C212 Final Project - Arcade

Last Updated 12/8/2021 1:00pm EST

For your final project, you are going to need to use what you've learned this semester to create an interactive arcade, where you'll be able to:

- Play games, including
    - Hangman
    - Guess the Number
    - Trivia
    - Blackjack
- Have multiple distinct users play, and will each have their own saved balance and inventory
- Use in-game currency
- Go to a lobby
- There will be an ingame store and place where you can view your inventory

The goal of this project is for your team of 3-4 students to collaborate to produce a fun, object-oriented program that you'd be proud to show off. Like a real arcade, you'll be implementing various games, and you'll also be able to buy your own items using currency that you win in these games. Unlike a real arcade, this will be an (almost) entirely console-driven application. If you are so inclined after this class, you could even extend this to be GUI-based.

This a view of the lobby, with all of the arcade places that you are going to create:

```
Welcome to the C212 arcade, Adam! You're currently in the lobby. Your balance is: 517.0. Where do you want to go next?
1: Lobby (cost: $0.0). Game? no
2: Guess The Number (cost: $5.0). Game? yes
3: Blackjack (cost: $20.0). Game? yes
4: Hangman (cost: $5.0). Game? yes
5: Trivia (cost: $0.0). Game? yes
6: Inventory (cost: $0.0). Game? no
7: Store (cost: $0.0). Game? no
```

Starter code

Download the starter code: here.

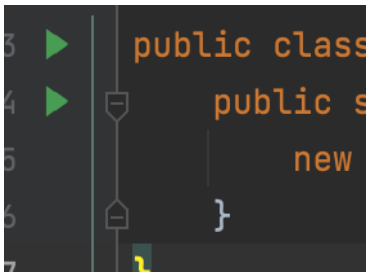
The Hangman and Trivia games will be using http APIs to introduce randomness in those games. For your convenience, we have provided starter code so that you do not have to implement the API calls yourself.

These calls are located in edu.iu.c212.utils.http.HttpUtils. To get a random hangman word, you can call getRandomHangmanWord, and to get a random number of trivia questions, you can call getTriviaQuestions.

Gradle and external libraries

The starter code uses Gradle, which is a build tool and dependency manager standard in modern Java development. You will not need to know what gradle is nor how to use it; instead, you will just need to be aware of the following **important** details:

- There is a build.gradle which contains the dependencies used in this project. You **can** add additional dependencies if you wish, though you must ask Corey or Adam first.
- **<u>To build and run this project outside of IntelliJ IDEA, you will need to type the following in the root of your project:</u>**
    - On windows: gradlew.bat run
    - On mac/linux: ./gradlew run
- If your IDE allows, you can continue to use a run button in ArcadeMain (your main class) to run your program, such as the following:

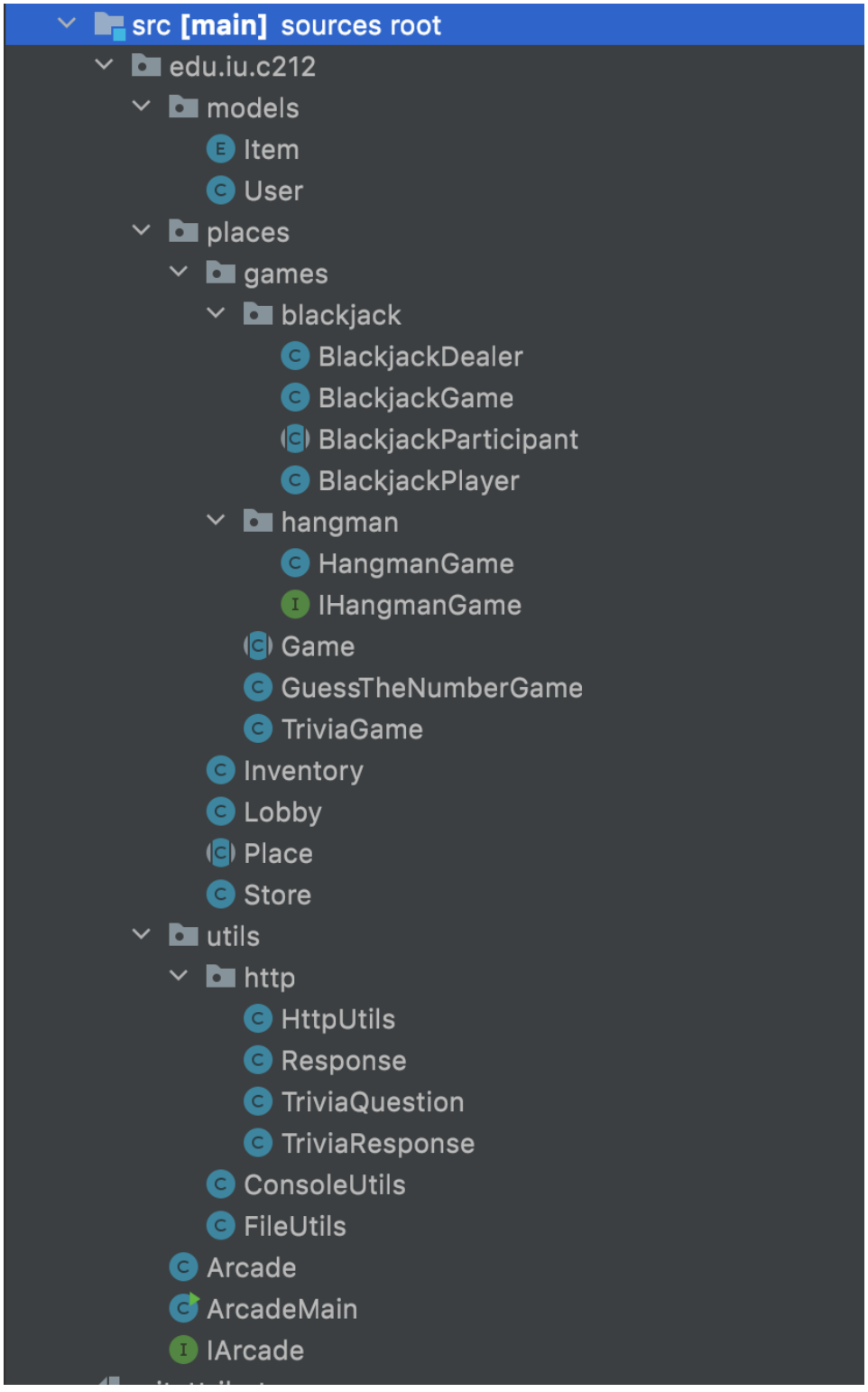


Required implementations in the starter code

You are required to implement methods in ConsoleUtils and FileUtils. There are comments in the code above the methods explaining how to implement these.

File structure

You will end up with a file structure similar to the following:



Arcade

Notes:

- When a user joins a game, buys an item, or sells an item, saveUsersToFile should be called to automatically save to file
- Each line in the users.txt file corresponds to a user. The format for a line is:
    - username|balance|item1,item2...
        - Note that neither username nor any item's readable names can contain a pipe (|)
- You should add getters and setters where appropriate.

public class ArcadeMain (edu.iu.c212.ArcadeMain)
- main(String[] args)
    - This will instantiate the Arcade object, which will handle everything about the Arcade.

public interface IArcade (edu.iu.c212.IArcade)
This interface defines the arcade functionality that the Arcade class will be implementing
- List<User> getUserSaveDataFromFile()
    - This will call FileUtils.getUserDataFromFile, and use System.exit to exit the program if an exception is thrown.
- void saveUsersToFile()
    - This should call FileUtils.writeUserDataToFile to write all users to the txt file
- void transitionArcadeState(String newPlaceNameToGoTo);
    - This should try to transition the currentUser to the new place.
        - If the user doesn't have enough money to go to the place, you should print a warning to the user and transition to the lobby
        - If the user has enough money to go to the place, you should subtract the entry fee from the player, save to file, and then enter the place
- User getUserOnArcadeEntry();
    - This should ask for a username to be entered.
        - If the username is not contained in the users as read from users.txt, a new user must be created and saved to users.txt. A welcome message should be printed
        - If the username already exists, a welcome back message should be printed
- List<Place> getAllPlaces();

public class Arcade implements IArcade (edu.iu.c212.Arcade)
This class concretely implements IArcade
- User currentUser
- List<User> allUsers
    - As read by getUserSaveDataFromFile(). This should contain currentUser
- List<Place> allPlaces
    - All places in the Arcade. Remember, this includes:
        - Lobby
        - Guess the Number
        - Blackjack

        - Hangman
        - Trivia
        - Inventory
        - Store
- public Arcade()
    - This should call getUserOnArcadeEntry() and set currentUser
    - It should instantiate the place list
    - It should transition the Arcade state to the Lobby

Models

public enum Item (edu.iu.c212.models.Item)
These are the purchasable items that can be in your inventory

- String readableName;
    - The name to display in the inventory
- double value;
    - How much the item costs
- Item(String readableName, double value)
- @Override String toString()
    - Must return readable name and the item cost

<u>You are required to add at least 10 different items of varying values!</u>

public class User (edu.iu.c212.models.User)
This represents a user.

- String username
- double balance
- List<Item> inventory
- User(String username, double balance, List<Item> inventory)

Places

The C212 arcade is organized into the following places:

- Lobby
- Inventory
- Store
- TriviaGame
- BlackjackGame
- HangmanGame
- GuessTheNumberGame

As an object-oriented language, you are expected to be able to reason about and create object hierarchies in Java.

We can group all of these places by an **is** relationship - all of these are places, and there can be some common logic between all places. So, we will introduce the **Place** abstract class, which all places will extend.

public abstract class Place (edu.iu.c212.places.Place)
This represents a place.

- String placeName
- Arcade arcade
    - A reference to the Arcade that the Place is in
- double entryFee
    - This should be 0, for places like the Lobby
- abstract void onEnter(User user)
    - What is invoked when the place is entered
- @Override String toString()
    - This should return the place name, the entry fee, and whether the place is a Game or not.

There are several places that are not games and thus only extend the Place class, including:

- Lobby
- Inventory
- Store

We will introduce those now.

Lobby

public class Lobby extends Place (edu.iu.c212.places.Lobby)
This represents the lobby, which you will enter at the beginning of the arcade and after exiting any other places. It should have a $0 entry fee.

- @Override void onEnter(User user)
    - This should use ConsoleUtils.printMenuToConsole to print a menu that 1) welcomes you to the arcade, 2) lets you know your balance, and 3) asks you to select a place to go to from the list of all places in the arcade (arcade.getAllPlaces())
    - It should then go to the selected place

Store

The store is where you'll go to buy and sell things. The available items to buy are the *Item.values()*, but you may only hold up to 3 items in your inventory at a time. Selling an item will only return 50% of its value to your balance. The store should have a $0 entry fee.
You will be required to make two classes: one enum (representing store actions), and one regular class (the store). You are allowed to make the StoreAction enum an inner class of the Store, if you wish.

enum StoreAction (edu.iu.c212.places.Store.StoreAction)

- Should contain buy/sell/leave (remember that enums must be all caps)
- @Override String toString()
    - This should return the enum's name as lowercase.

public class Store extends Place (edu.iu.c212.places.Store)
It should have a $0 entry fee.

- @Override void onEnter(User user)
    - This should loop infinitely until the user selects the leave action
    - You should use ConsoleUtils.printMenuToConsole to read a StoreAction asking whether the user wants to buy, sell or leave.
    - **Sell**
        - If the user has an empty inventory, you should complain and not let them sell anything.
        - You should print a menu to the console asking which item in their inventory to sell and listing all options. Warn the user they can only get 50% of the item value back.
        - You should print a menu asking for confirmation that the user wants to sell the item.
        - If the user selects yes, you should remove the item from the user inventory, increase their balance, and save.
    - **Buy**
        - This should print a menu to the console asking which item to buy and listing all options.
        - If the user doesn't have enough money to buy the item, or if the user's inventory already has 3 items, you should not let the user buy the item.
        - You should print a menu asking for confirmation that the user wants to buy the item.
        - If the user selects yes, you should add the item to the user inventory, decrease their balance, and save.
    - **Leave**
        - This action should stop the loop and transition back to the lobby

Inventory

The inventory place is where the user will go to see their inventory and net worth. You need to recreate the following output:

```
6
Hi, Adam! Your inventory looks like this:
Hamburger: 2 (value: 30.0)
Copper Wire: 1 (value: 30.0)
Total net worth: 560.0
REMEMBER! You can only have 3 items at a time. Sell one by going to the Store
==========
```

public class Inventory extends Place (edu.iu.c212.places.Inventory)
It should have a $0 entry fee.

- @Override void onEnter(User user)
    - This should print to the console all of your items, the amount you have of each item, and their value, as seen in the image above.
    - This should also print your net worth and, if you have 3 items in your inventory, a warning that you can only have 3 items at a time.
    - This should then transport you back to the lobby

Game

The C212 arcade will have 4 games:

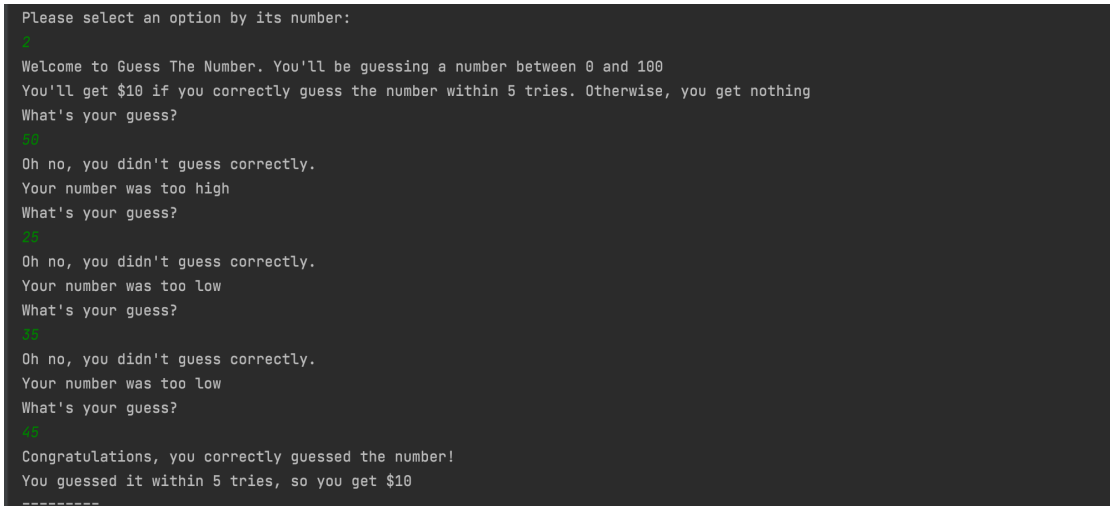
- Guess the Number
- Trivia
- Hangman
- Blackjack

These will be organized under the umbrella of the Game abstract class.
public abstract class Game extends Place (edu.iu.c212.places.games.Game)

Though you must implement all four games, you may choose to implement any game on the console or (especially Blackjack and Hangman) using a GUI.

Guess the Number

You will be creating the following game experience:



public class GuessTheNumberGame extends Game
(edu.iu.c212.places.games.GuessTheNumberGame)
The user will be guessing a random number between 0 and a number of your choice, at least 100. If they guess the number within 5 tries, they get $10. Otherwise, they don't get anything. This game has a $5 entry fee.

- @Override void onEnter(User user)
    - You should welcome the user, and give them up to 5 guesses.
    - You should let them know whether the number is too high or low at each iteration.
    - You **must** use the ConsoleUtils.readIntegerLineFromConsoleOrElseComplainAndRetry method to read a guess from the user.
    - Remember to autosave if the user balance changes
    - Go back to the lobby after the game ends

Trivia

You will be creating the following game experience:

```
Welcome to C212 trivia. You get $2 for every correct answer - there are 5 total questions in this trivia round.
You're on question 1. Ready?
=========
Who is the true moon princess in Sailor Moon?
1: Sailor Mars
2: Sailor Venus
3: Sailor Jupiter
4: Sailor Moon
=========
Please select an option by its number:
1
You got it wrong :( The correct answer is: Sailor Moon
You're on question 2. Ready?
=========
What was the nickname given to the Hughes H-4 Hercules, a heavy transport flying boat which achieved flight in 1947?
1: Spruce Goose
2: Trojan Horse
3: Noah's Ark
4: Fat Man
=========
Please select an option by its number:
4
You got it wrong :( The correct answer is: Spruce Goose
You're on question 3. Ready?
=========
Who was the Prime Minister of Japan when Japan declared war on the US?
1: Hideki Tojo
2: Fumimaro Konoe
3: Michinomiya Hirohito
4: Isoroku Yamamoto
=========
Please select an option by its number:
1
You got it right! You got $2.
You're on question 4. Ready?
=========
```

```
=========
The biggest distinction between a eukaryotic cell and a prokaryotic cell is:
1: The presence or absence of certain organelles
2: The mode of reproduction
3: The presence or absence of a nucleus
4: The overall size
=========
Please select an option by its number:
1
You got it wrong :( The correct answer is: The presence or absence of a nucleus
You're on question 5. Ready?
=========
How old was Adolf Hitler when he died?
1: 65
2: 56
3: 47
4: 43
=========
Please select an option by its number:
1
You got it wrong :( The correct answer is: 56
Aww, good try. You got 1 questions right.
Going back to the lobby...
=========
```

public class TriviaGame extends Game (edu.iu.c212.places.games.TriviaGame)
The user will be guessing the answers to 5 randomly-generated trivia questions. They will get $2 for each correct answer. There will be no entry fee for this game.

- @Override void onEnter(User user)
    - You should use the HttpUtils.getTriviaQuestions method to get the random trivia questions.
    - For each trivia question:
        - You should print “You’re on question #. Ready?” and then wait 1 second before continuing (*hint: Thread.sleep*).
        - The answers should be shuffled.
        - You should use ConsoleUtils.printMenuToConsole to ask the question and get the user selected answer.
        - If the user gets the correct answer, add $2 to their balance and save. Otherwise, let them know what the correct answer was.
    - If the user gets 3 or more questions right by the end, print “Nice! You got # questions right.”
    - Otherwise, print “Aww, good try. You got # questions right.”
    - Then, go back to the lobby

Hangman

You will be creating the following game experience:

```
You've guessed 0 times incorrectly ([]).
The current word is: *********. Please enter a lowercase letter in the following lexicon to guess: [a, b, c, d, e, f, g, h, i, j, k,
  l, m, n, o, p, q, r, s, t, u, v, w, x, y, z]
a
You've guessed 0 times incorrectly ([a]).
The current word is: **a*a****. Please enter a lowercase letter in the following lexicon to guess: [a, b, c, d, e, f, g, h, i, j, k,
  l, m, n, o, p, q, r, s, t, u, v, w, x, y, z]
e
You've guessed 0 times incorrectly ([a, e]).
The current word is: *ea*a**e*. Please enter a lowercase letter in the following lexicon to guess: [a, b, c, d, e, f, g, h, i, j, k,
  l, m, n, o, p, q, r, s, t, u, v, w, x, y, z]
s
You've guessed 1 times incorrectly ([a, e, s]).
The current word is: *ea*a**e*. Please enter a lowercase letter in the following lexicon to guess: [a, b, c, d, e, f, g, h, i, j, k,
  l, m, n, o, p, q, r, s, t, u, v, w, x, y, z]
t
You've guessed 2 times incorrectly ([a, e, s, t]).
The current word is: *ea*a**e*. Please enter a lowercase letter in the following lexicon to guess: [a, b, c, d, e, f, g, h, i, j, k,
  l, m, n, o, p, q, r, s, t, u, v, w, x, y, z]
r
You've guessed 2 times incorrectly ([a, e, s, t, r]).
The current word is: rea*a**e*. Please enter a lowercase letter in the following lexicon to guess: [a, b, c, d, e, f, g, h, i, j, k,
  l, m, n, o, p, q, r, s, t, u, v, w, x, y, z]
d
You've guessed 2 times incorrectly ([a, e, s, t, r, d]).
The current word is: rea*a**ed. Please enter a lowercase letter in the following lexicon to guess: [a, b, c, d, e, f, g, h, i, j, k,
  l, m, n, o, p, q, r, s, t, u, v, w, x, y, z]
l
You've guessed 2 times incorrectly ([a, e, s, t, r, d, l]).
The current word is: rea*a*led. Please enter a lowercase letter in the following lexicon to guess: [a, b, c, d, e, f, g, h, i, j, k,
  l, m, n, o, p, q, r, s, t, u, v, w, x, y, z]
n
```

```
You've guessed 3 times incorrectly ([a, e, s, t, r, d, l, n]).
The current word is: rea*a*led. Please enter a lowercase letter in the following lexicon to guess: [a, b, c, d, e, f, g, h, i, j, k,
  l, m, n, o, p, q, r, s, t, u, v, w, x, y, z]
m
You've guessed 4 times incorrectly ([a, e, s, t, r, d, l, n, m]).
The current word is: rea*a*led. Please enter a lowercase letter in the following lexicon to guess: [a, b, c, d, e, f, g, h, i, j, k,
  l, m, n, o, p, q, r, s, t, u, v, w, x, y, z]
v
You've guessed 4 times incorrectly ([a, e, s, t, r, d, l, n, m, v]).
The current word is: reava*led. Please enter a lowercase letter in the following lexicon to guess: [a, b, c, d, e, f, g, h, i, j, k,
  l, m, n, o, p, q, r, s, t, u, v, w, x, y, z]
i
Congrats, you won with 4 incorrect guesses! You got $15
=========
```

In addition to the HangmanGame class, as good practice, you will also be creating the IHangmanGame interface containing definitions for the two important methods that HangmanGame will implement.

public interface IHangmanGame (edu.iu.c212.places.games.hangman.IHangmanGame)

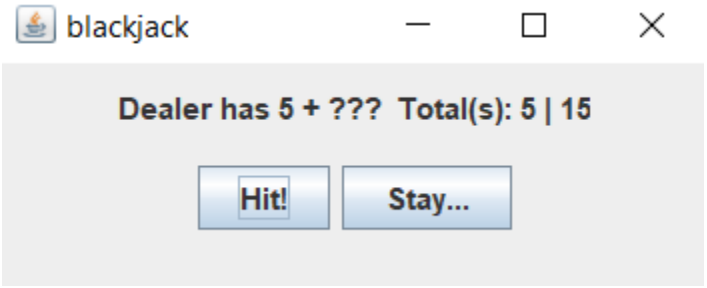
- String getBlurredWord(List<Character> guesses, String word)
    - This will blur out (with asterisks *) all letters in the word that have not been guessed.
- List<Character> getValidLexicon()
    - This will get the lexicon of allowable input characters for the hangman game (this should be all lowercase alphabetical characters).

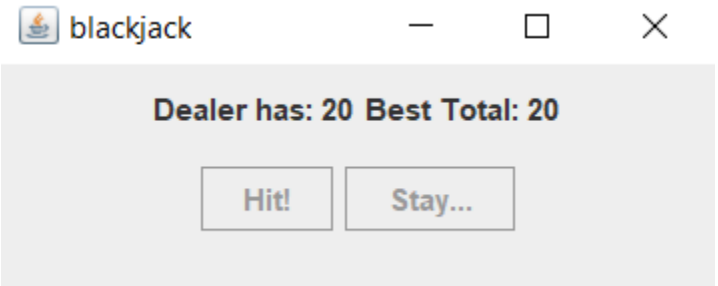
public class HangmanGame extends Game implements IHangmanGame
(edu.iu.c212.places.games.hangman.HangmanGame)
The user will be guessing a randomly generated word. If they guess the word within 6 guesses, they will get $15. The entry fee for this game is $5.

- @Override void onEnter(User user)
    - You should welcome the user, and tell them the game rules.
    - You should retrieve a random word using HttpUtils.getRandomHangmanWord and display the blurred text to the user.
    - You should use ConsoleUtils.readLineFromConsole() to read each guess. If a user enters a character that is not in the lexicon, or else doesn't enter anything, they are penalized by still counting that guess as an incorrect guess. You should display an error message if this is the case.
    - At the beginning of each guess, you should display the blurred text.
    - If, at any point, the user correctly guesses the word, or 6 guesses have been exhausted, the game will end.
        - If the user guessed the word, congratulate them and add $15 to their balance.
        - Otherwise, let them know they lost.
    - After the game ends, go back to the lobby

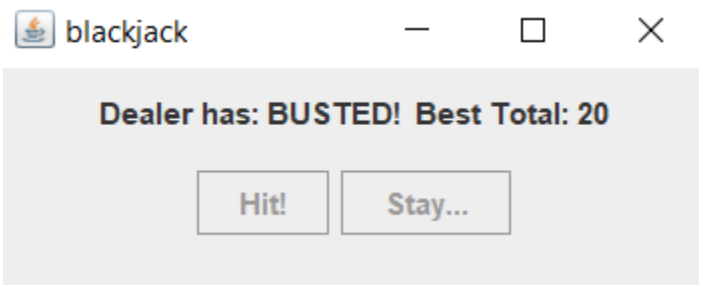
Blackjack

You will be creating the following game experience:






```
You had 20, the dealer had 20
You lost!
```




```
You had 20, the dealer busted!
You won!
```

Blackjack's back, alright! Follow the expectations for Lab 8's BlackjackPlayer game, but now, once the user chooses to "stay", a dealer "AI" will play out the remainder of the game, along with some other enhancements. Note that you will need to do some significant restructuring of your Lab 8 code to get this working -- in fact, *you are allowed to create this game in its entirety using the console*, similar to the other games, if you so choose. There are some technical tricks we'll walk you through to get the Swing GUI to work alongside the rest of the console-based program, but overall transitioning from a completed Lab8 BlackjackPlayer to this project's Blackjack deliverable is not too difficult, and we'd recommend using GUI to create the game. The entry fee is $20, and if the user wins they get $50.

public abstract class BlackjackParticipant
(edu.iu.c212.places.games.blackjack.BlackjackParticipant)
This is the superclass of the player in the game, BlackjackPlayer, and the AI dealer, BlackjackDealer.

- protected static ArrayList<{type}> cards
    - The player and the dealer will be drawing from the same deck of 52 cards without replacement, thus we can contain it in the superclass statically. You are free to represent card objects as you want (e.g. Strings, Integers, a custom class…)
- protected int[] handTotals
- public void hit()
    - Add the value(s) of a randomly-chosen card to handTotals, without replacement
- public abstract int getBestTotal()

public class BlackjackPlayer extends BlackjackParticipant
(edu.iu.c212.places.games.blackjack.BlackjackPlayer)
Note that BlackjackPlayer in this assignment is NOT the same as BlackjackPlayer in Lab 8. Instead, it would be more akin to this project's BlackjackGame class.

- public BlackjackPlayer()
    - Give the superclass the starting full deck of cards, initialize handTotals, and call hit() twice. Ensure you aren't reusing a partial deck of cards when playing the game several times
- public String getCurrentTotalsString()
    - Return the String representation of totals, only displaying two values if they are different and the greater value is <= 21
- public int getBestTotal()

public class BlackjackDealer extends BlackjackParticipant
(edu.iu.c212.places.games.blackjack.BlackjackDealer)

- private {type} shownCard
- private int dealerBest
- public BlackjackDealer()
    - Ensure you create the BlackjackDealer object after the BlackjackPlayer so that the dealer has a deck to draw from. Simply initialize handTotals and call hit() twice.
- @Override public void hit()

        - Override the superclass's method to, in addition to drawing a card without replacement, setting the first drawn card in a game to be the shownCard
- public String getPartialHand()
    - Return the String representation of the dealer's visible hand, e.g. "Q + ???"
- public void play()
    - By the end of the method, dealerBest will either mutate from an initial value of -1 to their best total or stay at -1 if the dealer busts during their turn. This method encompasses the "AI" of the dealer and acts out its full turn in the following way, once the user has chosen to stay, following these rules from Bicycle:
        - "If the total is 17 or more, [the dealer] must [stay]. If the total is 16 or under, they must take a card. The dealer must continue to take cards until the total is 17 or more, at which point the dealer must [stay]. If the dealer has an ace, and counting it as 11 would bring the total to 17 or more (but not over 21), the dealer must count the ace as 11 and [stay]. The dealer's decisions, then, are automatic on all plays"
        - There are no exceptions to these rules -- even if the player has 15 and the dealer already has 16, the dealer MUST hit once.
- public int getBestTotal()

public class BlackjackGame extends Game
(edu.iu.c212.places.games.blackjack.BlackjackGame)

- The members as described in Lab8 should be included here, as well as player, user, and dealer members, and a dealerLabel member that can be updated during events like the player's label
- public BlackjackGame(Arcade arc)
- public void onEnter(User user)
    - Initialize player, dealer, and user members
    - Set up and display the JFrame as in Lab 8
- private class HitButtonListener implements ActionListener
    - public void actionPerformed(ActionEvent e)
        - Use your hit() method on the player
        - Follow other expectations from Lab 8
- private class StayButtonListener implements ActionListener
    - public void actionPerformed(ActionEvent e)
        - Follow other expectations from Lab 8
        - Use your play() method on the dealer

Miscellaneous Notes for Blackjack

- This was not required previously, but now the dealer will receive cards from the deck just like the user. Also note that card draws are taken from a standard 52-card deck without replacement.
- To win the game, the player must have a strictly higher valid total than the dealer (however it is common to return the player's bet if they "tie" the dealer's best total in

many Blackjack rulesets, which you may optionally implement if you'd like). If the player busts, they automatically lose, even if the dealer also busts. If the dealer busts but the player does not, the player wins.
- You need not create an Interface for Blackjack as you did for the other games, though you may. You do, however, need to use the abstract superclass for player and dealer.
- **Extra Credit Opportunity:** As with Lab 8, you are welcome to simplify the program such that there are only two hand value calculations: one that treats all aces as 1, and the other that treats all aces as 11. Of course, in the actual game, you could have, say, a 21-valued hand consisting of an ace (worth 1), another ace (worth 11), and a 9. In your solution, if you manage to allow all valid combinations of ace values correctly, you will receive (minor; check the rubric) extra credit points for the project. Hint: you'll likely need to use some variable-length data structure in place of the handTotals array. If you're going to put in the effort for this stretch goal, we'd recommend doing plenty of testing!

An Aside on Thread Locking

You should notice that, as long as you are managing state and transitions between places in your Arcade correctly, many of these steps are unnecessary for preventing the issue described below. The original text will stay here, but parts that are irrelevant and steps that you do NOT need to follow have been struck out. You DO still need to implement the WindowClosedListener for a GUI-based game and related steps. If you have already implemented the thread locking technique, you don't need to remove it -- it shouldn't have any negative effects. As long as your program doesn't print the lobby message until the window is closed, you're good to go.

In playing around with Swing GUI previously, you may have noticed that a program can concurrently run the window and the console -- like how we could see console messages from our KeyListener while the window was still open. This is because Java runs these processes on separate threads. Concepts like multithreading and thread safety in modern programming languages are quite interesting to learn about but certainly beyond the scope of the course, so we will keep our discussion of them limited. ~~As you're aware with console-based menu systems, only one part of the code is running at a time, which allows us to track what state the user is in. When part of the program is running on another thread simultaneously, we lose this restriction on the user, and they could just leave the window open and continue interacting with the console! So, we'll use a simple trick with Java's Threads to lock the thread of the console while the Swing GUI is open, as well as~~ an Event Listener -- not unlike the ones you've seen -- ~~to unlock the thread when the window is closed.~~ This section only applies if you're making Blackjack a GUI-based game -- but don't let this scare you off from doing so!

When you feel your program is mostly working as intended, follow these step-by-step instructions:

1) ~~Add a private member, Object lock, to your BlackjackGame. The functionality we need from lock is actually part of all Objects, so we'll just use the Object class~~
2) ~~Add a private static member, Thread t, to your BlackjackGame. We'll need to reference it from elsewhere in the project, so making it static is an easy way to do so.~~

3) Change the default close behavior for the JFrame to dispose on close, instead of exiting on close, using frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);, so we don't end the entire program when the window is closed.

4) At the end of onEnter() in BlackjackGame, paste the following code, which will lock the thread:

```
        lock = new Object();
        t = new Thread(() -> {
            synchronized (lock) {
                while (frame.isVisible()) {
                    try {
                        lock.wait();
                    catch (InterruptedException ignored) {}
                }
            }
        });
        t.start();
```

5) Just like you did with HitButtonListener and StayButtonListener, add a subclass WindowClosedListener that implements the WindowListener interface to your BlackjackGame class. It should have several Overrides, which you can find here, but we'll only use windowClosed(). It should first check if the game is ongoing (there are several ways you could do this, like having a boolean member that defaults to true and becomes false when the user busts or stays), and if so, count it as a loss and inform the user that they forfeit the game by closing the window early. If the game is not in progress -- that is, both the player and dealer have finished their turns -- decide who won. As with the other games, print an appropriate message to the console and mutate the user's balance. At the very end of this event, paste the following code, which will unlock the thread:

```
        synchronized (lock) {
            lock.notify();
        }
```

6) Register the WindowClosedListener to the JFrame.

7) Immediately after the line of code that calls onEnter() for the BlackjackGame in your program's main menu loop, paste the following line of code, which locks the thread while the game is running:

```
            BlackjackGame.t.join();
```

Then, add those lines to the try-body of a try-catch statement. As in Step 4, you can simply use catch (InterruptedException ignored) {} as the catch clause.

With these modifications, your program should only print something to the console again after the window is closed by the user. Note that this is a very quick-and-dirty solution for the underlying problem and not best-practice, but it works for our purposes. Please see a TA, preferably Adam or Corey, with questions about getting these steps working if you have ANY difficulties.

Rubric (out of 100 points)

- Arcade (20 points)
  - IArcade interface correct and complete (2)
  - Arcade constructor gets user, reads file, goes to lobby (3)
  - Nothing in ArcadeMain besides Arcade instantiation (2)
  - getUserSaveDataFromFile and saveUsersToFile correct, with error handling (2)
  - transitionArcadeState implemented correctly (3)
  - getUserOnArcadeEntry gets username, maps username to existing user or creates new user and saves (3)
  - Place class (3)
    - toString implemented (1)
    - All places implement Place (2)
  - Entry fees are implemented correctly (2)
- ConsoleUtils (2 points)
  - readIntegerLineFromConsoleOrElseComplainAndRetry implemented correctly (2)
- FileUtils (8 points)
  - User class correct and complete (2)
  - writeUserDataToFile writes all user data to file in the correct format (3)
  - getUserDataFromFile reads all user data from file in the correct format and returns user list (3)
- Lobby (8 points)
  - ConsoleUtils.printMenuToConsole is used to obtain a Place (6)
  - Transition to place done correctly (2)
- Store (15 points)
  - Store loops infinitely until you leave (1)
  - Buy (5)
    - Complains if you have 3 items in inventory or not enough balance to buy item (1)
    - Uses ConsoleUtils.printMenuToConsole to get item to buy (1)
    - Gets user confirmation to buy (1)
    - Buys item correctly (2)
  - Sell (5)
    - Complains if you have empty inventory (1)
    - Uses ConsoleUtils.printMenuToConsole to get item to sell (1)
    - Gets user confirmation to sell (1)
    - Removes item correctly (2)
  - Leave StoreAction goes back to the lobby (2)
  - StoreAction enum is completed with valid toString (2)
- Inventory (5 points)
  - Net worth calculated correctly (1)
  - Warning is printed if necessary (1)
  - All items are printed, along with their quantity (3)
- Games (42 points total)
  - Game interface correct and complete (2 points)

- Trivia (10 points)
    - Uses HttpUtils.getTriviaQuestions correctly (1)
    - Loops through each question, with 1s wait (1)
    - Uses ConsoleUtils.printMenuToConsole to print question and get answer (3)
    - Answers are shuffled (1)
    - Balance is changed and autosave if correct answer, correct output if incorrect answer (2)
    - Endgame summary is correct (2)
- Guess the Number (10 points)
    - Welcome message displayed (2)
    - Generates random number correctly (1)
    - Uses ConsoleUtils.readIntegerLineFromConsoleOrElseComplainAndRetry correctly to read each guess (3)
    - Handles correct guess correctly (2)
    - Summary message displayed (2)
- Hangman (10 points)
    - IHangmanGame interface complete and correct (1)
    - getBlurredWord implementation correct (1)
    - getValidLexicon implementation correct and is used (1)
    - HttpUtils.getRandomHangmanWord is used correctly (1)
    - Welcome message displayed along with blurred word (1)
    - Game end on 6 incorrect guesses, guessing correct word should add to balance (4)
    - Game summary message displayed (1)
- Blackjack (10 points)
    - If not using GUI:
        - (1) for working initializations of hands, etc
        - (1) for BlackjackParticipant
        - (1) for BlackjackPlayer
        - (1.5) per working event listener x 2
        - (3) for BlackjackDealer
        - (2.5) for deciding winner and updating user balance
    - If using GUI:
        - (2) for working initializations of hands, Swing, etc.
        - (1) for BlackjackParticipant
        - (1) for BlackjackPlayer
        - (1) per working event listener x 3
        - (2) for BlackjackDealer
        - (2) for deciding winner and updating user balance
        - (1) ~~for thread locking the console~~ for console not printing anything until after the game has ended
    - (2 points EC): allow for more than 2 handTotals in either solution type