

PG2 – LAB 2: BLACKJACK OBJECTS

CONTENTS

Overview.....	2
Part A - Classes.....	2
Part A-1: Setup	2
Grading: 5 points.....	3
Part A-2: enums.....	3
Grading: 5 points.....	3
Part A-3: ICard	3
Grading: 5 points.....	3
Part A-4: The Card Class	4
PROPERTIES.....	4
METHODS.....	4
Grading: 10 points.....	4
Part A-5: The Card Factory class.....	4
METHODS.....	4
USAGE EXAMPLE	5
Grading: 5 points.....	5
Part A-6: The Deck Class.....	5
FIELDS.....	5
METHODS.....	5
Grading: 10 points.....	5
Part A-7: The Hand Class	6
FIELDS.....	6
METHODS.....	6
Grading: 10 points.....	6
Part A-8: The Menu	7
Grading: 5 points.....	8
Part A-9: Card appearance	8
Grading: 5 points.....	8
Part B – Inheritance, Polymorphism.....	8

Part B-1: BlackjackCard class.....	9
PROPERTIES.....	9
METHODS.....	9
Grading: 10 points.....	9
Part B-2: BlackjackHand class.....	9
PROPERTIES.....	9
METHODS.....	10
Grading: 15 points.....	10
Part B-3: Update the Factory.....	10
METHODS.....	10
USAGE	11
Grading: 5 points.....	11
Part B-4: Unit Test	11
Grading: 10 points.....	11
Lab 2: Rubric	11
Part A.....	11
Part B.....	12

OVERVIEW

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

There will be 2 parts to Lab 2:

- A. ICard, Card, Deck, Hand (Lecture 4)
- B. BlackjackCard, BlackjackHand (Lecture 5)

NOTE: Your lab must follow the specifications listed below. If you instead use code from the internet, you will get a 0 for Lab 2.

PART A - CLASSES

Part A-1: Setup

You will need to create 2 projects for your part A:

1. A C# Console Application

Create the classes, interface and enums in the class library.

The menu can be handled in the console application.

GRADING: 5 POINTS

COMMON MISTAKES:

- 3: you did not create a class library for your classes.
- 2: 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

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 draw the face.

Part A-3: ICard

Create an [interface](#) to represent the public interface for a Card. Note: everything in an interface is public so don't put public on anything.

Add the following **properties** to the interface: Face, Suit. ONLY have a **get** – do not put a **set** on the properties. Use the enums for the Face and Suit.

Add a **Draw** method. It should not return anything but take 2 int parameters: x and y. **NOTE: interfaces cannot have any code so just put the method signature.**

GRADING: 5 POINTS

COMMON MISTAKES:

-2: missing the Draw method signature

Part A-4: The Card Class



Create a Card class. **Implement the ICard interface.** The properties should have private setters.

Add a **constructor** to the Card class that takes 2 parameters for face and suit. Also add a Draw method that will draw 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 constructor for the Card class. Set the properties of the class to the values passed in the parameters.
Draw	void	x,y	Draws the card starting at the x,y position in the console.

GRADING: 10 POINTS

COMMON MISTAKES:

-5: no code for the Draw method in Card

Part A-5: The Card Factory class

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

METHODS

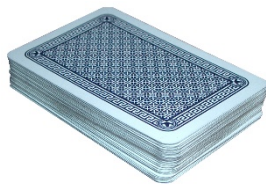
NAME	RETURNS	PARAMETERS	COMMENTS
CreateCard	ICard	face, suit	A static method that creates a Card instance using the parameters and returns it.

USAGE EXAMPLE

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

GRADING: 5 POINTS

Part A-6: 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, Draw).

The **constructor** for the Deck class should generate all 52 cards (4 suits * 13 cards per suit). You can do it in the constructor or call a method that creates the 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.

FIELDS

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

METHODS

NAME	RETURNS	PARAMETERS	COMMENTS
Deck		(none)	A constructor that initializes the list of cards.
Deal	ICard	(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.

Part A-7: 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.

Draw should take x,y parameters. They will serve as the starting top-left coordinates for where to start drawing the cards. **NOTE: this method should call the Draw method of each card in `_cards`.** It should not actually draw the cards – that is the responsibility of the Card class. The Hand Draw method should only determine **where** (the x and y) each card will be drawn. 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
<code>_cards</code>	List<ICard>	Initialize in the constructor or in the declaration. Make this field protected .

METHODS

NAME	RETURNS	PARAMETERS	COMMENTS
AddCard	void	ICard	Adds the card to the list of cards
Draw	void	x,y	Calls the Draw method of each card in <code>_cards</code> . Draw 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<ICard>()`)

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

-5: no code for the Draw method in Hand

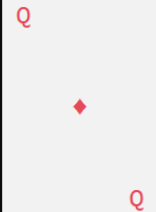


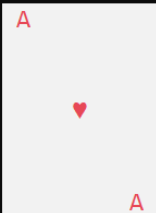
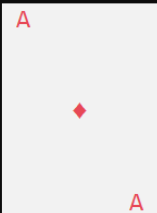
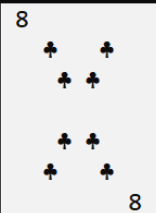
Part A-8: The Menu

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.
3. Exit
 - a. exits the app

```
1. Play Blackjack
2. Shuffle and Show Deck
3. Exit
Choice:
```

```
1. Play Blackjack
2. Shuffle and Show Deck
3. Exit
Choice: 2
```

NOTE: your cards do not have to be this detailed.

GRADING: 5 POINTS

COMMON MISTAKES:

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

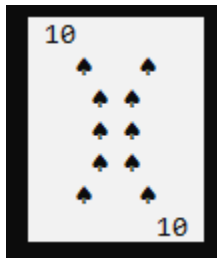
Part A-9: Card appearance

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: 5 POINTS

COMMON MISTAKES:

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

PART B – INHERITANCE, POLYMORPHISM

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

Part B-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).

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 B-2: 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 Draw method: **draw the score** 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.
AddCard	void	ICard	Override the AddCard method to update the score of the hand after calling the base AddCard method
Draw	void	x,y	Override the Draw method of the Hand class. In addition to drawing the cards, draw the score and if it's a dealer hand, hide the first card.

NOTE: to update the score, you'll have to cast the ICard 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 drawing the blackjackhand, if the hand is a dealer, then hide the first card but draw the rest of the cards.
- 5: no code for the Draw method in BlackjackHand
- 2: BlackjackHand's Draw method does nothing if IsDealer is false. It should call base.Draw.

Part B-3: Update the Factory

Add a new method to the Card Factory that will create an instance of the BlackjackCard. **Update the Deck class to call CreateBlackjackCard when creating cards.**

METHODS

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

USAGE

```
ICard 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.
- 1: after updating the Factory with the CreateBlackjackCard method, you were supposed to update the Deck class to call that method instead (you want your game to create blackjack cards)

Part B-4: Unit Test

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

To **add a Unit Test project** easily, right-click the BlackjackHand class and select "Create Unit Test..."

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

GRADING: 10 POINTS

COMMON MISTAKES:

- 10: no unit test project and no unit tests
- 2: the unit tests do not pass

LAB 2: RUBRIC

Part A

FEATURE	VALUE
A-1: Setup	5
A-2: Enums	5

A-3: ICard	5
A-4: The Card Class	10
A-5: The Card Factory Class	5
A-6: The Deck Class	10
A-7: The Hand Class	10
A-8: The Menu	5
A-9: Card Appearance	5
TOTAL	60

Part B

FEATURE	VALUE
B-1: BlackjackCard Class	10
B-2: BlackjackHand Class	15
B-3: Update the Factory	5
B-4: Unit Test	10
TOTAL	40