Name: _____

Date: _____

ACTIVITY 1

Introduction to Celebrity

In this activity you'll be introduced to and play the game of Celebrity. Then you'll brainstorm different design options for creating a computer version of the game, **including the `Game` class which contains a `play` method**.

- Your teacher will divide the class into groups.
- On the given sheets of paper, write down the names of five celebrities.
- Fold each paper and put the folded papers into your group's container along with the papers of all group members.
- Within your group, break into teams of two people, so a group of six students will consist of three teams of two students each. These are the teams you'll be competing against.
- Play a round of Celebrity: On each team one person is the designated reader and the other(s) the guesser. Team 1 will go first, they have one minute. The reader picks a paper at random and can say anything but the celebrity's name to get the guesser to say the name of the celebrity. If the guesser gets the name right a point is earned, and another paper is selected. If the reader reveals any part of the celebrity's name or passes, a point is lost, and another paper is selected. This continues until the minute is up. The papers are not added back to the pool.
- At this point it's another team's turn, and the process is repeated with a new reader and guesser. Each team takes turns this way until all of the papers have been used. The team with the highest total number of correct guesses wins.

Once you are done:

1. Assuming you were creating your own program from scratch to play Celebrity, brainstorm which classes might be used. Make sure to include a class that keeps track of overall game information.

2. Remember instance variables in a class represent information associated with an object (think nouns). Based on your experience playing the game of Celebrity, list what information might be needed in the `Game` class. This list will be improved upon in a later activity.

3. Write up a list of behaviors that might be needed for the `Game` class. What are the things the `Game` class must do? An example behavior would be to play a game, or update a team's score when the guesser correctly names the celebrity.

4. Looking at question 3, which behaviors might you make into methods? Why might you make that behavior a method?

5. Assuming we would like to use a `play` method to organize and call other methods in the `Game` class, describe the `play` method. Because it has not yet been implemented, use pseudocode, a list, or whatever best outlines what the `play` method described above should do.

Check Your Understanding

6. Pick a real-world object and identify the information and behaviors associated with it. Knowing what you do about primitive data, would you consider any of the information needed for the object to also be an object?

7. Share your real-world object with a partner and provide suggestions on the information and behaviors you have chosen. Make adjustments to your own object based on feedback from your partner.

Name: _____

Date: _____

ACTIVITY 2

The Celebrity Class: A Simple Version

1. What is the purpose of a constructor in a class?

2. Describe what you know about the heading of a constructor.

3. What instance variables need to exist in the `Celebrity` class?

4. Given the `play` method that was designed for the `Game` class in the last activity, what methods should exist in the `Celebrity` class?

5. Based on your answers to the above questions, complete the `Celebrity` class.

6. Write code to test your `Celebrity` class.

Check Your Understanding

7. If your `Celebrity` class has more than one constructor, explain the difference between the constructors that you wrote. If you only wrote a single constructor, provide the code for an additional constructor that could be included in your class, and discuss how it differs from the constructor that you wrote.

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?

Name: _____

Date: _____

ACTIVITY 4

Extending the Celebrity Class

At this point the Celebrity game is functional with a generic `Celebrity` class. While this is a perfectly valid implementation, it's also somewhat limited. Every celebrity that is created has the same attributes. But what if you wanted to create a more specific type of `Celebrity`? One that had additional attributes and behaviors apart from what all `Celebrity` objects have? By using inheritance to extend the `Celebrity` class, you can do just that.

With a partner, think about a subset or certain classification of celebrities that would have additional attributes or behaviors. If you were to incorporate a different type of celebrity into the Celebrity game, what kind of celebrity would you want to make?

1. Identify the attributes and behaviors that belong to this new type of celebrity that are separate from a "default" celebrity.

Class Name:	
Attributes	Type
Behaviors	Return type

It's important to spend time on the design before implementation begins. This step cannot be overstated, as the use of inheritance in a program signifies a relationship between objects and has the goal of code reuse and sharing information.

Compare

Look at the supplied `LiteratureCelebrity` class. It has an `ArrayList<String>` for the clues that are associated with it. It also has a constructor with two parameters just like the `Celebrity` class. There is a method `processClues` that is called in

the constructor and the overridden `getClue` method that maintains the integrity of the original data (clue). There is another overridden method `toString` that uses the `super.getClue` method to access the original data from the `Celebrity` class `clue` variable as well. This is one of the benefits of using inheritance in a design, since superclass methods can easily be called and used in the subclass. All subclasses need to have a clear relationship to the superclass and should provide additional, separate functionality from the superclass to differentiate it from the superclass.

The `@Override` prefix is provided as a cue to other developers to formally express that you as a developer are changing what is done in this subclass. It's not a requirement in Java but is an accepted practice to provide good documentation.

Implement

In this activity, you will create the `Celebrity` subclass you designed earlier.

2. Create the Java file for your `Celebrity` subclass.

3. Define the instance variables for the `Celebrity` subclass.

4. Write the constructor for your class.

5. Override the `getClue` and/or the `getAnswer` method(s).

Tip

When overriding methods in a subclass, method signatures must be the same. This includes the number, type, and order of any parameters of the overridden method.

6. Override the `toString` method.

7. Write any other methods that your design has indicated will be required. Look over your design plan and make sure that you have implemented all required components. Check that your code compiles, making sure to test any methods you write.

Tip

Methods that exist in a superclass can be accessed within the subclass or with subclass objects. When a method is called on a subclass object, the method that is executed is determined during runtime and if the subclass does not contain the called method, the superclass method will automatically be executed. From within the subclass, if you would like to call the superclass version of an overridden method the keyword `super` must be used before the method call.

`CelebrityGame` Update

`processGuess(String)`

Looking at the method `processGuess` you can see that it only needs to interact with the `String` matching the name of the celebrity and that method `getAnswer` must have the same signature (without parameters) in all subclasses of `Celebrity`. This method is shown to illustrate that no code needs to be changed in order for it to work with the new subclass(es).

```
/**
 * Determines if the supplied guess is correct.
 *
 * @param guess - The supplied String
 * @return Whether it matches regardless of case or extraneous external
 *         spaces.
 */
public boolean processGuess(String guess)
{
   boolean matches = false;
   /*
    * Why use the .trim() method on the supplied String parameter? What
    * would need to be done to support a score?
    */
   if (guess.trim().equalsIgnoreCase(gameCelebrity.getAnswer()))
   {
      matches = true;
      celebGameList.remove(0);
      if (celebGameList.size() > 0)
      {
         gameCelebrity = celebGameList.get(0);
      }
   }
   return matches;
}
```

8. Modify the `addCelebrity` method. As you see, the parameter `type` is used to tell the `CelebrityGame` which kind of `Celebrity` to make. This `String` will be sent from the GUI to the game so that the appropriate subclass constructor can be called. The provided code shows how the method was updated to work with the `LiteratureCelebrity` subclass as an example. Add another conditional branch using `else if` to handle your new subclass.

Since all instances of any subclass of `Celebrity` are a `Celebrity`, each can be added to the `ArrayList<Celebrity>` `celebGameList` by polymorphism. No changes are needed to that part of the program.

Tip

In addition to parameters and local variables declared in a method, a method always has access to any instance variables that are declared within the enclosing class. These instance variables are often required to complete the goal of the method. If any of the available variables are objects (reference data), then those objects may have their own methods, variables, and constants that would be accessible within the given method as well.

```
/**
 * Adds a Celebrity of specified type to the game list
 *
 * @param name - The name of the celebrity
 * @param guess
 *            The clue(s) for the celebrity
 * @param type
 *            What type of celebrity
 */
public void addCelebrity(String name, String guess, String type)
{
   /*
    * How would you add other subclasses to this CelebrityGame?
    */
   Celebrity currentCelebrity;
   if (type.equals("Literature"))
   {
      currentCelebrity = new LiteratureCelebrity(name, guess);
   }
   else     //Add an else if here
   {
      currentCelebrity = new Celebrity(name, guess);
   }
   this.celebGameList.add(currentCelebrity);
}
```

9. The `validateClue` and `validateCelebrity` methods may need to be updated for the subclass to check that the supplied `String` values match the requirements for the subclass. The provided code shows how the `validateClue` method was updated to work with the `LiteratureCelebrity` subclass as an example. Add another conditional branch using `else if` to handle your new subclass.

```
public boolean validateClue(String clue, String type)
{
  boolean validClue = false;
  if (clue.trim().length() >= 10)
  {
   validClue = true;
   if (type.equalsIgnoreCase("literature"))
   {
    String[] temp = clue.split(",");
      if (temp.length > 1)
    {
     validClue = true;
     }
    else
    {
     validClue = false;
    }
    }
   //You will need to add an else if condition here for your subclass
  }
  return validClue;
}
```

GUI Update

You'll need to add to the GUI class to support the new addition to the project. The framework for the project has been set so that **you only need to make changes to the `StartPanel` class**. Don't start this until you have completed your new `Celebrity` subclass.

Again, the `CelebrityFrame`, `StartPanel`, and `CelebrityPanel` classes have no knowledge of the `Celebrity` class hierarchy. They only understand `String` and `boolean` values (`JTextField` and `JRadioButton`).

The `StartPanel` class needs to be updated to allow for the selection and creation of multiple types of celebrities. The following instructions assume that all input can be parsed from the `clue` and `name` instance variables. If you want to add more instance variables then you'll need to define, initialize, and `setVisible` additional GUI components based on the selection of the associated `JRadioButton`. This is to make the GUI as easy to use with the least amount of configuration. **All of the following code should be added to the `StartPanel` class**.

10. Add `private String` and `JRadioButton` variables to the `StartPanel` class to match the custom subclass in the instance variable section. Throughout the instructions the placeholders `yourRadioButtonName` and `nameOfCelebrity` are used, however you should use more meaningful variable names for both of these when completing your implementation.

```
private JRadioButton yourRadioButtonName;
```

```
private String nameOfCelebrity;
```

11. Initialize the `yourRadioButtonName` and `nameOfCelebrity` variables in the constructor after the call to `super`. As a practice, it's often a good idea to put variables of the same type together. This makes it easier to identify when other components are added.

```
yourRadioButtonName = new JRadioButton("Your Celebrity Type");
```

```
nameOfCelebrity = "Your celebrity type clue format hint";
```

12. In the `setupPanel` method add the following lines. Remember to change `yourRadioButtonName` to match your variable name. This method is used to add the component to the `JPanel` and to be part of the `RadioButton` group.

```
this.add(yourRadioButtonName);
```

```
typeGroup.add(yourRadioButtonName);
```

13. Change the `setupLayout` method. This method is used to arrange all the components into the panel at the specified locations. The values are used to show the relationship between the different components. To help identify specific locations, look for the bolded words. Change the line

```
panelLayout.putConstraint(SpringLayout.NORTH, literatureRadio,
   10, SpringLayout.SOUTH, celebrityRadio);
```

to be

```
panelLayout.putConstraint(SpringLayout.NORTH, literatureRadio,
   10, SpringLayout.SOUTH, yourRadioButtonName);
```

and add the following lines below the line you just changed.

```
panelLayout.putConstraint(SpringLayout.WEST,
   yourRadioButtonName, 0, SpringLayout.WEST, celebrityRadio);
```

```
panelLayout.putConstraint(SpringLayout.NORTH,
   yourRadioButtonName, 10, SpringLayout.SOUTH, celebrityRadio);
```

14. The `setupListeners` method is the method that links all the GUI components that allow user input to the `ActionListeners` to be processed by the program. In the radio button section of the `setupListeners` method, add this line:

```
yourRadioButtonName.addActionListener(select ->
   clueLabel.setText(yourCelebrityClue));
```

15. In the validate method:

- add an else-if after the literature section based on the `yourRadioButtonName.isSelected` result
- Call the `validateClue` method in the `Celebrity` class, passing the associated values

16. In the `addToGame` method, add an else-if `yourRadioButtonName.isSelected` to provide the correct string to the type variable that is used in the `controller.addCelebrity(answer, clue, type)` call.

At this point you should execute the `CelebrityRunner` class to verify game functionality.

Check Your Understanding

17. How do we identify if a method is an overridden method?

18. How do we send information from the subclass to the superclass?

19. What keyword is used in Java to identify inheritance?

20. What method is executed when an `ArrayList` is made of the superclass but a subclass instance is stored in it?

E.g.

```
ArrayList<Animal> zooList = new ArrayList<Animal>();
zooList.add(new Gryphon());
Animal temp = zoolist.get((int) Math.random() * zooList.size());
temp.playSound();
```

Name: _____

Date: _____

ACTIVITY 5

Open-Ended Activity

As a class, spend a few minutes reviewing the requirements of the open-ended activity.

Requirements:

- Create a superclass
- Create at least one subclass of the superclass
- Override at least one method from the superclass
- Create a program with a `main` method
- Utilize polymorphism

In addition, review the provided scoring guidelines so that you understand what you'll be expected to explain once you're done completing your program.

It's strongly recommended that the implementation of the program involve collaboration with another student. Your selected program can be anything that you choose that meets the requirement and allows you to demonstrate your understanding.

Before beginning, make sure that you understand the expectations for the activity.

- Who will you be working with? Are you allowed to work with a partner? In a group of three or four?
- Among the members of your group (or with your partner), how will the implementation will be completed?
- If you'll be using pair programming, will your teacher be instructing you when to switch driver and navigator, or is this something that you need to keep track of?
- What should you do if your group/pair is stuck? Does your teacher want you to come straight to them? Are you allowed to ask another group?

Check Your Understanding

Once your program has been implemented and tested, answer the following questions on your own:

1. Why did you choose to implement this program?

2. Describe the development process used in the completion of the project.

3. Describe another class that could be designed as a subclass to the superclass you created. Describe additional attributes and behaviors for this new class and explain how this subclass would be useful.

4. Create the inheritance UML diagram for the classes you created.

5. Copy and paste one code segment that uses polymorphism. Other than specific syntax, describe how implementing this program without inheritance would change the complexity of your program, using your copied code segment as an example.