Assignment 5
CS 3443 Application Programming
Due: Sunday, November 30

Whack-A-Mole

Whack-A-Mole is an arcade game where players use a mallet to whack moles as they pop-up from their homes. At any moment during the game 0 or more moles may have popped-up from their homes and can be whacked. The game lasts a fixed amount of time and a player's score depends on how quickly he or she can whack each mole. Moles move independently of each other. Each mole stays in its home for a random amount of time, pops-up and stays up for a random amount of time, and then return to its home; this cycle repeats for the duration of the game.

A Whack-A-Mole application in JavaFX simulates the physical arcade game. The application has five ImageViews in which moles can appear, a time-remaining Label to indicate the number of seconds remaining in the game, a score label that displays the player's current score, and a Button to start the game. See Figure 1.

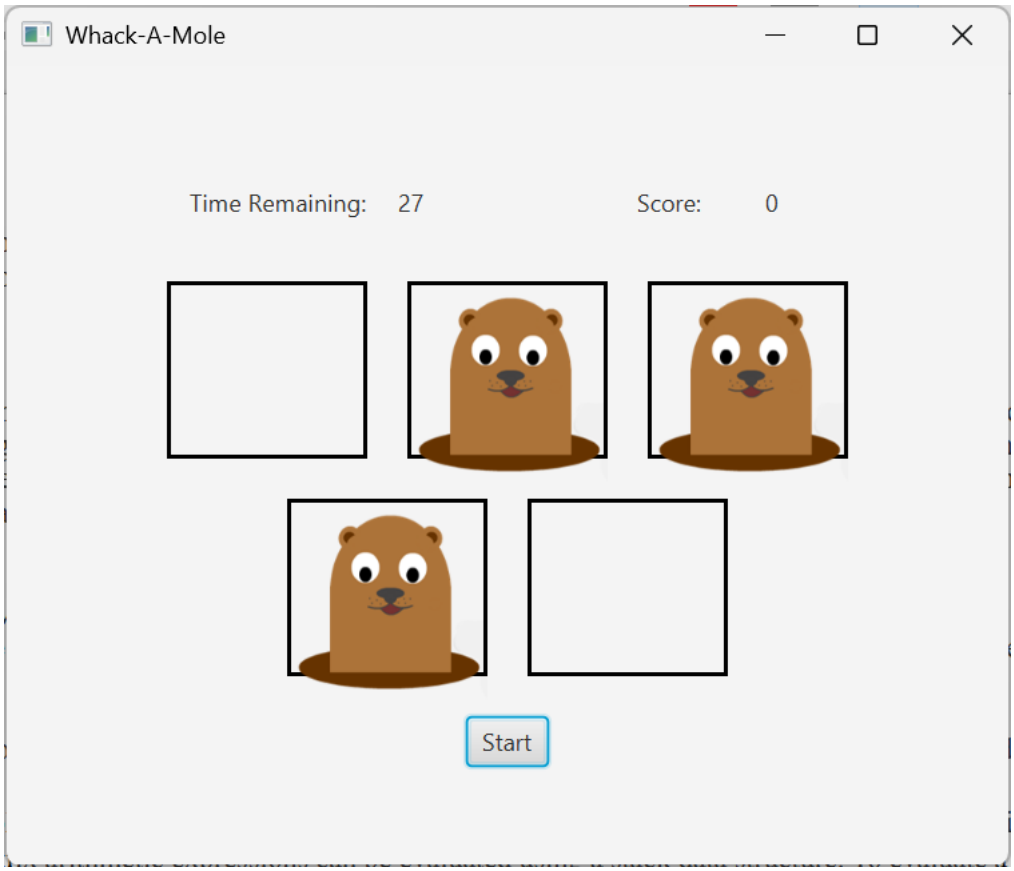

Figure 1. A Whack-A-Mole application

Programming the Whack-A-Mole Game

Create a JavaFX project named "a5_abc123" where abc123 is your UTSA ID. The project should use Java version SE 21.

Use the attached WhackAMole.fxml file. Open it in SceneBuilder to examine the fx:ids of the Labels, Panes and ImageViews and to examine the On Action properties of the ImageViews and Button. Do not change the file.

Also included with this assignment is mole.png, an image file of a mole.

Use the Model-View-Controller design pattern and create three subpackages named `application.model`, `application.view`, and `application.controller`.

Create a UML class diagram for your program indicating which subpackages each class belongs to.

The following is a suggested way to design the application. You may add or remove methods from the classes as well as adding or removing classes. Figure 2 at the end of this document contains a class diagram for this suggested design.

Random Class

Use the Java Random class to generate random numbers in order to determine how long each mole should be hidden and exposed. These durations should be newly calculated for each hidden/exposed cycle of each mole. Use only a single instance of the Random class which can be distributed to each mole.

Model

The Whack-A-Mole model consists of three classes: `CountDownTimer`, `Mole`, and `WhackAMole`.

- The `CountDownTimer` class implements the `Runnable` interface and is responsible for keeping track of the game-time. The `CountDownTimer` runs in its own thread for the duration of a game (30 seconds) and is started by the `WhackAMole` class.
    - The `run` method waits for 1 second (1000 milliseconds) at a time and updates the time-remaining label. After 30 seconds, the `endGame` method of the `Whack-A-Mole` class is called.

- Each instance of the `Mole` class (which implements the `Runnable` interface) is responsible for a single mole and runs in its own thread. Each `Mole` instance is assigned an index number so that its matching `ImageView` can be easily identified.
    - The main loop of the `run` method runs until the game is over. The body of the loop consists of two main sections. In the first section, the mole is hidden and waiting for a random amount of time between 2000 and 5000 milliseconds. In the second section, the mole is exposed and waiting for a random amount of time between 1000 and 2000 seconds. The waiting in both sections can result in exceptions being thrown due to the game ending. In the second section, the wait might also throw an exception due to the player clicking on a popped-up mole. Since an exception can be thrown for two reasons, the `WhackAMole` class provides a boolean `endGame` method that returns true if the game has ended and false otherwise. When an exposed mole is whacked by clicking on it, the `run` method must compute the time that passed from the moment that the mole was exposed until it was clicked on. This elapsed time is passed to the `Whack-AMole` class' `updateScore` method which determines the points for it and updates the player's total score.

- The `WhackAMole` class starts threads for the timer and moles and supports thread interaction such as notifying individual mole threads when the game is over and maintaining the total score.
    - The `startGame` method creates and starts threads for the `CountDownTimer` and each `Mole` instance (only if the current game is over).

- The `endGame` method is called by the `CountDownTimer` when the game is over. This method then interrupts each of the `Mole` threads.
- The `gameOver` method is called by each `Mole` thread when it is interrupted in order to determine the cause of the interrupt.
- The `updateScore` method is called by `Mole` instances when a player has clicked on an `exposed` mole. Its only argument is the number of milliseconds that elapsed between the mole being initially exposed and the player clicking on it. If this elapsed time is less than 500 milliseconds, then the player is awarded 100 points; if it is equal to or more than 500 milliseconds but less than 1000 milliseconds then the player is awarded 50 points; If it is 1000 or more milliseconds, then the player is awarded 10 points.
- The `setExposed` method is called by each `Mole` to indicate if it is currently exposed (and can be clicked on to get points). Its arguments are the `Mole` instance's index number and a boolean value indicating if the `Mole` is exposed to the player. This information is used by the `whackMole` method to determine if a `Mole` thread should be interrupted when the player clicks on an `ImageView`.
- The `whackMole` method is called by the `MainController` when a player clicks on a mole's `ImageView` in order to whack the mole. This method takes a `Mole`'s index as its only argument. If the `Mole` is currently exposed, then its thread is interrupted. If the `Mole` is not currently exposed, then the method does nothing.

View

Carefully inspect the WhackAMole.fxml file in SceneBuilder. Note that there are four Labels, however two of them do not have fx:ids and are not changed by the application.

- The application view consists of a single class, `MainView` with the following methods.
  - The `displayTimeRemaining` method takes a string as an argument and updates the scene's time-remaining Label.
  - The `displayScore` method takes a string as an argument and updates the scene's score Label.
  - The `displayImage` method takes an index and an `Image` object as arguments and updates an `ImageView` in the scene.

Controller

Carefully inspect the WhackAMole.fxml file in SceneBuilder. Note that each `ImageView` is within its own `Pane`. Each `Pane` is assigned an fx:id so that a border can be placed around it by the `MainController`'s `initialize` method.

- The application controller consists of a single class, `MainController` with the following methods.
  - The `initialize` method adds border to each `Pane` in the scene, creates `WhackAMole` and `MainView` instances, and assigns to each `ImageView` instance an index number stored as user data.
  - The `startButtonAction` method simply calls the `WhackAMole` class' `startGame` method.
  - The `imageViewAction` method retrieves the `ImageView` object's index number stored as its user data and calls the `WhackAMole` class' `whackMole` method.

Programming Style

Use good programming style. Your comments should include a description of each class and of each method in Javadoc. Your implementation should follow the DRY principle.

Testing

It is strongly recommended that you test methods and classes individually as you write them instead of testing only after finishing the entire project. Here is a suggested pathway for testing:

1. Create a simplified version of `CountDownTimer` that prints the time remaining to the console. Test it by creating a thread for it.

2. Create the `MainView` class and test it by creating a simplified `MainController` and use its initialize method to exercise the `MainView`'s functionality

3. Update the `CountDownTimer` to use the `MainView`, change the `MainController`'s initialize method and verify that the `CountDownTimer` correctly uses the `MainView`.

4. Create a simplified version of the `Mole` class and verify that it can cycle through hidden/exposed behavior by having it print to the console.

5. Expand the `Mole` class, create the `WhackAMole` class, and verify that it can be used to start threads for the timer and moles and handle the timer signaling the end of the game and the moles' requests to update the player's score.

6. Expand the `MainController` class so that the entire game can be played.

Deliverables

- Create a UML class diagram of your application.
- Create an Eclipse JavaFX project.
- Implement the Whack-A-Mole game using the Model-View-Controller design pattern.
- Export your complete project to a zip file named "a5_abc123.zip" where abc123 is your UTSA student ID.
- Verify that the zip file contains your work.
- Submit your zip file and UML class diagram on Canvas
- Verify that you have submitted your zip file correctly.

Rubric

1.	Good programming style: Javadoc comments, indentation, names, DRY	10%
2.	UML class diagram	5%
3.	Model-View-Controller pattern used correctly	10%
4.	Timer counts down correctly	15%
5.	Mole hide/exposed cycle	15%
6.	Mole whacks do not affect hidden moles	15%
7.	Mole whacks affect exposed moles and yield correct scoring	15%
8.	No other issues that affect the correct functioning of the application	15%
	Total	100%

No credit is given for submissions that do not compile.

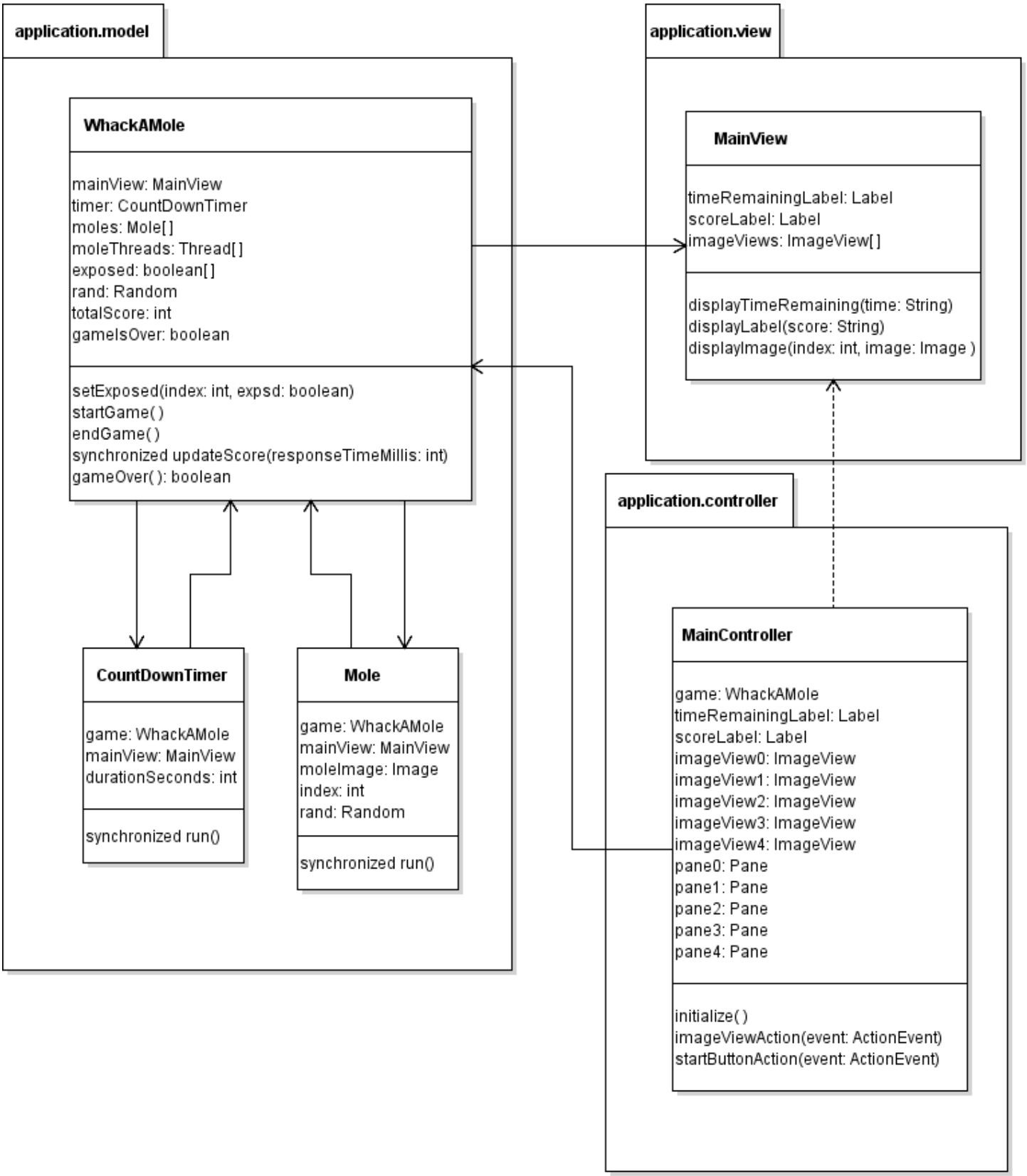

Figure 2. A UML class diagram for the suggested design