Assignment 4: Shapes and Game UI

- Submit deliverables to CourSys: https://courses.cs.sfu.ca/
- Late penalty: 10% per calendar day (each 0 to 24 hour period past due), max 2 days late.
- **This assignment is to be done individually.** Do not show other students your code, do not copy code found online, and do not post questions about the assignment online. Do not use solutions from previous semesters, courses, or offerings. Please direct all questions to the instructor or TA: cmpt-213-help@sfu.ca;
  - You may use *ideas* you find online and from others, but your solution must be your own.
- See the marking guide for details on how each part will be marked.
- Assignment must be done in **IntelliJ**, and submission must include project's .iml file.

1. Shapes

In this part you will implement a system for drawing basic shapes using blocks in a graphical user interface. A `Canvas` class is provided to handle the basic drawing. The shape classes use an inheritance hierarchy to build up functionality without repeating code. The shape classes are shown in Figure 1. The provided code, plus shape classes, are described below.

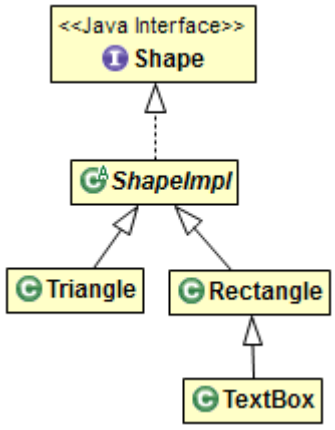

Figure 1: Shape class hierarchy.

1.1 PicturePanel

The following classes provide the foundation of your application:

- `MainGUI`: "Client" code which creates a GUI, plus code to exercise all other classes in the system.
- `PicturePanel`: A high-level class which is instantiated by the client code to hold and draw any number of `Shape` objects.
- `Canvas`: Support class used to draw the boxes. Supports making each box a different colour and showing a character in the box.
- `CanvasIcon`: Support class used to makes a `Canvas` able to be drawn on the screen.

Figure 2 shows a UML class diagram for some of these classes.

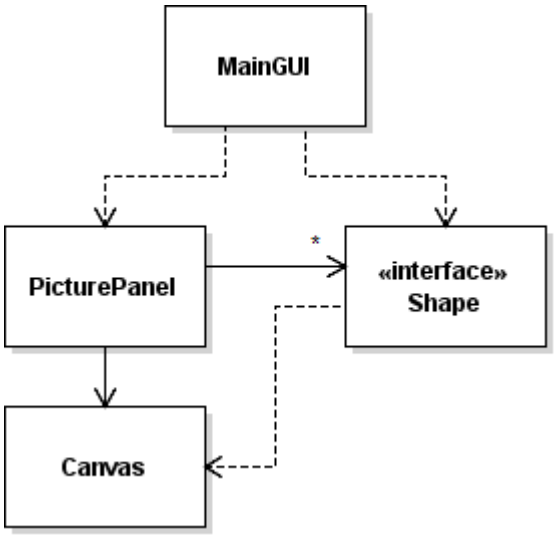

Figure 2: UML class diagram of shapes and client code.

1.1.1 Example Canvas

`MainGUI.makeExampleCanvas()` is an example of how to use the canvas. It demonstrates some of the operations that are supported on the canvas. It fills in a `Canvas` to look like Figure 3.

- Set colour of a cell in the canvas.[1]
- Set the character shown on a single cell of the canvas.
- You may ignore the code related to `JFrame`, `JLabel`, `CanvasIcon`, and `PicturePanel` (they are part of rendering the Canvas into a graphical Swing GUI).

Note that this code only shows how to interact with the `Canvas`; it does not show making actual shapes. See Figure 4 and sample output on course website for the shapes that the provided `MainGUI` code produces once you have implement the required classes.

Other "`make...()`" functions in `MainGUI` have been commented out. Uncomment these as you implement the shapes and `Picture` classes.

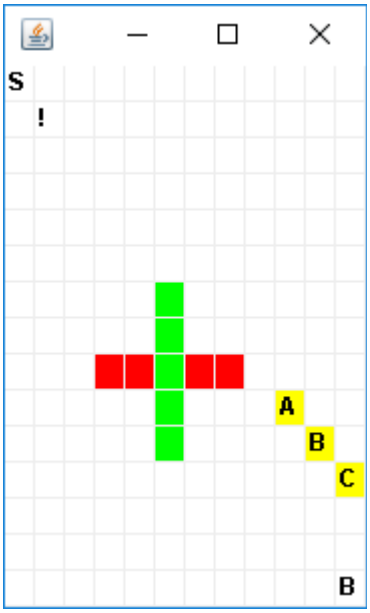

Figure 3: Canvas produced by makeExampleCanvas()

1.2 Required Object Structure

You must:

- Create a `Shape` interface
- Create a `ShapeImpl` class which provides a base-level implementation of shared functionality required by derived classes.
- Create a `Triangle`, `Rectangle`, and `TextBox` class.
  - Each class must work with the interface expected by `MainGUI`.

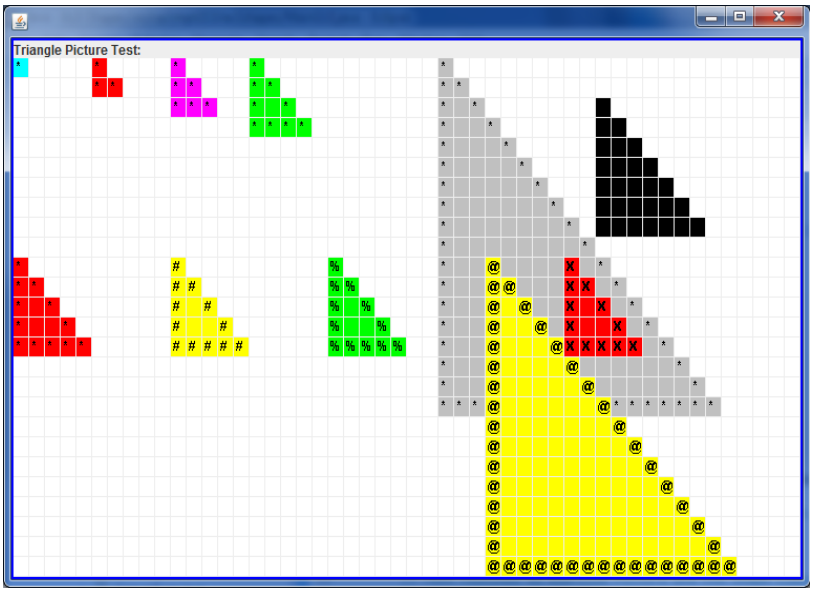

Figure 4: Sample program output for Triangles

1 Yes, colour has a 'u'; however, since Java does not spell it with a 'u', it is best to be consistent in code.

1.3 Suggested OOD

Figure 5 shows a *suggested* OOD for the shapes in this assignment. Each of these are described below. Both the diagram and the notes below are just suggestions; you may choose to do it a different (but at least as good) way.

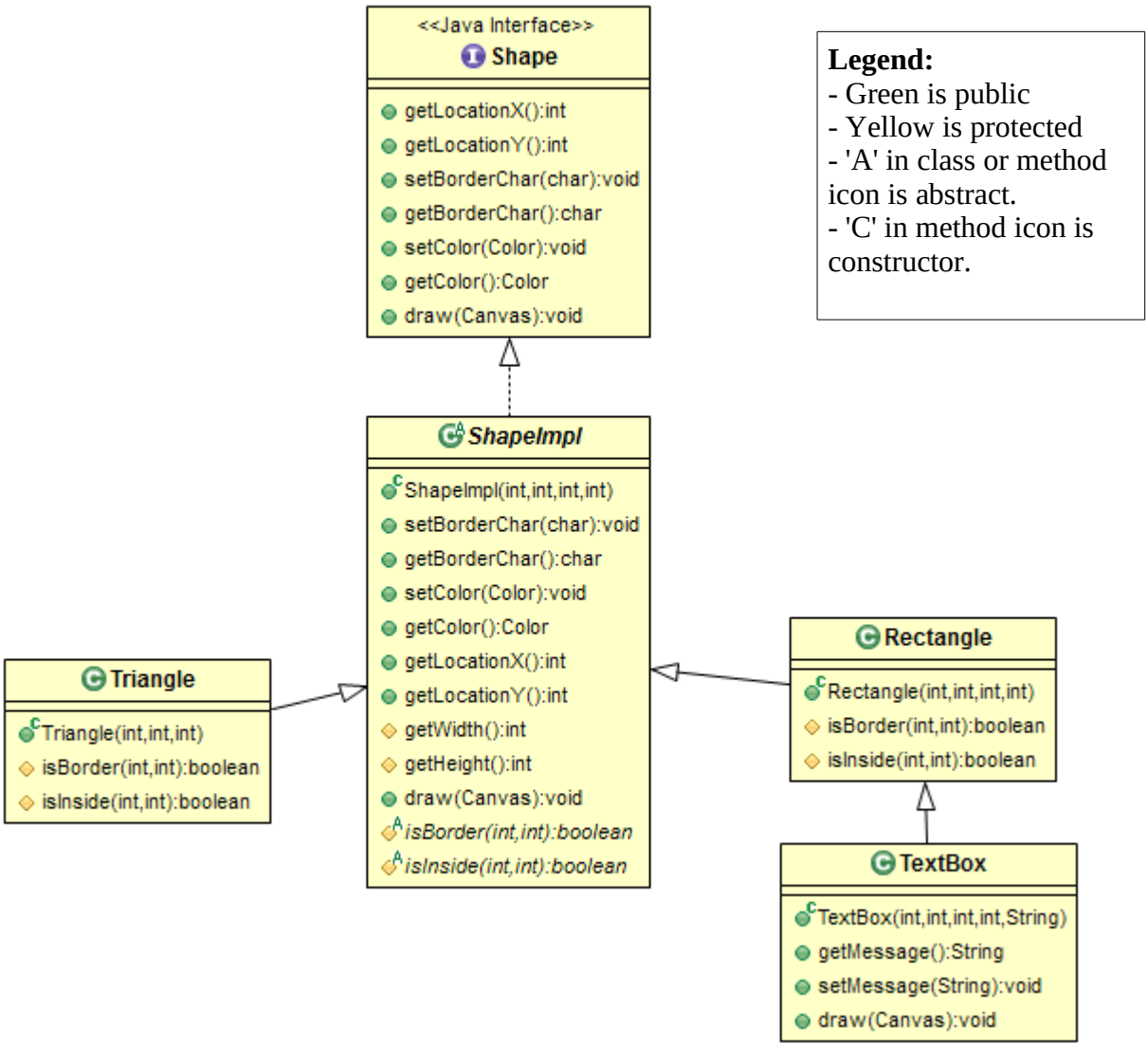

Figure 5: Detailed suggested UML design.

1.3.1 Shape Hints

Create a Java interface named `Shape`, as shown in Figure 5:

- Each shape stores its location as the top-left X and Y coordinates of the shape (integers).
  - The constructor of each class derived from `Shape` should set the coordinates.
- All `Shapes` are drawn with a border (the border character), and have a coloured background.
- The `draw()` function allows any `Shape` object to draw itself onto a passed in `Canvas`.

1.3.2 ShapeImpl Hints

This abstract base class implements much of the functionality specified by the `Shape` interface.

- It stores the location, border character, and colour as private fields.
  - Default character for the border is '*'.
  - Default colour is yellow (`Color.YELLOW`).
- It implements the required getter and setter functions for these fields.
  - If a derived classes needs access to its location, border character, or color it uses accessor functions (encapsulation).
- The constructor accepts the `x` and `y` location, plus the `width` and `height` of the shape.
- The `draw()` function draws the shape into the `Canvas` parameter. However, since this is the abstract base class, it cannot know exactly how to draw the specific shape (such as a triangle). To draw a specific shape, it must "ask" the derived class. Here is this could work:
  - The shape's `width` and `height` are used by `draw()` to scan through all positions which are between `(x,y)` and `(x+width-1, y+height-1)`.
  - For each position, `ShapeImpl`'s `draw()` asks the derived class if that position is:
    - a border cell (drawn with a coloured background and the border character) by calling `isBorder(xPos, yPos)`
    - inside the inner region of the shape (drawn with a coloured background but no border character) by calling `isInside(xPos, yPos)`
    - otherwise it is neither a border nor inside and not drawn at all.
  - Note that this is the "Template Method" design pattern (more in class or online):
    - `draw()` is the "template method".
    - `isBorder()` and `isInside()` are the "primitive operations".
- `isBorder()` and `isInside()` are abstract methods which concrete derived classes override.

1.3.3 Triangle Hints

- `Triangle` draws an equilateral right-triangle where the left and base (bottom) edges are the same size. See sample output.
- `x` and `y` of the `Shape` class are the top left corner of the `Triangle`.
- Constructor takes `x`, `y`, and the triangle's size.

1.3.4 Rectangle Hints

- `Rectangle` stores a width and height.
- `x` and `y` of the `Shape` class are the top left corner of the `Rectangle`.

1.3.5 TextBox Hints

`TextBox` is-a `Rectangle` and adds a message inside it.

- When drawn on a `Canvas`, the `TextBox` places the text inside the box drawn by the `Rectangle`.
  - Use overriding to create a custom draw function, but use base class implementation to help.
  - Do not repeat the drawing code from `Rectangle`! You wrote it once; it works; use it!
- Laying out the text is non-trivial:
  - If the message is too long to fit on one line, it must be split across multiple lines.
    - Strip leading and trailing spaces on each line of text in the `TextBox`.
  - If there is a space on the line, break on the space; otherwise fill the entire line inside the `Rectangle` with text and break mid-word as needed.
  - Each line of text must be centred horizontally inside the `Rectangle`.
    - If there are an odd number of extra spaces on the line, it does not matter if the extra space is before or after the text, or even if it is inconsistent. For example:

Extra space after: `" Hi "`
Extra space before: `" Hi "`

- Text need not be centred vertically; start output on the first line inside the `Rectangle`.
- Text must not overlap the `Rectangle`'s border (drawn by `Rectangle::draw()`).
- If there is more text than will fit inside the rectangle then some text will not be displayed.

1.4 Implementation Suggestions

Suggested implementation order:

1. Create `Shape` interface and `ShapeImpl` class.
2. Create `Rectangle` class.
3. Test using the `makeRectanglesPicture()` function.
4. Move on to `Triangle`, then `TextBox`.

Carefully compare your output to the sample output on the course website. The provided `MainGUI` should generate virtually identical output with your code without any modification, other than commenting/uncommenting the appropriate `make__()` functions.

2. Tic-Tac-Toe Web App

Create a Spring Boot server application which allows the user to play Tic-Tac-Toe.

- The course website has script files with curl commands to interact with the server. Your system must respond to these messages as shown in the file.
- It must use the following REST API end points

HTTP Method & URL	Description
GET /about	Return a string stating your name as the game designer.
GET /games	Return a the list of games.
POST /games	Create a new game (returns 201). Body of request is JSON object for game.
GET /games/5	Return data on game #5.
GET /games/5/moves	Return all the moves in game #5. Each move is assigned by the server a number, indicating the move number in that game.
POST /games/5/moves	Create a new move in game #5 (returns 201). Body of request is JSON object for the new move.
GET /games/5/board	Return the state of the game board. Response object has three fields: row1, row2, and row3. Each is a space-separated string of characters for each cell of the game. Unplayed cells return space as well.

- All data exchanged between the server and client is in JSON format (except for /about which may come out just plain text).
- HTTP status of success GET responses are 200 (OK) and POST are 201 (created).

- Responses from successful posts requests return the object just created.
- Do not save the state of the games between executions of the server (no persistence).

2.1 Error Handling

Your application will be tested with only the following errors:

HTTP Status	Error
400	1. If 'O' tries to go first 2. If the move is neither 'X' nor 'O' 3. If a player tries to go twice in a row. 4. Move is out of bounds (row -1, col 100) 5. Trying to play in a location already played 6. Trying to make a move after the game has been won.
404	1. Requesting data for a game which does not exist, like GET /games/1000 GET /games/31351/moves

3. Deliverables

Submit two ZIP files to CourSys: https://courses.cs.sfu.ca/.

3.1 Shape

ZIP file of your project for part 1. Must contain a MainGUI (unmodified from course website beyond uncommenting/commenting out code). See directions on course website.

3.2 WebApp

ZIP file of your project for part 2. See directions on course website.

Please remember that all submissions will automatically be compared for unexpected similarities.