




# Further Programming COSC 2391/2401, S2, 2020

## Casino Style Card Game

### Assignment Part 2: AWT/Swing UI Implementation

	<b>Assessment Type:</b> Individual assignment; no group work. Submit online via Canvas→Assignments→Assignment 2. Marks are awarded for meeting requirements as closely as possible according to section 2 and the supplied rubric. Clarifications/updates may be made via announcements/relevant discussion forums.
	<b>Due date:</b> Due 6:30PM Fri. 16th Oct. 2020. Late submissions are handled as per usual RMIT regulations - 10% deduction (4 marks) per day (Saturday and Sunday count as separate days). You are only allowed to have 5 late days maximum unless special consideration has been granted.
	<b>Weighting:</b> 40 marks (40% of your final semester grade)

#### 1. Overview

This assignment requires you to build an AWT/Swing graphical user interface for the Card 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!

#### 2. Assessment Criteria

This assessment will determine your ability to implement Object-Oriented Java code according to a written and visual specification and incorporating specific design patterns. In addition to functional correctness (i.e. getting your code to work) you will also be assessed on code quality. Specifically:

- You should use MVC implementation for your system.
- You should write all your listeners as separate controller classes in the controller package.
- You must not use any static referencing (e.g. no use of the Singleton pattern permitted).
- You should aim to provide high cohesion and low coupling as covered in this course.
- You should aim for maximum encapsulation and information hiding.
- You should rigorously avoid code duplication.
- You should comment important sections of your code remembering that clear and readily comprehensible code is preferable to a comment.
- Since this course is concerned with OO design you should avoid the use of Java 8+ lambdas which are a functional programming construct.
- You should emphasize basic usability of the UI (based on the usability attributes described in the lecture notes).
- You should use extra threads where necessary (based on the sample code provided in this document) to ensure smooth UI operation that is correct according to the Java API specification.

#### 3. Learning Outcomes

This assessment is relevant to the following Learning Outcomes:

CLO1: **Explain** the purpose of OO design and **apply** the following OO concepts in Java code: inheritance, polymorphism, abstract classes, interfaces and generics.

CLO2: **Describe and Document Diagrammatically** the OO design of the Java Collection Framework (JCF) and **apply** this framework in Java code.

CLO3: **Describe and Document Diagrammatically** the OO design of the Java AWT/Swing APIs and **apply** these APIs to create graphical user interface (GUI) code.

CLO4: **Demonstrate Proficiency** using an integrated development environment such as Eclipse for project management, coding and debugging.

CLO5: **Describe and Document Diagrammatically** common OO design patterns such as Model View Controller (MVC), Observer, Decorator and **apply** in Java code.

#### 4. Assessment details

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 `nextCard()/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) displaying an error message or dialog for invalid data*

*Remove Player*

*Place a bet (per player) including displaying an error message or dialog for invalid bets*

*Cancel/Remove a bet (i.e. reset a bet to 0)*

*Players Deal (per player) .. cards are shown in real-time in the **CardPanel** as they are dealt (see **drawing the cards** below) with appropriate delay timing*

*House Deals - Once all players have bet and dealt the House automatically deals and again their cards are displayed in real time in the **CardPanel***

*Display updated player balances in the **SummaryPanel** after the house deal has finished including stating WIN/LOSS per player and then reset all bets to 0 for the next round*

**NOTE:** You should set your delay parameters to 100 ms for testing and submission.

#### VISUAL LAYOUT

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 switchable per player **CardPanel** showing the card graphics as well as the animated and correctly timed deal (see UI Implementation/Limitations below for more detail)*

*A **SummaryPanel** which is always visible which shows player names, their current points balance, their current bet and their most recent win/loss amount.*

*A mechanism such as a button for initiating a player deal (but the house deal should occur automatically once all players have placed a bet and dealt)*

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** called `CardGameGUI` 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 OO interfaces provides complete separation such that code written by independent parties can work together seamlessly without any change!

### UI IMPLEMENTATION/LIMITATIONS

You must only **display** the **CardPanel** for a single player at a time, whereby the visible/active player can be directly selected from a list or combo box (for example a `JComboBox` in the toolbar).

You should also implement the **house** as a player in the UI (but they will not be a player in the `GameEngine` since this is not how it was designed). Furthermore, the *house player* is unique in that it cannot bet or deal since dealing is automated as described above. The house can however be selected so you can see the last dealt cards, and also the house player should be automatically selected when the house auto deals.

More importantly, since the UI is per player, if a player is currently visible/selected and dealing, when you switch to another player the UI should stop animating the dealt cards and should instead display the last cards dealt by the player you switched to. You can however assume that only one player will be dealing at a time so you do not have to handle concurrency in the `Game Engine` but you should enforce this in the UI (for example by disabling the deal button during a deal).

The animation, positioning and scaling of the **CardPanel** 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 arithmetic (scaling and coordinate translation) and a bit of patience/determination to get it right :)

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

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

### **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 `dealPlayer()` / `dealHouse()` methods of `GameEngineImpl` executes in a loop with a delay (or using a `Timer`) 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 `dealPlayer()/dealHouse()`) 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.

### DRAWING THE CARDS

I have provided a video `CardDeal_100_delay.mp4` on Canvas using my own sample implementation for reference so you can see the exact card design as well as the required sizing/scaling and positioning of the cards as they are dealt (rather than me providing a long-winded written description!). **Do note** that the cards are a **simplified representation of standard playing cards** to simplify the drawing although personally I think the “minimalist” approach is quite stylish 😊

**NOTE:** This video only shows the betting and dealing but does not include the *SummaryPanel* and other functionality described above which must also be implemented.

To potentially obtain full marks the cards **MUST be drawn manually** in an overridden `paintComponent(Graphics g)` method (plus helper methods as needed) using appropriate `java.awt.Graphics` methods and manually calculated geometry in order to obtain the correct placement and scaling. This behaviour is evident in `CardDeal_100_delay.mp4`. The only part of the card that is drawn using images is the suit and for this purpose I have provided public domain suit images which you can place in the `project img/` folder.

Furthermore, the cards are drawn such that a maximum of six cards can fit width-wise with appropriate padding and the card height is 1.5 times the width regardless of scaling.

I have also provided a full set of card .PNGs which I also generated programmatically using my own solution, however these are **for reference only** since you need to draw your own cards using **graphics and text primitives** as described above. Note however that during testing, or as a backup, you can use these cards to draw, however **you will only receive a MAXIMUM of 50% of the marks for the card drawing functionality** in the rubric if you do not draw your own cards!

Finally, to simplify doing the animation you can redraw the background on every update and redraw all the cards.

## GAME RULES SUMMARY

A player cannot deal if they have not placed a bet

A player cannot bet more than their current points balance

If a player reaches 0 points you must automatically remove them from the game

Another player cannot start a new deal during an ongoing deal but you can switch players to see their last dealt cards

The House automatically deals when all players have bet and dealt

After the House deals and the **SummaryPanel** is updated (including player WIN/LOSS/DRAW), all bets are reset to 0 for the next round

## 5. Referencing and third party code exclusion

- You are free to refer to textbooks and notes, and discuss the design issues (and associated general solutions) with your fellow students or 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.

## 6. Submission format

The source code for this assignment (i.e. complete compiled **Eclipse projects**<sup>1</sup> 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`.

You **MUST** include a README.TXT file in the root of the project providing any details about your project and running your submission.

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

## 7. Academic integrity and plagiarism (standard RMIT warning)

**NOTE:** Any discussion of referencing below in the standard RMIT policy is generic and superseded by the third-party code exclusion in section 5.

Your code will be automatically checked for similarity against other students' submission so please make sure your submitted assignment is entirely your own work.

Academic integrity is about honest presentation of your academic work. It means acknowledging the work of others while developing your own insights, knowledge and ideas. You should take extreme care that you have:

- Acknowledged words, data, diagrams, models, frameworks and/or ideas of others you have quoted (i.e. directly copied), summarised, paraphrased, discussed or mentioned in your assessment through the appropriate referencing methods,
- Provided a reference list of the publication details so your reader can locate the source if necessary. This includes material taken from Internet sites.

If you do not acknowledge the sources of your material, you may be accused of plagiarism because you have passed off the work and ideas of another person without appropriate referencing, as if they were your own.

RMIT University treats plagiarism as a very serious offence constituting misconduct. Plagiarism covers a variety of inappropriate behaviours, including:

- Failure to properly document a source
- Copyright material from the internet or databases
- Collusion between students

For further information on our policies and procedures, please refer to the [University website](#).

## 8. Assessment declaration

When you submit work electronically, you agree to the [assessment declaration](#).

---

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