Objects

In this assignment, you will build a map editor for a vertically scrolling video game.

1. **Make a backup copy** of your previous assignment. If you do not already have a habit of regularly backing up your work, develop that habit now. These days, storage space is cheap. No one regrets making backups. You have been warned.

2. **Update your game window for testing.** Add a key press option in Controller to exit your Game window when either Q/q or esc is pressed. There are two different ways to exit this program and BOTH need to be in your code. Q/q will exit the Game window AND esc will exit the Game window. When we are testing your code, we will expect to be able to use these keyboard shortcuts and they are required! You are welcome to add more shortcuts in your code as it makes sense to you, but these the expected ones for right now.

   ```
   System.exit(0); //will quit your program's execution.
   ```

   Hint for key presses...

   Also, change the title of your window to "A3 - Map Editor" (or something of your choice that indicates which assignment this is and its general purpose) - this will make it easier to keep track of which assignment you're currently working on in case your IDE starts running a different project for you without letting you know!

3. **Make a class named "Wall"**. Add four private integer class member variables, named "x", "y", "w", and "h", to hold the position and dimensions of the wall. Create a constructor to initialize its position. Create some getters and setters for your class member variables. There is a "wall.png" file on the Blackboard site, but feel free to use your own.

4. **Load Wall image**. In View, load the wall image similarly to how you loaded the turtle image. Use appropriate variable names.

5. **Clean up**. **Update your build script** to also compile your new class as well. If you still have a button or a turtle in your program, remove those (or comment them out for now), and any variables and methods connected to moving the turtle or associated with the button (dest_x and dest_y would be included here!). When you program, it is important to stay organized. Always take time to use meaningful variable names, indent your code properly, add comments, and keep it readable. These assignments build on top of each other. If you make a complicated mess, you are the one who will have to deal with it later.

   At this point, you should have a blue screen with nothing shown when you run your code. Feel free to update the size of the window as needed.

6. **Make your model hold an ArrayList of Walls**. (Use your favorite search engine to search for "java ArrayList". Read about the "add", "get", and "remove" methods. Be sure to note whether you will need an import statement!) Add a member variable to Model like this:

   ```
   private ArrayList<Wall> walls;
   ```

Initialize it in your Model constructor like this:

```
walls = new ArrayList<Wall>();
```

Also, let's add a Wall to the Arraylist:

```
Wall wall = new Wall(250, 300, 200, 100);
walls.add(wall);
```

7. **Make your View draw all of the walls**. Remember where the turtle was being drawn in the last assignment? Put this, with appropriate modifications, there.

```
for(int i = 0; i < /*number of walls being stored in model's ArrayList*/; i++)
{
        Wall wall = /*get a specific wall out of the arraylist*/;
        g.drawImage(wallImage, wall.getX(), wall.getY(), wall.getW(), wall.getH(), null);
}
```

Remember that we are encapsulating our variables, so we need to use the appropriate getters and setters to access attributes in another class!

8. **Add a wall where the user indicates with the mouse**. The user will press, drag, and release the mouse button. A new wall should be placed with corners indicated by the location where the mouse was pressed and released. The user should be able to add as many walls to the screen as they like. (Also, get rid of that one wall that we initially put in the ArrayList. That was just there so you could see if the previous step worked.)

Hint...

You will likely start the wall upon mouse press (call a method in model to save/create (x1,y1) for the wall) and end the wall upon mouse release (call another method to calculate the width and height using (x2,y2), and find the upper left corner). Walls can be overlapping (we'll add remove functionality soon).

8-bit alternative...

9. **Add some additional functionality to Controller**. Add a variable to control an "editMode" (use 'e' and 'E') - this should toggle back and forth allowing the user to edit their map on the fly (this will make future projects easier to work with!).
Add a variable to control whether we are adding new Walls, or removing Walls (use 'a' and 'A'). As soon as editMode is toggled to true, we should be able to add new Walls to our map (add = true). If we toggle it to false, then we should be able to remove Walls (see below).
If the user presses 'c' or 'C', clear all walls that are being displayed on the screen (call a function in Model to clear everything).

10. **Add a method to detect when the user clicked on an existing Wall**. Add a method to your Wall class that takes two parameters, the x and y position where the user clicked. Return true iff the user clicked on the wall. Adjust your program such that when editMode=true && addWalls=false && the user clicks on an existing wall, it is removed.

11. **Enable the View to scroll over a larger model**. Add a private variable to the View* class to store a "camera" position. When you draw the walls, subtract the camera position from the vertical position of the wall For example:

```
g.drawImage(image, x, y - scrollPosY, w, h, null);
```

Adjust the scroll position value when the user presses the up or down arrow keys. (Adjust by some value greater than 1, so it is not painfully slow.)

*

12. **Read about JSON format**. JSON is a text-based format for encoding arbitrary objects. Visit http://json.org and examine the pictures. If you don't understand them, find a tutorial about JSON and figure out what it is all about. There is a Java class that enables one to load or save in JSON format on the Blackboard site called Json.java. Download it, add it to your project, update your build script, and study the example usage in the comments at the top of the file.

13. **When the user presses 's' or 'S', save your model to a Json file**. You can find "Json.java" on Blackboard - add it to your project, and **update your build script**. Examine the comments, which include an example of how to use it. Add a "marshal" method to your Wall class that marshals the object into a Json node. Also, add a "marshal" method to your Model class that marshals it into a Json node. Use the Json.save method to save the Json node to a file named "map.json".

14. **When the user presses 'l' or 'L', load the model from a Json file**. Add an unmarshaling constructor to your Wall class that will initialize its four member variables from Json objects. Also, add a method to Model that will clear the ArrayList of walls and unmarshal them from Json objects. Use the Json.load method to load the Json node from "map.json".

    Alternatively....

15. **Active KeyPresses/MousePresses for Assignment 3.**
    - <up arrow> - scroll up
    - <down arrow> - scroll down
    - l or L - load map
    - q or Q - quit program
    - c or C - clear map
    - e or E - toggle edit map
        - a or A - toggle add walls
            - if editing and adding walls, **mouse press** should start a wall, and **mouse release** should finish a wall
            - if editing and add walls is false, then **mouse press** should remove a clicked wall
        - As soon as edit is toggled to true, addWalls should also be immediately true. Toggle addWalls to false to remove walls instead.

          8-bit alternative note...

        - if edit is false, then neither mouse press or mouse release or mouse dragged should do anything
    - s or S - save map
    - esc - quit program

16. **Test your code and submit.** Make a zip archive called Assignment3.zip containing ALL of your source files (including map.json), image file(s), and a working build script. Do not submit any .class files or .jar files. Your build script (build.bash or build.bat only) should build and execute the program (**all** java files should be explicitly compiled). Test your zip file before you submit it by extracting into a separate folder and running your build script. You should be able to load your map from the assignment file! Remember to add your name, date, and assignment description at the top of EVERY java file you turn in for every assignment. Finally, go the course web site on Blackboard (https://learn.uark.edu/) and submit your zip archive for Assignment3.

17. Make sure your code meets each of these requirements. Please keep in mind that this is a very large class and the TAs do not have time to fix your code, rename files, etc.:
    1. Single zip file named Assignment3.zip (step 16)
        - Contains exactly 6 java files: JSON.java, Game.java, Model.java, Controller.java, View.java, Wall.java, 1 build script (build.bat or build.bash), 1 map.json file, and 1+ image(s) you need (named however) with no additional folders
        - There should be 9 files, with possibly more images bringing the total above 9
    2. Build script has the correct commands to build ALL files and run the code (steps 5 and 12)
        - Windows (.bat) and Linux (.bash) are different!
    3. **The code compiles (step 16)**
        - Do not skip testing! (especially because of the map and images!)
    4. The Java program executes correctly (step 15)
    5. Name, the date, and the assignment description is at the top of all files (step 16)
    6. Caps lock on the keyboard still gives same functionality? (step 2 hint)
    7. Button and turtle functionality are removed (step 5)
    8. Title bar of the window has been updated (step 2)
    9. Scrolling and adding walls still work (step 11)
    10. Walls can be drawn from any direction (step 8)
    11. When your code first loads, you should just have a blank screen. Upon keypress (lowercase L - 'l'), then your map should be loaded.

FAQ:

1. **Q: How does one use a try/catch block?**
   A: If you must use a try/catch block, make sure the catch part does not hide any errors. A good way to inform the user about an error is to print a stack trace and terminate the program. (If you don't terminate the program, you might not even notice that something bad happened.) Here is an example of a well-crafted try/catch block:

```
try
{
        image = ImageIO.read(new File("filename.png"));
}
catch(Exception e)
{
        e.printStackTrace(System.err);
        System.exit(1);
}
```

   Here is an example of the wrong way to use a try/catch block:

```
try
{
        image = ImageIO.read(new File("filename.png"));
}
catch(Exception e) {}  // BAD, BAD, BAD!!!!!!
```

   Also, it is a good idea to put as little code as you can inside the try section.