# Assignment 3: Shapes and Online Superhero Tracker (20% of course total)

- Due date is **17 July, 11:59pm**. Refer to Canvas for any updates.
- Submit deliverables to the corresponding assignment folder on Canvas.
- Make sure you also complete the corresponding online quiz at CourSys to receive full portion of the coding part. Refer to Course Outline and Marking Guide for details.
- Late penalty is 10% per calendar day (each 0 to 24 hour period past due), max. 2 days late (20%)
- This assignment is to be done individually. Do not show another student your code, do not copy code from another person/online. Please post all questions on Canvas.
- You may use general ideas you find online and from others, but your solution must be your own.
- You may use any tool for development. But we will be grading your submission using IntelliJ IDEA Community, so make sure your code compiles and runs there. We support only IntelliJ.

## Description

This assignment has two separate parts and are independent of each other, focusing on different topics.

### Part 1: Shapes

In this part you are going to implement a system for drawing basic shapes using blocks (cells) in a graphical user interface (GUI). A Canvas class is provided to handle the basic drawing. The shape classes use an inheritance hierarchy to build up functionalities without repeating code. The relationships of these classes are show in Figure 1. The provided code, plus the shape classes are described below.

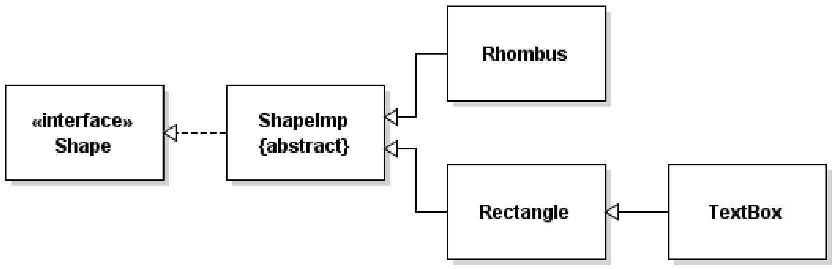


*Figure 1. Shape class hierarchy.*

You will create a Shape interface and write classes that implement it, including a ShapeImpl abstract class which provides a base-level implementation of shared functionality required by derived classes, and some concrete classes (Rhombus, Rectangle, and TextBox) that will be able to draw themselves into the provided Canvas class (and thus work with the interface expected by MainGUI).

#### *Provided Code*

The following classes provide the foundation of your application (Figure 2 shows a simplified UML class diagram for some of these classes):

- 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 cells. Supports making each cell a different colour and showing a character in the cell.
- CanvasIcon: Support class used to make a Canvas able to be drawn on the screen.

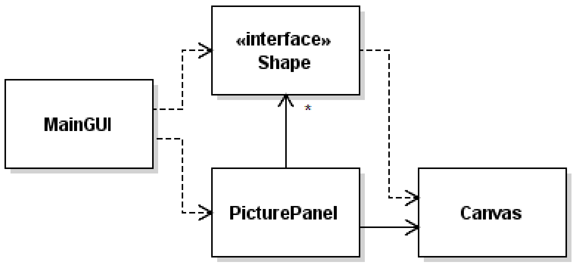


*Figure 2. Simplified UML class diagram of shapes and client code.*

### *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 cold related to `JFrame`, `JLabel`, `CanvasIcon`, and `PicturePanel` (they are code rendering the canvas into a graphical Swing GUI).
- Note that this code only shows how to interact with the `Canvas`. It does not show how to make actual shapes. See Figure 4 and sample output at the end of this document for the shapes that the provide `MainGUI` code produces once you have implemented the required classes.
- Other "`make…()`" functions in `MainGUI` have been commented out. Uncomment them as you implement the classes to try out the rest of the example canvas.

### *MainGUI*

The `MainGUI` program uses your shape classes to show some "pictures". This application is structured to allow you to test each shape one at a time during development. For example, Figure 4 shows its output for the rectangle picture.

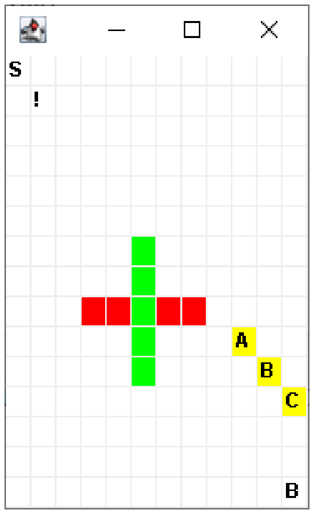


*Figure 3. From DemoCanvasGUI.*

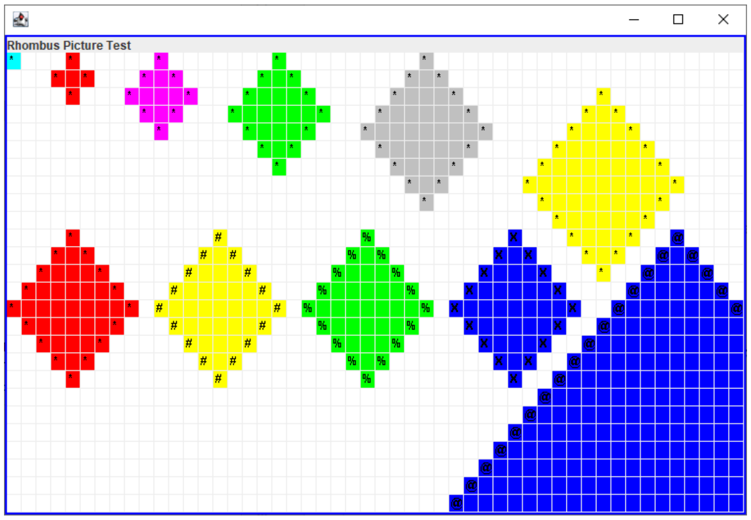


*Figure 4. From MainGUI drawing Rhombuses. The top 3 rhombuses have sizes 1, 2, & 3.*

[1] Yes we spell "colour" with a 'u', but since Java doesn't, we dropped it in the code to keep things consistent.

The application's `main()` calls a number of `makeXYXPicture()` methods to generate some predesigned `PicturePanel`s. You should not need to modify this file at all, other than un/commenting the functions one by one as you implement the necessary shapes. During testing you may also comment out them to make the UI fit your screen.

When marking, we will likely replace this file with a clean copy to ensure there were no inadvertent edits or add extra test cases. To support marking, this program also writes each `PicturePanel` to a file. You can safely ignore them if you like – they are generated so we can quickly diff your output to the reference implementation.

### *Required Object Structure*

Your code must be structured as follows:

- Use a package hierarchy:
    - Package `ca.cmpt213.a3`: Provided GUI program files along with `main()`.
    - Package `ca.cmpt213.a3.UI`: Provided classes for drawing to the screen (`Canvas`, `CanvasIcon`, `PicturePanel`).
    - Package `ca.cmpt213.a3.shapes`: All the shape classes/interfaces.
- Create a `Shape` interface
    - Each concrete shape must support the method (required by `PicturePanel`): `void draw(Canvas canvas);`
- Create a `ShapeImp` abstract class which provides a base-level implementation of shared functionalities required by its derived classes.
- Create a `Rhombus`, `Rectangle`, and `TextBox` class.
    - Each of these classes must extend the `ShapeImpl` abstract class.
    - Each of these classes must work with the interface expected by `MainGUI`. Of specific interest is their constructors.
    - For each of the `makeXYZPicture()` test methods, the `MainGUI` must generate the identical outputs as shown in this document.
    - There must not be much, if any at all, duplicate code between the classes. You must use inheritance effective.

### *Suggested OOD*

Figure 5 shows a *suggested* OOD for the shapes in this assignment. Each of the classes are described below. Both diagrams and the notes are just suggestions (with member fields and potential helper methods omitted). You may choose to do it in a different (but at least as good) way.

### Shape Details

An interface specifying methods that needs to be supported in order to draw them onto the canvas.

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

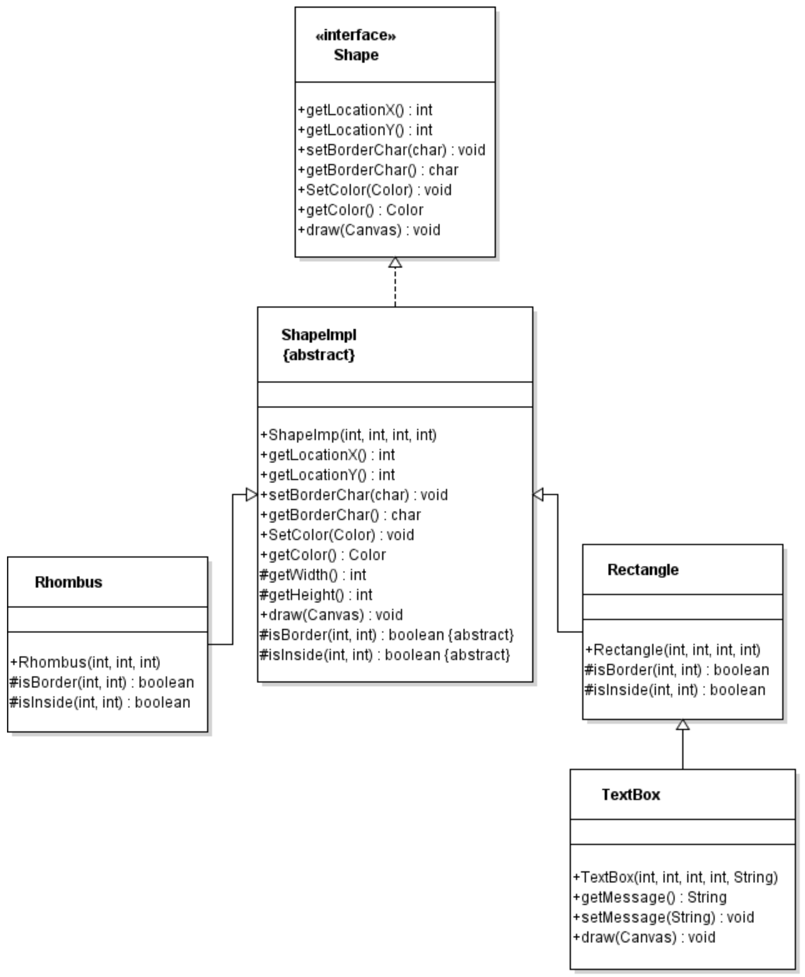


*Figure 5. Suggested UML design (member fields and potential helper methods omitted.*

## ShapeImpl Details

An abstract base class that implements much of the functionality specified by the Shape interface.

- Stores the location, border character (default is '*'), and colour (default is blue) as private fields.
- Implements the required getter and setter methods for these fields (to maintain encapsulation).
- Provides a constructor accepting the X & Y coordinates, plus the width & height of the Shape.
- Provides the `draw()` method that draws the shape into the `Canvas` parameter. However, since this is the abstract base class, it does not know exactly how to draw a specific shape. To do so, it must "ask" the derived class as follows:
    - The shape's width & 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, the `draw()` method asks the derived class if that position is:
        - A border cell (drawn with a coloured background & the border character) by calling `isBorder(xPos, yPos).`
        - An inner cell (drawn with just the coloured background) by calling `isInside(xPos, yPos).`
        - Otherwise it is neither a border nor inside and this not drawn at all.

  - Note that this is the "Template Method" design pattern (discussed in class) where:
    - The `draw()` method is the "template method".
    - The `isBorder()` & `isInside()` are the "primitive operations", and are abstract methods which concrete derived classes override.

### Rhombus Details

A concrete derived class from `ShapeImpl`, drawing a rhombus. See sample output for details.

- The X & Y coordinates are the top-left corner of the bounding box of the rhombus, not the top.
- The size is the number of cells on one side. All for sides of a rhombus are the same.

### Rectangles Details

A concrete derived class from ShapeImpl, drawing rectangle. See sample output for details.

- The X & Y coordinates are the top-left corner of the rectangle.
- The width & height are the dimensions of the rectangle.

### TextBox Details

A class with the "is-a" relationship with `Rectangle`, drawing rectangle and adding a message inside it. See sample output for details.

- When drawn on the canvas, it places the text inside the box drawn by the `Rectangle`.
  - Use overriding to create a custom draw method, but use the base class's `draw()` implementation.
  - Do not repeat the drawing code from `Rectangle`. You wrote it once, it works, use it.
- Layout of the text is non-trivial:
  - Essentially it mimics the "center" alignment of textboxes in applications like Powerpoint with monospace fonts such as `Courier New`:
    - If the message is too long to fit in one line, it must be split across multiple lines.
    - To break the long text into parts, first attempt to break on a space if there is one; otherwise fill the entire line inside the Rectangle with text and break mid-word as needed (and continue on the next line, if available).
    - Trailing space for each line is ignored (i.e., does not affect the centering of text) and can in theory go as long as it needs because it is invisible (try this behaviour in a Powerpoint textbox with center alignment).
    - Leading space of every line is ignored as they can be viewed as trailing space of the previous line; except in the first line, which will affect the centering of text.
  - Each line of text must be centered horizontally inside the `Rectangle`.
    - If there are odd number of extra spaces place the extra space before the text, for example, "`   Hi  `".
  - Text starts at the top row inside the `Rectangle` and must not overlap its border.
  - If there is more text than it will fit inside the `Rectangle` then some text will not be displayed (i.e., truncated).

*there is no space between the words for "Thin text in a box!" in makeMixedPicture() because all of the space are leading space in non-first line (verify this in Powerpoint using a very thin textbox).

## Part 2: Online Superhero Tracker

In this part you are going to create an online (but simplified) version of the Superhero Tracker system in Assignment 1 using Sprint Boot REST API.

Your system (the server) must respond to these messages from the client using curl commands:

| HTTP method & URL | Description |
|---|---|
| **GET /hello** | Return a string greeting the user saying:<br>Hello welcome to Superhero Tracker Online! |
| **GET /listAll** | Return all superheroes as a JSON array object. |
| **POST /add** | Add a new superhero (and respond with status 201). Body of request is a JSON object representing the superhero, for example,<br>{<br> "name" : "Batman",<br> "heightInCm" : 183.0,<br> "civilianSaveCount" : 100,<br> "superPower" : "Super rich"<br>}<br>• Add an id (type `long`) to the `SuperHero` class that you create to store a superhero. Each time a superhero is added this id is incremented by 1. Use `AtomicLong` and its `incrementAndGet()` method to achieve that.<br>• Return the newly created superhero object.<br>• The system will only check for non-negative values of height and civilian saved. If that happens, throw an `IllegalArgumentException` and respond with status 400. |
| **POST /remove/id** | Remove a superhero from the system indicated by id.<br>• Return the resulting list of superheroes as a JSON array object.<br>• The system will only check for non-existing id. If that happens, throw a `SuperheroNotFoundException` (create one yourself by extending `RuntimeException`) and respond with status 404. |
| **POST /update/id** | Update information of a superhero in the system indicated by id. Body of request is a JSON object representing the superhero, for example,<br>{<br> "name" : "Batman",<br> "heightInCm" : 183.0,<br> "civilianSaveCount" : 100,<br> "superPower" : "Super rich"<br>}<br>• Return the updated superhero object.<br>• The system will check for non-negative values of height and civilian saved (though all four fields can be changed). If that happens, throw an |

| | |
|---|---|
| | IllegalArgumentException and respond with status 400.<br>• The system will also check for non-existing id. If that happens, throw a SuperheroNotFoundException (create one yourself by extending RuntimeException) and respond with status 404. |
| **GET /listTop3**<br>***This functionality is optional*** | Return the top 3 superheroes who have saved the most civilians as a JSON array object. If there are not enough to show, throw a NotEnoughSuperheroesException (create one yourself by extending RuntimeException) and respond with status 404. |

More details:

- All data exchanged between the server and client is in JSON format (except for /hello which is returned as just plain text).
- The JSON objects should be printed in "pretty format". To do so you will need to create a class provided here (for details watch the video Making a REST API using Spring Boot in IntelliJ below) https://github.com/drbfraser/SpringBootIntro/blob/master/src/main/java/ca/pledgetovote/controllers/MakeSpringPrettyPrintJSON.java
- Unless otherwise specified, HTTP status of successful GET responses are 200 (OK) and POST are 201 (created).
- Unless otherwise specified, responses from successful post requests return the object just created.
- Do not save the state of the games between executions of the server (no persistence), that is, there is no need to save the superheroes in a JSON file.

### *Using Spring Boot REST API*

Spring Boot is a tool for Java to quickly program a web service.

You will be using the Gradle build system in IntelliJ to import all the necessary dependencies for Spring Boot to work. In the build.gradle file, use the following code*:

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-
plugin:2.2.2.RELEASE")
    }
}

plugins {
    id 'java'
    id 'org.springframework.boot' version '2.2.2.RELEASE'
    id 'io.spring.dependency-management' version '1.0.7.RELEASE'
}

group 'OnlineSuperheroTracker'
version '1.0-SNAPSHOT'
sourceCompatibility = 1.8
```

```
repositories {
    mavenCentral()
}

dependencies {
    compile("org.springframework.boot:spring-boot-starter-web")
}
```

*Note that in the current IntelliJ version (2020.1) the default gradle wrapper version is not compatible with JDK 14. To keep using JDK 14 as with other projects of the course, first follow the instructions at the 1:00 mark of the “Setting up Spring Boot in IntelliJ“ video to download the latest gradle (6.5). Once you have the gradle project created in IntelliJ, go to File > Settings > Gradle to point your project to the new gradle, along with using JDK14 for Gradle JVM. After that you will have to manually create a source directory (src) in the project by right-clicking the project name and choose New > Directory, which brings up four directories (src/main/java, src/main/resources, src/text/java, src/text/resources). Choose all. Read this for reference: https://stackoverflow.com/questions/49710330/src-folder-not-created-when-creating-simple-intellij-java-gradle-project. Lastly, set the content of the build.gradle file as shown, and click on the reload icon (or by opening the Gradle panel on the right) to load all the Spring Boot resources.

Then follow the instructions at the 6:00 mark of the “Setting up Spring Boot in IntelliJ” video to create an Application class and onwards.

### *Curl Commands*

The cURL tool is a command letting users to send and receive data between a client and a server. It is available in recent Windows. It supports many options but here we only need to do simple GETs/POSTs. For example, the following sends a GET request to a localhost server.

```
curl -i -H "Content-Type: application/json" -X GET localhost:8080/hello
```

For details read the supplied curl-commands.txt file and watch the provided videos in this document.

### *Required Object Structure*

Your code must be structured as follows (in a separate project from Part 1):

- Use a package hierarchy:
    - Package `ca.cmpt213.a3.onlinesuperherotracker`
        - The `Application` class acting as the starting point of the program.
    - Package `ca.cmpt213.a3.onlinesuperherotracker.controllers`
        - The `MakeSpringPrettyPrintJSON` class for formatting the JSON object.
        - The `SuperheroController` class for handling all the get/post requests
        - The `SuperheroNotFoundException` class for representing the exception. If you choose to implement the optional listTop3 get request, create an additional `NotEnoughSuperheroesException` class.
    - Package `ca.cmpt213.a3.onlinesuperherotracker.model`
        - The `Superhero` class representing a superhero. Basically the same as your Assignment 1, but remember to add an id field in the primitive type `long`.

## Overall Coding Requirements

- Your code must conform to the programming style guide for this course; see course website.
- All classes must have a class-level JavaDoc comment describing the purpose of the class.
- Include your student information (name, student ID, email) as comment in all files.

## Submission Instructions

- Submit a ZIP file of your two parts in separate projects to the corresponding folder on Canvas. Name it using this format: **<firstname_lastname>_<studentID>_Assignment3.zip**. Note that you don’t need to include ‘<’ & ‘>’, and studentID is the 9-digit number on your student ID card. See course website for directions on creating and testing your ZIP file for submission. If you have any difficulties in submitting your file on Canvas, email it to the instructor.
- Save a copy in a secure location for save keeping and do not modify it after the deadline.
- If you use any libraries they have to be included in the project as well.
- All submissions will automatically be compared for unexplainable similarities

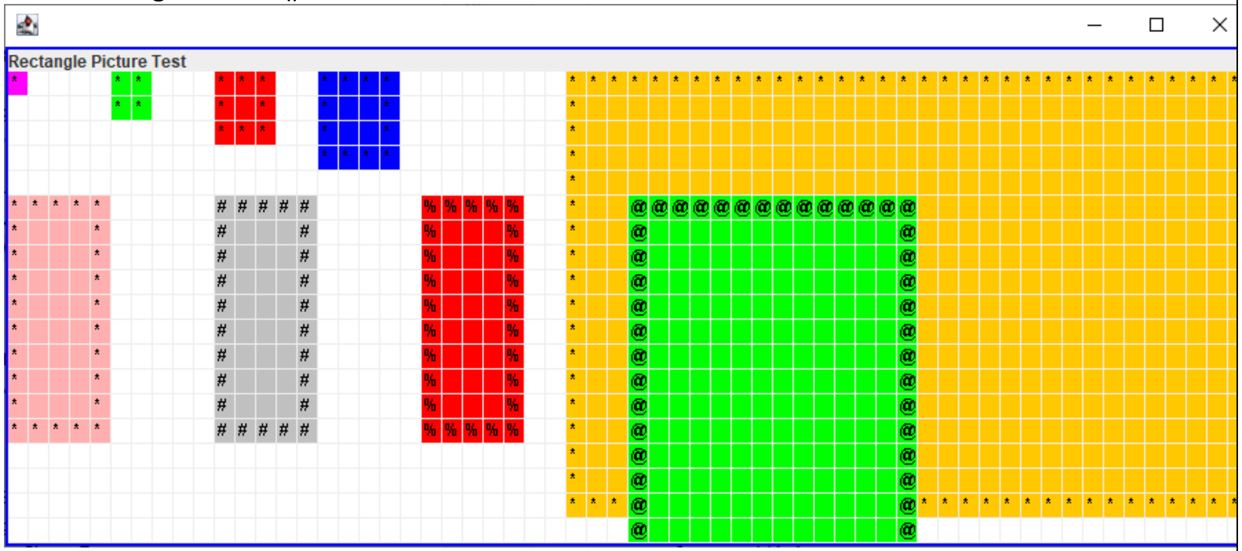
## Useful Resources

- Creating a Gradle project in IntelliJ: https://www.jetbrains.com/help/idea/gradle.html
- Setting up Spring Boot in IntelliJ by Dr. Brian Fraser: https://www.youtube.com/watch?v=he63dwZdhOM
- Making a REST API using Spring Boot in IntelliJ by Dr. Brian Fraser: https://www.youtube.com/watch?v=rXBsnNCH59o
- Using the Postman tool to test the REST API (optional): https://www.youtube.com/watch?v=-HTV4NTKo7I
- Building a RESTful Web Service: https://spring.io/guides/gs/rest-service/

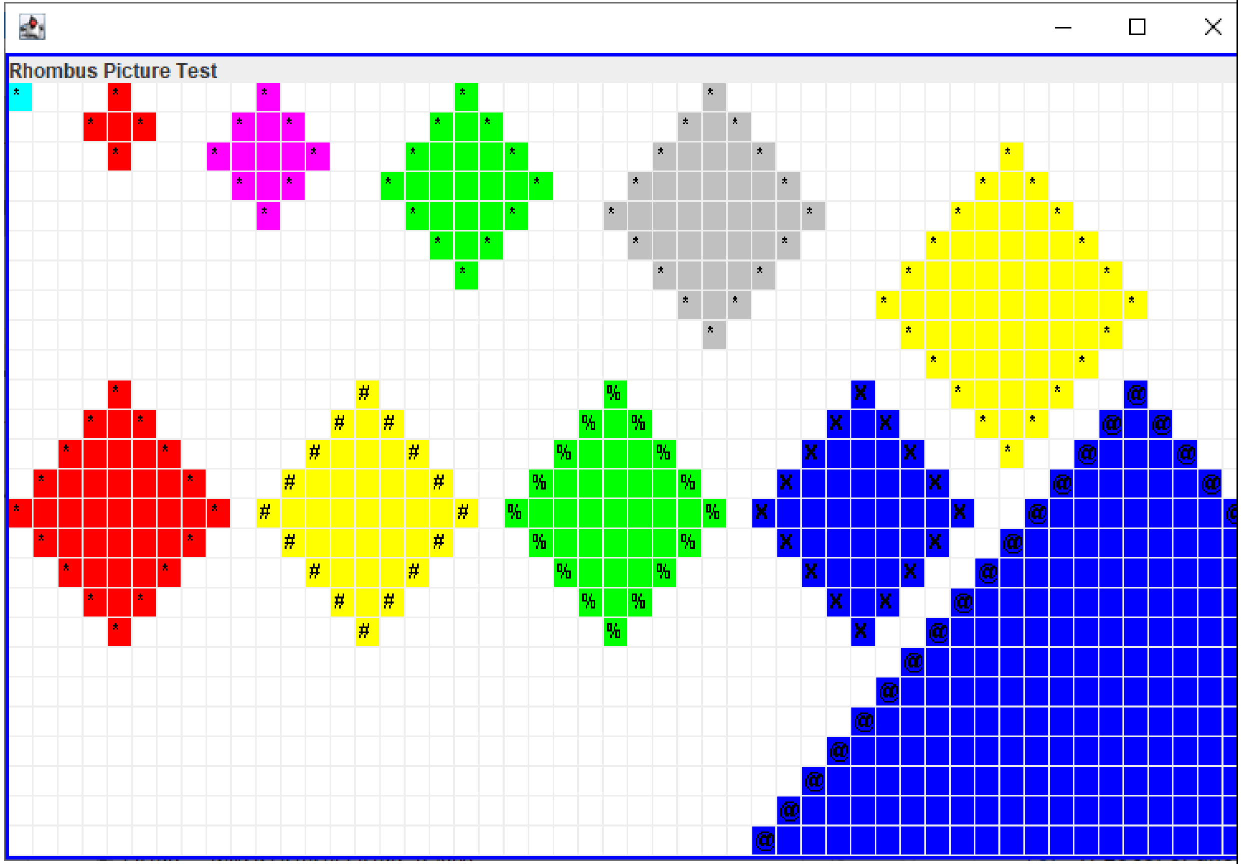
## Sample Input/Output

Part 1: Shapes

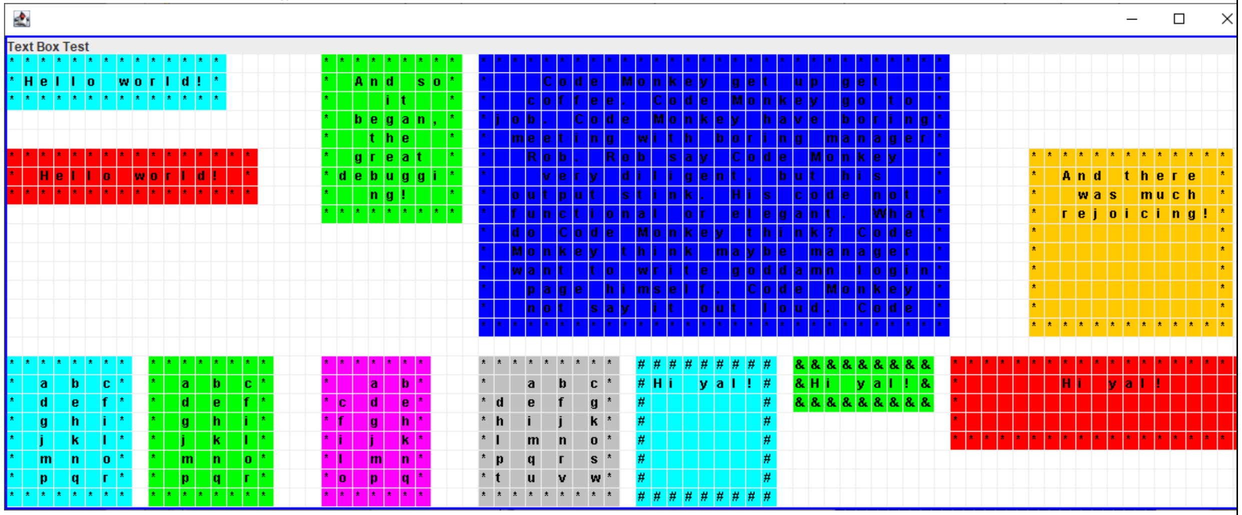
makeRectanglesPicture()



makeRhombusesPicture()

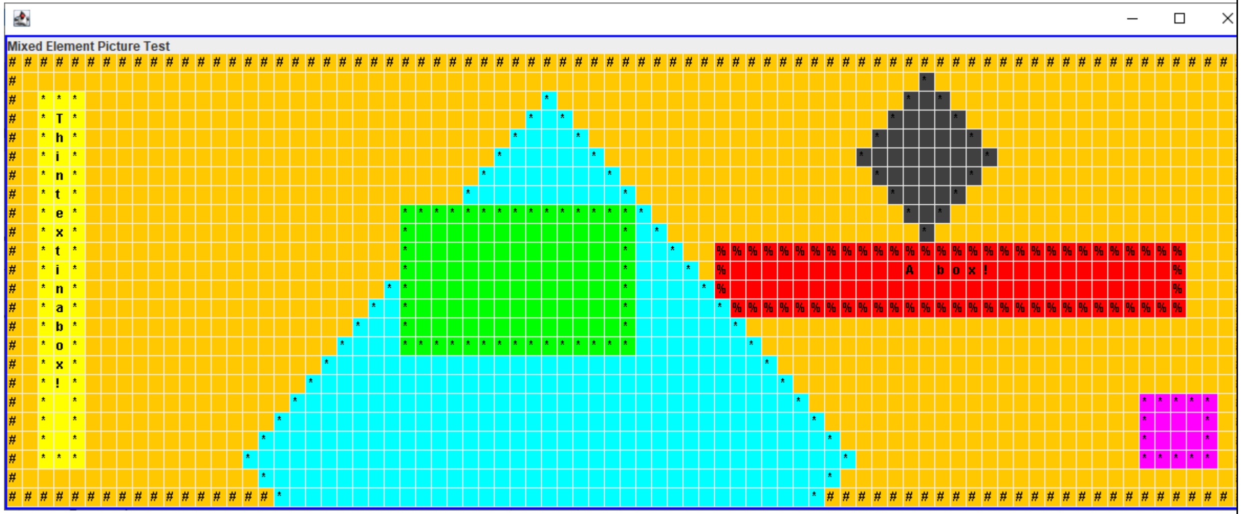


makeTextBoxesPicture()


Text Box Test

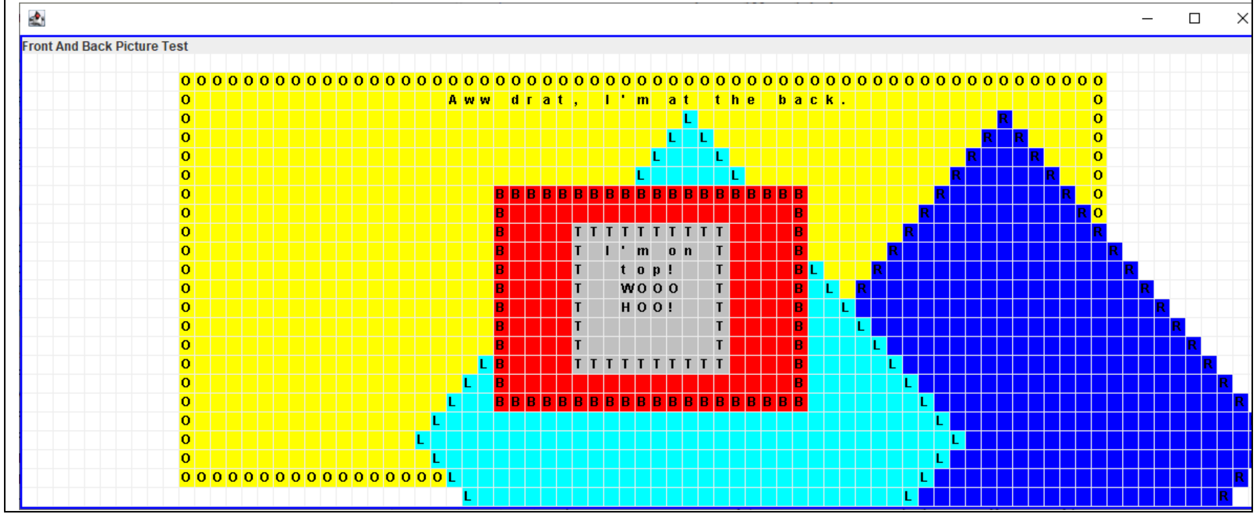
```
* * * * * * * * * * * * * *
* H e l l o   w o r l d ! *
* * * * * * * * * * * * * *

* * * * * * * * * * * * * * * *
*   H e l l o   w o r l d !   *
* * * * * * * * * * * * * * * *

* * * * * * * * *
*   A n d   s o *
*       i t     *
*   b e g a n , *
*     t h e     *
*   g r e a t   *
* d e b u g g i *
*     n g !     *
* * * * * * * * *

*     C o d e   M o n k e y   g e t   u p   g e t       *
*   c o f f e e .   C o d e   M o n k e y   g o   t o   *
* j o b .   C o d e   M o n k e y   h a v e   b o r i n g *
*   m e e t i n g   w i t h   b o r i n g   m a n a g e r *
*     R o b .   R o b   s a y   C o d e   M o n k e y     *
*       v e r y   d i l i g e n t ,   b u t   h i s       *
*   o u t p u t   s t i n k .   H i s   c o d e   n o t   *
*   f u n c t i o n a l   o r   e l e g a n t .   W h a t *
*   d o   C o d e   M o n k e y   t h i n k ?   C o d e   *
*   M o n k e y   t h i n k   m a y b e   m a n a g e r   *
*   w a n t   t o   w r i t e   g o d d a m n   l o g i n *
*     p a g e   h i m s e l f .   C o d e   M o n k e y   *
*     n o t   s a y   i t   o u t   l o u d .   C o d e   *

*   A n d   t h e r e   *
*     w a s   m u c h   *
*   r e j o i c i n g ! *

*   a   b   c *     *   a   b   c *     *   a   b *     *   a   b   c *     # H i   y a l ! #     & H i   y a l ! &     H i   y a l !
*   d   e   f *     *   d   e   f *     * c   d   e *   * d   e   f   g *   #               #
*   g   h   i *     *   g   h   i *     * f   g   h *   * h   i   j   k *   #               #
*   j   k   l *     *   j   k   l *     * i   j   k *   * l   m   n   o *   #               #
*   m   n   o *     *   m   n   o *     * l   m   n *   * p   q   r   s *   #               #
*   p   q   r *     *   p   q   r *     * o   p   q *   * t   u   v   w *   #               #
```

makeMixedPicture()


Mixed Element Picture Test

```
T h i n t e x t i n a b o x !
A   b o x !
```

makeFrontAndBackPicture()


Front And Back Picture Test

```
A w w   d r a t ,   I ' m   a t   t h e   b a c k .
I ' m   o n
t o p !
W O O O
H O O !
```

Part 2: Online Superhero Tracker

```
C:\Users\Victor>curl -i -H "Content-Type: application/json" -X GET localhost:8080/listAll
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sun, 28 Jun 2020 22:26:01 GMT

[ ]
C:\Users\Victor>curl -i -H "Content-Type: application/json" -X POST -d "{ \"name\": \"Batman\",
\"heightInCm\": 183.0, \"civilianSaveCount\": 100, \"superPower\": \"Super rich\" }"
localhost:8080/add
HTTP/1.1 201
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sun, 28 Jun 2020 22:26:06 GMT

{
  "id" : 1,
  "name" : "Batman",
  "heightInCm" : 183.0,
  "civilianSaveCount" : 100,
  "superPower" : "Super rich"
}
C:\Users\Victor>curl -i -H "Content-Type: application/json" -X POST -d "{ \"name\": \"Captain
Marvel\", \"heightInCm\": 170, \"civilianSaveCount\": 200, \"superPower\": \"Proton Blast\" }"
localhost:8080/add
HTTP/1.1 201
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sun, 28 Jun 2020 22:26:10 GMT

{
  "id" : 2,
  "name" : "Captain Marvel",
  "heightInCm" : 170.0,
  "civilianSaveCount" : 200,
  "superPower" : "Proton Blast"
}
C:\Users\Victor>curl -i -H "Content-Type: application/json" -X POST -d "{ \"name\": \"Storm\",
\"heightInCm\": 200, \"civilianSaveCount\": 300, \"superPower\": \"Control Weather\" }"
localhost:8080/add
HTTP/1.1 201
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sun, 28 Jun 2020 22:26:17 GMT

{
  "id" : 3,
  "name" : "Storm",
  "heightInCm" : 200.0,
  "civilianSaveCount" : 300,
  "superPower" : "Control Weather"
}
C:\Users\Victor>curl -i -H "Content-Type: application/json" -X POST -d "{ \"name\": \"Earthworm
Jim\", \"heightInCm\": 173, \"civilianSaveCount\": 0, \"superPower\": \"Superior Strength\" }"
localhost:8080/add
HTTP/1.1 201
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sun, 28 Jun 2020 22:26:22 GMT

{
  "id" : 4,
  "name" : "Earthworm Jim",
  "heightInCm" : 173.0,
  "civilianSaveCount" : 0,
  "superPower" : "Superior Strength"
}
C:\Users\Victor>curl -i -H "Content-Type: application/json" -X GET localhost:8080/listAll
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sun, 28 Jun 2020 22:26:28 GMT

[ {
  "id" : 1,
  "name" : "Batman",
  "heightInCm" : 183.0,
  "civilianSaveCount" : 100,
  "superPower" : "Super rich"
}, {
  "id" : 2,
  "name" : "Captain Marvel",
  "heightInCm" : 170.0,
  "civilianSaveCount" : 200,
  "superPower" : "Proton Blast"
}, {
  "id" : 3,
  "name" : "Storm",
  "heightInCm" : 200.0,
  "civilianSaveCount" : 300,
  "superPower" : "Control Weather"
}, {
  "id" : 4,
  "name" : "Earthworm Jim",
  "heightInCm" : 173.0,
  "civilianSaveCount" : 0,
  "superPower" : "Superior Strength"
} ]
C:\Users\Victor>curl -i -H "Content-Type: application/json" -X POST -d "{ \"name\": \"Ultraman\",
\"heightInCm\": -2, \"civilianSaveCount\": 0, \"superPower\": \"Super rich\" }" localhost:8080/add
HTTP/1.1 400
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sun, 28 Jun 2020 22:26:39 GMT
Connection: close

{
  "timestamp" : 1593383199870,
  "status" : 400,
  "error" : "Bad Request",
  "message" : "Invalid values.",
  "path" : "/add"
}
C:\Users\Victor>curl -i -H "Content-Type: application/json" -X GET localhost:8080/listAll
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sun, 28 Jun 2020 22:26:42 GMT

[ {
  "id" : 1,
  "name" : "Batman",
  "heightInCm" : 183.0,
  "civilianSaveCount" : 100,
  "superPower" : "Super rich"
}, {
  "id" : 2,
  "name" : "Captain Marvel",
  "heightInCm" : 170.0,
  "civilianSaveCount" : 200,
  "superPower" : "Proton Blast"
}, {
  "id" : 3,
  "name" : "Storm",
  "heightInCm" : 200.0,
  "civilianSaveCount" : 300,
  "superPower" : "Control Weather"
}, {
  "id" : 4,
  "name" : "Earthworm Jim",
  "heightInCm" : 173.0,
  "civilianSaveCount" : 0,
  "superPower" : "Superior Strength"
} ]
C:\Users\Victor>curl -i -H "Content-Type: application/json" -X POST localhost:8080/remove/3
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sun, 28 Jun 2020 22:26:48 GMT

[ {
  "id" : 1,
  "name" : "Batman",
  "heightInCm" : 183.0,
  "civilianSaveCount" : 100,
  "superPower" : "Super rich"
}, {
  "id" : 2,
  "name" : "Captain Marvel",
  "heightInCm" : 170.0,
  "civilianSaveCount" : 200,
  "superPower" : "Proton Blast"
}, {
  "id" : 4,
  "name" : "Earthworm Jim",
  "heightInCm" : 173.0,
  "civilianSaveCount" : 0,
  "superPower" : "Superior Strength"
} ]
C:\Users\Victor>curl -i -H "Content-Type: application/json" -X POST localhost:8080/remove/100
HTTP/1.1 404
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sun, 28 Jun 2020 22:26:57 GMT

{
  "timestamp" : 1593383217640,
  "status" : 404,
  "error" : "Not Found",
  "message" : "Request ID not found.",
  "path" : "/remove/100"
}
C:\Users\Victor>curl -i -H "Content-Type: application/json" -X POST -d "{ \"name\": \"Batman\",
\"heightInCm\": 183.0, \"civilianSaveCount\": 150, \"superPower\": \"Super rich\" }"
localhost:8080/update/1
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sun, 28 Jun 2020 22:27:05 GMT
```

```
{
  "id" : 1,
  "name" : "Batman",
  "heightInCm" : 183.0,
  "civilianSaveCount" : 150,
  "superPower" : "Super rich"
}
C:\Users\Victor>curl -i -H "Content-Type: application/json" -X POST -d "{ \"name\": \"Batman\",
\"heightInCm\": 183.0, \"civilianSaveCount\": 150, \"superPower\": \"Super rich\" }"
localhost:8080/update/100
HTTP/1.1 404
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sun, 28 Jun 2020 22:27:13 GMT

{
  "timestamp" : 1593383233296,
  "status" : 404,
  "error" : "Not Found",
  "message" : "Request ID not found.",
  "path" : "/update/100"
}
C:\Users\Victor>curl -i -H "Content-Type: application/json" -X POST -d "{ \"name\": \"Captain
Marvel\", \"heightInCm\": -1, \"civilianSaveCount\": 200, \"superPower\": \"Proton Blast\" }"
localhost:8080/update/2
HTTP/1.1 400
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sun, 28 Jun 2020 22:27:23 GMT
Connection: close

{
  "timestamp" : 1593383243689,
  "status" : 400,
  "error" : "Bad Request",
  "message" : "Invalid values.",
  "path" : "/update/2"
}
C:\Users\Victor>curl -i -H "Content-Type: application/json" -X GET localhost:8080/listAll
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sun, 28 Jun 2020 22:27:27 GMT

[ {
  "id" : 1,
  "name" : "Batman",
  "heightInCm" : 183.0,
  "civilianSaveCount" : 150,
  "superPower" : "Super rich"
}, {
  "id" : 2,
  "name" : "Captain Marvel",
  "heightInCm" : 170.0,
  "civilianSaveCount" : 200,
  "superPower" : "Proton Blast"
}, {
  "id" : 4,
  "name" : "Earthworm Jim",
  "heightInCm" : 173.0,
  "civilianSaveCount" : 0,
  "superPower" : "Superior Strength"
} ]
C:\Users\Victor>
```