SCC110: Software Development Term 2. Java Programming.

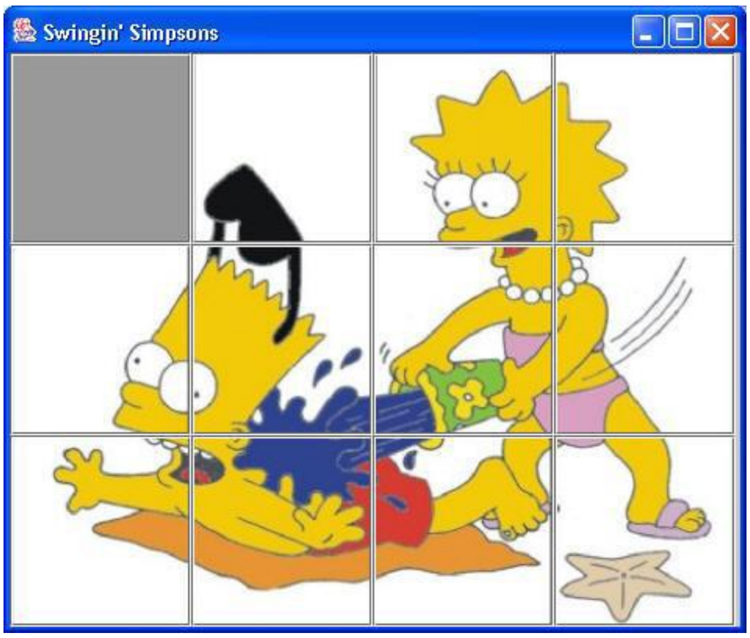
Assessed Exercise: **Sliding Tile Puzzle**
Moodle Submission Deadline: **16:00 Friday Week 20**
Assessment Mode: **IN LAB ASSESSMENT IN YOUR OWN WEEK 20 LAB SESSION**

Aims

This **assessed exercise** is designed to test your understanding of all the OO programming concepts we've seen in the Lectures and Labs, and their application in Java.

Your assignment is to implement a simple, but fully functional Java application - a sliding tile puzzle game, as illustrated below. I'm sure most of you recognise the puzzle - the aim of the game is to allow the player to move tiles left, right, up or down within the puzzle. The challenge being that you can only move tiles into the empty space... Ultimately, a player tries to reform the original image by moving the tiles back, one by one until every tile is in the correct location and puzzle is complete. **If you're not sure how the puzzle should work, ask your tutor.**

The assignment is separated into incremental tasks. The more tasks you complete, the more marks you will receive. However, many marks are available for the quality of your solution - **so remember to make sure you write object oriented code, and follow good Java code style conventions**. We'll also expect you to start using code versioning best practices, so make sure you use git for your work and make regular commits.



Background

We have seen in lectures and labs how to write object oriented programs in Java, and how to write simple Swing applications. This assignment builds on everything you have learned, but you haven't yet seen how to handle images in your applications. This section provides some guidance on this.

Images in Swing applications are represented by a class called **ImageIcon**. The ImageIcon class contains a constructor that lets you create an instance of an ImageIcon from a given filename.

The **JButton** class is able to create buttons showing images as well as text… In particular, the JButton class has a **constructor** that allows a JButton to be created from an ImageIcon, and **accessor** and **mutator** methods for the image currently associated with a JButton.

For example…

```
ImageIcon i = new ImageIcon("thing.jpg");
JButton b = new JButton(i);
```

```
ImageIcon i = new ImageIcon("stuff.jpg");
b.setIcon(i);
```

Task 1: Create a Java class which can display a Sliding Tile puzzle using Swing.

First, you'll need the relevant graphics for your application:

- Download the resources.zip file from Moodle, and unpack it. This file contains the images you can use for this exercise. We recommend you put these files in the same location as your java source files.

Now, you'll need a graphical user interface…

- Create a java class to represent your graphical interface.
- Create a constructor for your class, and in it write code to create a GUI like the one shown above using the Swing APIs. **HINT:** Think carefully about how the tiles could be represented and displayed and which Swing layout manager you want to use…
- Write a main method that launches your graphical user interface.

Task 2: Adding Functionality…

Develop your code so that when a tile is clicked, it will appear to swap places with the blank square. Swapping a tile with a blank tile is nowhere near as difficult as it sounds. Remember that we can't easily move components around the GUI (as Java controls their locations via layout managers), but, we can make it **LOOK LIKE** the tiles have changed place by swapping the images displayed on them. This way the tiles themselves never need to change place…

- Write appropriate java classes and methods to allow tiles to be swapped with the blank tile.

- Write an actionListener to detect when any of your tiles are clicked with the mouse. Write code so that when a tile is clicked, it will swap with the blank tile.

Task 3: Validation…

Notice you can swap any tile with the blank tile, regardless of where that tile is on the board. This is not the proper behaviour for a sliding tile puzzle!

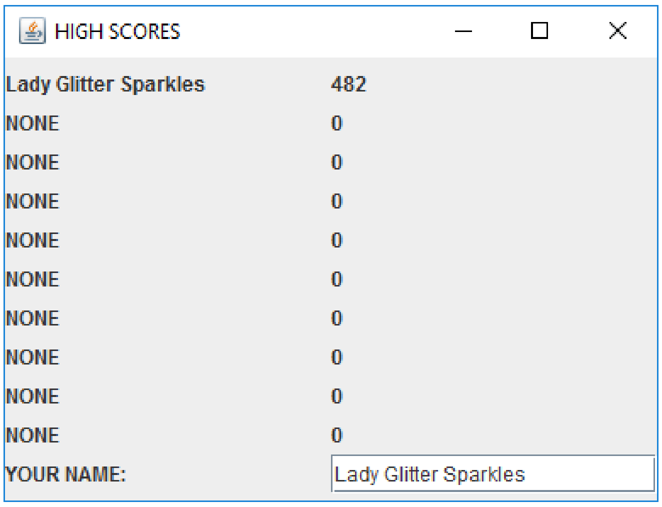
- Update your code so that your it only swaps tiles that are adjacent to the blank square (i.e. directly above/below/left or right). You should not permit diagonal movement.

Task 4: Scoring…

Add a scoring mechanism to you puzzle, such that:

- The current score should be displayed somewhere on your graphical interface.
- Every time a tile is moved, the score should be increased by one.
- You should detect when the puzzle is completed.
- When the game reaches a completed state, the player's score should be compared to a high score table and the table updated accordingly.
- There should be a way to allow players to enter their name.
- The high score table should also be displayed as a Swing graphical interface.

An example high score table might look a little like this:



Lady Glitter Sparkles	482
NONE	0
NONE	0
NONE	0
NONE	0
NONE	0
NONE	0
NONE	0
NONE	0
NONE	0
YOUR NAME:	Lady Glitter Sparkles

Exercise 5: The STAR task…

Add another button to your puzzle game that randomizes the puzzle. **BUT**, you need to ensure that the puzzle you present is always solvable. **THINK CAREFULLY** about a simple algorithm to randomize your puzzle that can guarantee the resulting puzzle is not impossible to solve.

Assessment

This work will be assessed through a practical demonstration and code inspection of your work.

Be prepared to show your program in your Week 20 practical session, and to answer questions about it posed by your markers.

To gain marks from this assignment:

- **You MUST submit your code to Moodle by the advertised deadline.**
- **You MUST demonstrate your work IN YOUR OWN LAB SESSION.**

FAILURE TO ADHERE TO THE ABOVE MARKS WILL RESULT IN A MARK OF ZERO FOR THIS EXERCISE. THE UNIVERSITY'S TYPICAL LATE SUBMISSION PENALTY DOES NOT APPLY TO THIS EXERCISE.

Marking Scheme

Your work will be marked based on the following five categories. Your final grade will be determined based on a weighted mean of these grades according to the weighting shown in the table below.

Functionality	50%
Code Structure and Object Oriented Design	20%
Code Style and Commenting	10%
Usage of Git Version Control	10%
Ability to Answer Questions on Code	10%

In all cases a grade descriptor (A, B, C, D, F) will be used to mark your work in each category. The following sections describe what is expected to attain each of these grades in each category.

Markers can also recommend the award of a distinction (+) category overall if they feel a piece of work exhibits clearly demonstrable good programming practice.

EXAMPLE indicators of grade boundaries. Markers will always use their discretion.

Functionality:
A: Fully working program meeting all requirements of Tasks 1, 2, 3 and 4.
B: Working program meeting all requirements of Tasks 1, 2 and 3.
C: Working program meeting all requirements of Tasks 1 and 2.
D: Working program meeting all requirements of Task 1.
F: No working program demonstrated, or program does not meet requirements of Task 1.

Code Structure and Elegance:
A: Well written, clearly structured code showing student's own examples of good OO practice.
B: Well written, clearly structured code.
C: Clearly identifiable but **occasional** weakness, such as repetitive code that could be removed through use of a loop, poor use of public/private, unnecessary / unused code, inappropriate naming and scoping of variables.
D: Clearly identifiable **systematic** weakness, such as multiple examples of repetitive code that could be removed through use of a loop, systematically poor use of public/private, large sections of unnecessary / unused code, consistently inappropriately named and scoped variables.
F: All the above.

Code Style and Commenting
A: Consistently well indented, well named and well scoped variables with Javadoc commenting.
B: One code block showing poor naming, scope or indentation or occasionally vague and/or inaccurate comments.
C: Two or Three code blocks showing poor naming, scope or indentation, code is partially commented **or** systematically vague and/or inaccurate comments.
D: Four or Five code blocks showing poor naming, scope or indentation or comments.
F: Five or more code blocks showing poor naming, scope or indentation or comments.

Version Control:
A: Project is versioned using a private github repository with a strong and timely commit history
B: Project is versioned using a private github repository with a strong or timely commit history
C: limited use of github, commits are infrequent or weak commit messages
D: limited use of github, commits are infrequent and weak commit messages
F: No use of version control

Ability to Answer Questions on Code:
A: All questions answered in detail.
B: Student failed to explain one technical question about their code.
C: Student failed to explain two technical questions about their code.
D: Student failed to answer questions, but could provide a basic overview of their code.
F: No evidence that the student understands the code being demonstrated.

Star Categories:
By showing a demonstrable example of the star task, your marker **may** choose to award a star (*) grade to your work. If you feel your work shows additional merit beyond the specification, it is your responsibility to make your marker aware of this during your in lab marking session.