Name: ______________________________

Date: _______________

ACTIVITY 3

Putting It All Together

1. There are several components that are necessary to create a GUI. As your teacher discusses a few of the components and classes, pick one that you're interested in learning more about. Look at the Java GUI tutorial **(https://docs.oracle.com/javase/tutorial/uiswing/components/index.html)** for using swing components and record some information about your chosen class. Some possible things to record include the type of event the component triggers (which corresponds to the type of listener that can be added to the component), and methods that can change the look of the component.

Share your class with a partner.

When creating large-scale projects, separating roles in program code is very important. This is similar to the roles necessary to put on a play. In addition to the actors that speak and move across the stage, there are crew members who are responsible for changing scenes and handling the light and sound, and all of these people are managed by the director.

When creating programs that utilize a GUI, it's desirable to separate the "logic" of the program with the functionality of the GUI as much as possible. This allows for better maintainability of program code, because the layout or look and feel of a program can change often, while the code dealing with the logic can remain untouched. Or, conversely, background processes can be updated and improved without having to change the look and feel of a program. As a class, discuss some other benefits of this separation of roles in programming.

Open the `CelebrityGame.java` file. All of the code described in this activity will be added to `CelebrityGame.java`, unless explicitly stated otherwise.

2. In the declaration section add an instance variable named `celebGameList` that is an `ArrayList` of `Celebrity` objects. What visibility should this variable have? Write the statement to declare the `celebGameList` instance variable with the appropriate visibility below and add this statement to the `CelebrityGame` class.

3. You also need an instance variable named `gameCelebrity` for the current `Celebrity` that is being used in the game. Will its visibility match `celebGameList`? If not, what should its visibility be?

Constructor

4. In the `CelebrityGame` class, make sure the `celebGameList` is instantiated appropriately in the `CelebrityGame` constructor.

Tip

When assigning values to instance variables in a constructor, the type of variable cannot be included. If the type is included before the variable name in the constructor, a local variable of the same name is created and initialized and the instance variables of the object are never initialized. Once the constructor is complete the local variables no longer exist, leaving only the uninitialized instance variables.

5. In the declaration section add an instance variable named `gameWindow` that is a `CelebrityFrame` object. In the constructor initialize the `CelebrityFrame` using the line

```
gameWindow = new CelebrityFrame(this);
```

The `this` keyword is used to provide a reference to the current game to the GUI window so it can be accessed from the GUI components. This GUI will not be used with multiple games, rather it will use the reference to the current game instance.

Setting Up the Game

When playing Celebrity without a computer, the game was prepared by writing the names of celebrities on slips of paper. The clues were not specified ahead of time and instead came from the reader. The preparation will be modeled in the program via the `prepareGame` method in the `CelebrityGame` class. The class `StartPanel`, which is a panel GUI component, will be responsible for prompting the user for input and then allow the game to start once at least one celebrity has been created.

```
/**
 * Resets the game by changing screens.
 */
public void prepareGame()
{
   celebGameList = new ArrayList<Celebrity>();
   gameWindow.replaceScreen("START");
}
```

In order to provide the validation of the user input from the start screen, the following code needs to be added to the `CelebrityGame` class for the validate methods `validateCelebrity` and `validateClue`.

Note that both methods are `public boolean` methods so that the results from their calls can be used in external classes. Both methods will need to be enhanced in Activity 4 to support the subclasses of `Celebrity` that you'll be implementing.

Both methods should use the `trim` method on the supplied `String` parameter to remove any extraneous spaces that may be added to provide a better user experience. The logic of what makes a valid clue or answer is the responsibility of the `Game` class, not the GUI. A requirement has been introduced that a `Celebrity` has a name of at least 4 characters so someone like Cher, Bono, P!nk, and Iman will be recognized as valid.

```
/**
 * Validates the name of the celebrity. The name of the celebrity must have at least
 * 4 characters to be considered valid.
 * @param name The name of the Celebriity
 * @return true if the supplied Celebrity is valid, false otherwise.
 */
public boolean validateCelebrity(String name)
{
   /* To be implemented */
}
```

6. Find the `validateCelebrity` method in the `CelebrityGame` class. In the area specified "To be implemented," implement the process described above to validate the name (or answer) of a celebrity. Note that the method provided currently returns `false` so that the class compiles. To help the player have a better chance of guessing who the celebrity is, at least 10 characters in the clue are necessary, which requires a more detailed clue to be provided.

```
/**
 * Checks that the supplied clue has at least 10 characters and/or
 * is a series of clues.
 * This method would be expanded based on your subclass of Celebrity.
 * @param clue The text of the clue(s)
 * @param type Supports a subclass of Celebrity (LiteratureCelebrity)
 * @return If the clue is valid.
 */
public boolean validateClue(String clue, String type)
{
   /* To be implemented */
}
```

7. Find the `validateClue` method in the `CelebrityGame` class. In the area specified "To be implemented," implement the process described above to validate the clue of a celebrity. Note that the method provided currently returns `false` so that the class compiles.

Once a user is finished validating the name and clue for a celebrity, an instance of the `Celebrity` class needs to be added to the list of celebrities.

The first two parameters are used to create all `Celebrity` objects, and the third `String` parameter is used to identify which subclass of celebrity is in use so that the correct constructor can be called. The third parameter will be used when completing Activity 4 but can be ignored for now.

```
public void addCelebrity(String name, String guess, String type)
{
   /* To be implemented */
}
```

8. Find the `addCelebrity` method in the `CelebrityGame` class. In the area specified "To be implemented," implement the process described above to create a new `Celebrity` object and add it to the list of celebrities.

Once all user input has been collected and the list of `Celebrity` objects has been built, it's time to start the game. The program will ensure that all values are present before switching to the game screen. The `StartPanel` class verifies that the `validateCelebrity` and `validateClue` methods return `true` before enabling the start game button.

The play method tells the `CelebrityFrame` to switch screens allowing play to start for the Celebrity game with the celebrities and associated clues that were just input.

```
/**
 * Ensures that the list is initialized and contains at least one Celebrity.
 * Sets the current celebrity as the first item in the list. Opens the game
 * play screen.
 */
public void play()
{
   if (celebGameList != null && celebGameList.size() > 0)
   {
      this.gameCelebrity = celebGameList.get(0);
      gameWindow.replaceScreen("GAME");
   }
}
```

Since the GUI depends on user interaction, there is no continuous loop in the `play` method. Instead, the methods are called based on the events that occur in the GUI.

Execute the `CelebrityRunner` class. You should see a GUI window displayed that allows you to enter a celebrity name and clue. You can enter as many of these as desired, but once at least one has been entered the start button should be enabled. You will see the code you place in methods activated as you test the game functionality.

Complete the Game Play Methods

9. Complete the implementation for `getCelebrityGameSize`. This method will be used to help populate a label in the game as well as determine when the game is over. The game size is directly linked to the number of `Celebrity` instances in the `celebGameList`.

10. Complete the `processGuess` method. This is the main action that is called when the user interacts with the game. As inferred by the method name it represents the interaction of processing a guess with the guess assigned as the parameter. The provided implementation needs to be replaced with an algorithm that will interact with the `Celebrity` instance and send the result back to the GUI for the game to be played. As usual, data passed in as a parameter must not be destroyed, however it's possible to ignore extraneous spaces at the front and back end of the submission by calling the `trim` method on the parameter value. It's also important to ignore whether the capitalization is used in the guess, so the `equalsIgnoreCase` method in the `String` class will be used when comparing against the `gameCelebrity.getAnswer` call as opposed to just using the `equals` method.

If the guess matches the name of the `gameCelebrity`, the current celebrity must be removed from the list and the next `Celebrity` set as the `gameCelebrity` if there are still `Celebrity` items in the list. If there are no more `Celebrity` items in the list, set `gameCelebrity` to a new `Celebrity` with an empty string for name and clue. The variable being returned from the method also needs to be updated. Make sure to test functionality of the GUI after completing implementation of this method.

```
public boolean processGuess(String guess)
```

Before updating the `sendClue` method, try running the application again and see how the game operates with the update to `processGuess`.

11. Complete the `sendClue` method so that it will return the clue for the current `Celebrity`. Make sure to test functionality of the GUI after completing implementation of this method.

Check Your Understanding

Answer and discuss the following questions:

12. What class handles interaction with the `Celebrity` objects?

13. Outside of the class identified above, what knowledge does the rest of the GUI have about the `Celebrity` class?