the Deal method to get a card from the deck and add it to the player's / dealer's hand.



Print should loop over the list of cards and call Print on each card. Make sure the cards do not overlap and properly wrap to the next line.

Once completed, you can now finish the “**Shuffle and Show Deck**” menu option. Create an instance of the Deck and call Print on the instance.

Example output for the Shuffle and Show Deck menu option:



## FIELDS

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

## METHODS

NAME	RETURNS	PARAMETERS	COMMENTS
<b>Deck</b>		(none)	A default <b>constructor</b> that initializes the list of cards. Call CreateAllCards to fill the list.
<b>CreateAllCards</b>	void	(none)	Creates all 52 cards and puts them in the list of cards.
<b>Deal</b>	Card	(none)	Returns a card from the list of cards. Recreate the deck if the list of cards is empty.
<b>Shuffle</b>	void	(none)	Reorders the cards in the list to mimic real-life shuffling
<b>Print</b>	void	(none)	Loops over the list of cards and calls print on each card. Make sure the cards do not overlap.



## 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 C – INHERITANCE, POLYMORPHISM

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

### Part C-1: BlackjackCard class

#### Lecture Videos

[Derive From Base](#)

[Constructors](#)

[Derive Example](#)

[Derive Challenge](#)

#### Lab Overview Video

[Part C-1 Overview](#)

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.

#### PROPERTIES

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

#### METHODS

NAME	RETURNS	PARAMETERS	COMMENTS
BlackjackCard		face, suit	A parameterized <b>constructor</b> for the <b>BlackjackCard</b> 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

### Lab Overview Video

[Part C-2 Overview](#)

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

## Part C-3: BlackjackDeck class

### Lecture Videos

[Overriding](#)

[Overriding Example](#)

[Overriding Challenge](#)

### Lab Overview Video

[Part C-3 Overview](#)

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



## METHODS

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

### Lecture Videos

[Optional Parameters](#)

[Optional Parameters Example](#)

[Optional Parameters Challenge](#)

### Lab Overview Video

[Part C-4 Overview](#)

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).

## 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.



<b>AddCard</b>	void	Card	<b>Override</b> the AddCard method to update the score of the hand after calling the base AddCard method
<b>Print</b>	void	x,y	<b>Override</b> 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.
<b>Clear</b>	Void	(none)	<b>Override</b> the Clear method of the Hand class. Extend it to also clear the score.
<b>Reveal</b>	Void	x,y	If IsDealer is true, temporarily set IsDealer to false and call Print. Otherwise, just call Print.

**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.

## Part C-5: Sample Hands Menu

### Lecture Videos

[Instances](#)  
[Instances Example](#)  
[Instances Challenge](#)  
[Class Challenge](#)

### Lab Overview Video

[Part C-5 Overview](#)

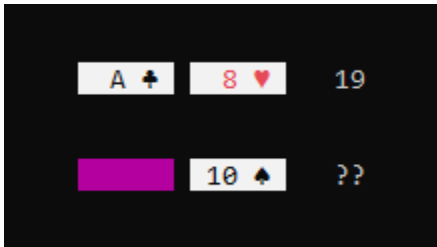
Add code for the "Sample Hands" menu option.

- 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.

Example output of the Sample Blackjack Hands menu option:



FULL SAIL  
UNIVERSITY.



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

## LAB 3: RUBRIC

### Part A

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

### Part B

FEATURE	VALUE
B-1: A-4: The Hand Class	10
B-2: The Card Factory Class	5
B-3: The Deck Class	10
TOTAL	25



FULL SAIL  
UNIVERSITY.

## 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

## 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.