# Further Programming COSC 2391/2401
# Semester 1, 2019
# Casino Style Gaming Wheel

## Assignment Part 2: AWT/Swing UI Implementation (25 marks)

This assignment requires you to build an AWT/Swing graphical user interface for the Wheel Game that reuses your code from assignment part 1.

You should build on top of your existing assignment part 1 functionality and should not need to change your existing Game Engine code if it was correct in assignment part 1. You can seek help in the tutelab/consultation to finish assignment part 1 if necessary (since it is a core requirement of this course that you should be able to work with basic OO classes and interfaces). Furthermore, you will require a solid understanding of classes/interfaces to write effective AWT/Swing!

As part of your new implementation you must write a `GameEngineCallbackGUI` class that is added to the `GameEngine` via the existing `addGameEngineCallback()` method. This class will be responsible for managing all of the graphical updates as the game is played. NOTE: this class should not actually implement the UI functionality but instead for cohesion it should call methods on other classes, especially the view. To state another way, it must be the entry point for any UI code resulting from game play, in order to avoid coupling between the `GameEngine` and the UI which is counter to the original specification.

This `GameEngineCallbackGUI` is to be added **in addition to** the console based `GameEngineCallbackImpl` from assignment 1 in order to demonstrate that your `GameEngine` can work with multiple callbacks i.e. both (all) callbacks are called for any relevant operation such as `nextSlot()/result()`. NOTE: This is the main reason the assignment part 1 specification required the `GameEngineImpl` to handle multiple callbacks! It should also help debugging since you will have the console output to match against the UI behaviour.

**AWT/Swing User Interface**

You are to develop an AWT/Swing user interface that implements the following basic functionality:

*Add players (including name and initial betting points balance)*
*Remove Player*
*Place a bet (per player) including specifying the bet type and displaying an error message or dialog for invalid bets*
*Spin the wheel and draw the animation using the supplied .png file (see details below)*
*Display updated player balances after the spin has finished including stating WIN/LOSS per player*

**NOTE**: You should set your delay parameters to (1, 500, 25) ms for testing and submission.

You have some flexibility with designing the layout and appearance of your interface and should focus on clarity and simplicity rather than elaborate design. However, to demonstrate competency you should include at least one each of the following and must also meet the specific requirements in the UI implementation section below.

*A pull-down menu (like the standard File menu in Eclipse)*
*A dialog box*
*A toolbar*
*A status bar*
*A main **WheelPanel** showing the gaming wheel graphic as well as the animated ball (see below)*
*A **SummaryPanel** which is always visible which shows player names, their current points balance, their current bet (including **bet type**) and their most recent win/loss calculated according to the betting odds from assignment 1.*

*A mechanism such as a button for initiating a spin (but the spin should also occur automatically once all players have placed a bet)*

Marking emphasis will be on the quality of your code and your ability to implement the required functionality as well as basic usability of the UI (based on the usability attributes described in the lecture notes).

Your code should be structured using the **Model View Controller** pattern (MVC) where the `GameEngineImpl` from assignment part 1 serves as the model, the listeners represent the controllers as separate classes (each in a separate file placed in the `controller` package), whereas the `GameEngineCallbackImpl` and new `GameEngineCallbackGUI` are part of a `view` package (or sub-packages), as are any additional UI classes such as frames, dialogs, components, menus etc. Furthermore, you must NOT use static referencing (e.g. no use of the Singleton pattern) and therefore all references required by different classes must be passed as parameters (e.g. via constructors).

**IMPORTANT:** Your assignment 2 code should be a **separate Eclipse project** which references assignment 1 via the Eclipse build path dialog. This provides clear separation and helps ensure all of your GUI code (MVC views including the `GameEngineCallbackGUI` and controllers) is separate from your `GameEngineImpl` implementation (MVC model). Furthermore, all of your GUI code should call your `GameEngineImpl` implementation via the `GameEngine` interface only. You should not add any UI code to the `GameEngineImpl` with the primary test being that your UI code should work with any `GameEngine` implementation (for example our own implementation we will use for testing). i.e. your UI should not require anything additional from the assignment 1 `GameEngineImpl` if implemented correctly according to the Javadoc and assignment 1 specification. Finally, your `GameEngineImpl` should still pass the Validator checks from assignment 1. Another way to think about it is that any changes to the original assignment 1 project should be to fix missing functionality, not to add anything new.

**NOTE**: It is a core learning outcome of this assignment to demonstrate that encapsulation and programming to interfaces provides complete separation such that code written by independent parties can work together seamlessly without any change!

## UI Implementation

The animation of the main wheel panel is likely the most challenging part of the assignment and should be left until the end when everything else is implemented correctly. That said it just requires some basic trigonometry and arithmetic (scaling and coordinate translation). i.e. x = r * cos(theta), y = r * sin(theta) and a bit of patience to get it right :)

That said, you can get the main functionality working first using a simple `JLabel` that is updated for each new intermediate result (`nextSlot()`) received from the `GameEngineCallback` method. You can also refer to the console logs.

Additionally, the supplied `Basic_roulette_wheel_1024x1024.png` should be drawn square (i.e. circular not elliptic); scaled to be centred in the ***WheelPanel***; and resized as the frame resizes to take up all the space in its containing panel (a `BorderLayout` is useful here).

To simplify doing the animation you can redraw the background on every update rather than having to save and restore the pixels under the drawn ball although you are welcome to do this at the end if you want a challenge since it is much more efficient :)

The ***SummaryPanel*** should only take up one quarter, to a maximum one third, of the frame with the ***WheelPanel*** taking the remainder of the space (excluding any toolbars, status bars etc. which should be compact). Resizing should be appropriately handled using layout management.

I have provided a video `WheelSpin.mp4` on Canvas using my own sample implementation for reference so you can see the exact required resizing and spinning behaviour of the wheel. NOTE: This video only shows the spin but not include the ***SummaryPanel*** and other functionality described above which must also be implemented.

## Threads

Although threads will be covered towards the end of the semester, AWT/Swing requires all calls to the UI (any methods in AWT/Swing such as creating a component, laying out, or updating) to be done on the AWT Event Dispatch Thread. Furthermore, since the `spin(…)` method of `GameEngineImpl` executes in a loop with a delay it is necessary to run it in a separate thread so it does not lock up the UI.

To achieve this you can use the code below. You don't need to know exactly how this works for now (although the API docs are useful here and we will also cover in class) but hopefully you are able to identify how the code is simply using anonymous inner classes to execute some code on a separate thread!

**To call a GameEngineImpl method (such as spin()) on a separate thread.**

```
new Thread()
{
   @Override
   public void run()
   {
      //call long running GameEngine methods on separate thread
   }
}.start();
```

**To update the GUI from the callback ..**

```
SwingUtilities.invokeLater(new Runnable()
{
   @Override
   public void run()
   {
      // do GUI update on UI thread
   }
});
```

## Implementation Details

**IMPORTANT**: As with assignment one you must not change any of the interfaces but you may implement any other helper classes that you need to. A correct implementation should not require any changes to the `GameEngineImpl`, only the addition of new AWT/Swing classes to build the UI, in addition to a new `GameEngineCallbackGUI`.

To demonstrate cohesion and correct OO techniques, all UI code must be written carefully by hand. You can use builder tools (such as NetBeans) to aid prototyping but all final code must be written by hand since this is a programming not a UI design course :) i.e. You **will lose marks** if you submit generated code! You will also be breaching the "no third-party code" requirement below.

## SUMMARY

1. You should use MVC implementation for your system.
2. You should write all your listeners as separate controller classes in the `controller` package.
3. You must not use any static referencing (e.g. no use of the Singleton pattern permitted).
4. You should aim to provide high cohesion and low coupling as covered in this course.
5. You should aim for maximum encapsulation and information hiding.
6. You should rigorously avoid code duplication.

7. You should comment important sections of your code remembering that clear and readily comprehensible code is preferable to a comment.
8. Since this course is concerned with OO design you should avoid the use of Java 8+ lambdas which are a functional programming construct.
9. You should use extra threads where necessary (based on the sample code provided above) to ensure smooth UI operation.

## Submission Instructions

- You are free to refer to textbooks and notes, and discuss design issues (and associated general solutions) with your fellow students on Canvas; however, the assignment should be your **own** individual work and **IS NOT** a group assignment.
- You may also use other references, but since you will only be assessed on your own work you should **NOT** use any third-party packages or code, or any generated code (e.g. UI builders) (i.e. not written by you) in your work.

The source code for this assignment (i.e. complete compiled **Eclipse projects**[1] **i.e BOTH assignment 1 and 2 projects**) should be submitted as a SINGLE .zip file by the due date using the Eclipse option `export->general->archive`.

**IMPORTANT:** SUBMISSIONS OF A ROULETTE OR ANY OTHER GAME WHICH DO NOT IN ANY WAY ADHERE TO THE SPECIFICATION AND PROVIDED CODE WILL RECEIVE A **ZERO MARK**

***Due 6:00PM Fri. 31st May 2019 (25%)***

***Late submissions are handled as per usual RMIT regulations - 10% deduction (2.5 marks) per day. You are only allowed to have 5 late days maximum.***

[1] You can develop your system using any IDE but will have to create an Eclipse project using your source code files for submission purposes.