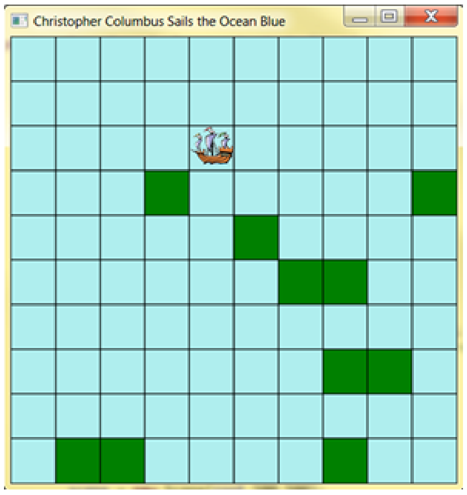
CSCI-513

Lab 2: Christopher Columbus Sails the Ocean Blue

Objective



This lab is a precursor to Homework #2. It will give you the opportunity to become more familiar with JavaFX before using it to complete your second homework assignment. The lab work involves completing a few guided steps in the JavaFX domain. The steps are part of the "Christopher Columbus" adventure – which will be extended for Homework 2.

In this adventure – you will (1) create a grid of rectangles representing an ocean, (3) place Christopher Columbus' ship onto the ocean, and (3) create a key event handler that moves the ship as you press up, down, left, and right arrows. If you have time, you will add islands to the ocean and integrate the logic that disallows the ship to sail through an island.

Design Thinking:

Create and display the "ocean" in JavaFX. Note: The ocean is initially a 10X10 grid of blue rectangles displayed in a Pane.

Before we start – let's think about the design. We are going to follow a very similar design to the Maze application and work with three classes: OceanExplorer (aka MazeGame), OceanMap (aka Maze), and Ship (aka Seeker). The OceanExplorer is going to include the main method, set up the JavaFX stage and screen, and include the key event handler. The OceanMap will include the grid (for now this will be quite simple – but it will get more interesting when we add islands etc). The ship tracks the current location of the ship and is responsible for move North, South, East, and West operations.

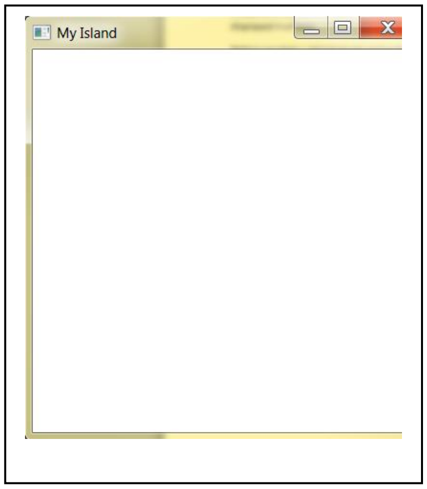
Step 1: Set up

1. Create a new java project called **ColumbusGame**
2. Create three new classes: OceanExplorer, OceanMap, and Ship. Make sure OceanExplorer contains a main() method.

Step 2: Create a Stage in Ocean Explorer

3. Make the OceanExplorer class extend Application. Import necessary packages and add the Start method (you can just accept Eclipse's recommendation to add unimplemented methods).
4. Add the statement:
   launch(args) to main(…).
5. Now you are ready to create and display a scene on the stage. In this first step we will just create an empty stage.

a. By default, the “Stage” argument passed to the start method is called primaryStage. You can name it something meaningful such as “oceanStage”.
b. Create a Pane of type AnchorPane (because we want to use specific X and Y coordinates). Look at the Maze example.
c. Create a new scene. (e.g. Scene scene = new Scene(myPane,width, height);)
d. Add the scene to the stage (e.g. stage.setScene(scene);)
e. Add a name to the stage (e.g. stage.setTitle(“…..”);
f. Finally – show the stage (e.g. stage.show());



You should have an empty stage displayed.

<u>**Step 3: Create and Draw the Ocean Grid**</u>

One of the things to be careful for here is that we need a scaling factor. Positions are in Pixels – so if you don’t scale your shape display – you will end up drawing a MINIATURE map in the top left hand corner. Note that all coordinates (x and y) start in the top left. X grows from left to right (as expected). Y grows from top to bottom. To handle the scaling factor, I normally create an int variable called scalingFactor which is used for placing shapes onto the scene.

1. In the OceanMap class create a 10X10 grid. For the current assignment we are going to use a Boolean 2D array (Boolean[10][10]) which simply tells us whether an ocean “cell” is empty (i.e. blue sea) or contains something. As you progress with your assignment you may wish to consider changing this to another type so you can get more information about what is actually in a cell. This is going to be very similar to your battleships grid. <u>For purposes of the lab – feel free to implement your grid in a different way if you prefer.</u>
2. Each “ocean” cell on the grid is going to be represented as a blue rectangle. For now, I suggest that the OceanMap class returns the entire grid to the OceanExplorer class. This is actually not a good design because it means that we are exposing a specific implementation detail of the OceanMap class. We will revisit this design decision in a couple of weeks and improve it.
   So OceanMap needs a method like this:

```
// Return generated map
public boolean[][] getMap(){
   return myGrid;  // Where hopefully “myGrid” has a more meaningful name.
}
```

3. In the OceanExplorer class you will need code to draw the ocean grid. Something like this:

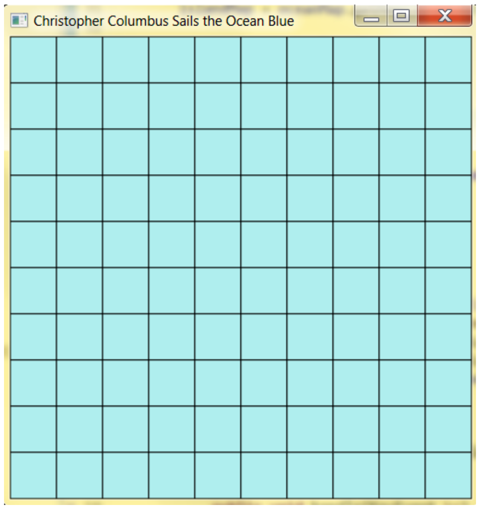
```
Create class level variables:
final int dimension = 10;  // We are creating 10X10 maps
final int scale = 50;  // Scale everything by 50.  You can experiment here.
```

```
oceanGrid = oceanMap.getMap();

for(int x = 0; x < dimensions; x++){
   for(int y = 0; y < dimensions; y++){
       Rectangle rect = new Rectangle(x*scale,y*scale,scale,scale);
      rect.setStroke(Color.BLACK);   // We want the black outline
      rect.setFill(Color.PALETURQUOISE);  // I like this color better than BLUE
      root.getChildren().add(rect);  // Add to the node tree in the pane
   }
}
```

Notice how my start method calls a drawMap() method. That would be a great place to put your code!!

Run your program and if you've done everything right you should now get a blue ocean!



4. Now we need to **add Christopher Columbus' ship to the ocean**. We'll use the Ship class to store the ship's current coordinates and to manage the movement of the ship. We'll handle the ship's image in the OceanExplorer class – because that class is responsible for GUI things.

(a) First create your ship class. For now it needs to track X and Y coordinates of the ship. Create a constructor that places the ship somewhere on the ocean. You could store the ship's grid coordinates using a Point class. Create a method that returns the ships current grid coordinates. I suggest something like:

```
public Point getShipLocation(){
   return currentLocation;
}
```

(b) Now load the image. Notice how my start method calls a "loadShipImage" method. In this step you are going to write that routine. It is going to load the ship image and add it to the pane.

You'll need to use the Image class and the ImageView class. Remember that you first load an image and then you place the image into the ImageView object. Finally you add the imageView object to the pane.

Something like this:

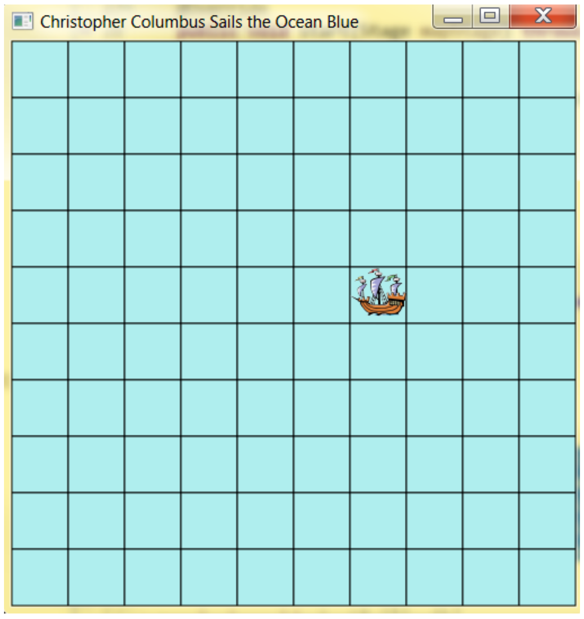
```
// Load image.  It takes 5 arguments:
// (1) file name, width, height, maintain ratio (T/F), smoothing (T/F)
Image shipImage = new Image("images\\ship.png",50,50,true,true);

// Now instantiate and load the image View.  Actually this probably needs to be
// a class level variable so you would already have defined ImageView shipImageview
shipImageView = new ImageView(shipImage);

// Assuming that you've already set the ship's starting coordinates in Ship and have
// created a getter method.
shipImageView.setX(oceanMap.getShipLocation().x * scale);
shipImageView.setY(oceanMap.getShipLocation().y * scale);

// Don't forget to add the ImageView to the Pane!
root.getChildren().add(shipImageView);
```

If all has gone well you will now see this when you run your program:



5. **Sailing Time**!

Finally, we are going to add a key event handler. One good place to put this is in a method called at the end of your start routine. The method could be part of the OceanExplorer class. If you create a method called startSailing() – then right after the stage.show() you should call startSailing();

What goes into the startSailing() method?

Basically it is an event handler exactly as we saw in the maze example in class. I have also posted the maze source code on d2l for your further investigation.

```
private void startSailing(){
  scene.setOnKeyPressed(new EventHandler<KeyEvent>(){

  @Override
  public void handle(KeyEvent ke) {
       switch(ke.getCode()){
              case RIGHT:
                 ship.goEast();
                 break;
              case LEFT:
                 ship.goWest();
                 break;
              case UP:
                 ship.goNorth();
                 break;
              case DOWN:
                 ship.goSouth();
                 break;
              default:
                 break;
       }
       shipImageView.setX(ship.getShipLocation().x*scale);
       shipImageView.setY(ship.getShipLocation().y*scale);
       }
   });
}
```

Notice that we are **delegating** the movement behavior to the Ship class. For example, when the RIGHT key code is recognized, then a method in ship goEast() is called to handle the movement. This method just checks to make sure the Ship is still within bounds of the grid if it moves east. If so – it updates the X coordinate of the ship.

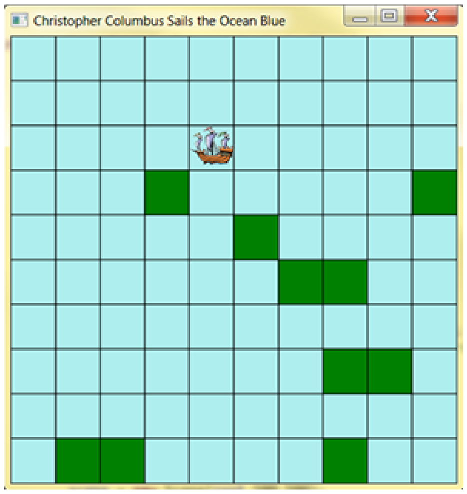
Following the switch statement – the shipImageView is updated by requesting current coordinates of the ship. Note that this could be done within the switch statement cases too!

As the shipImageView is already in the tree of nodes managed by the Stage, then updating the shipImageView coordinates means that the ship will actually move on the ocean. Try it for yourself! Your ship should move north, south, east, and west now while also remaining in the dimensions of the grid.

Well done! You have completed the lab!

6. **There is more**! To get a headstart on your homework assignment you could place some islands into the ocean. You'll need a new method in the OceanMap class that randomly places X islands (green rectangles) onto the grid. Previously all cells in the grid were set to false – now you will set some to true to represent islands. You could also change the type of the grid to int (where 0 = ocean, 1 = island, and 2 = pirate ship (needed for homework!). Place 10 islands at random positions on the map (not on top of each other). Island location should change each time you run the game. Modify your goEast, goWest, goNorth, and goSouth functions so that the ship

cannot sail through the islands. If you succeed in this step you will get something like this:



Homework Lookahead

Your second assignment will require you to complete this lab assignment and to add a pirate ship that uses the **observer pattern** to chase Christopher Columbus' ship. You'll also have the option to add a **few buttons** onto the grid to perform basic functions such as reset the game and to specify the number of islands to include in the game.

Lab Turnin

Push your completed project to GitHub.

This lab is VERY IMPORTANT for completing the homework assignment. I won't provide help with the homework assignment to anyone who doesn't first show evidence of completing the lab. So if you get stuck – ask for help with the lab first! I will help you with that.

Appendix A:

A snippet of the Ocean Explorer Class

```
public class OceanExplorer extends Application{

    boolean[][] islandMap;
    Pane root;
    final int dimensions = 10;
    final int islandCount = 10;
    final int scalingFactor = 50;
    Image shipImage;
    ImageView shipImageView;
    OceanMap oceanMap;
    Scene scene;
    Ship ship;

    @Override
    public void start(Stage mapStage) throws Exception {

        oceanMap = new OceanMap(dimensions, islandCount);
        islandMap = oceanMap.getMap(); // Note: We will revisit this in a

        root = new AnchorPane();
        drawMap();

        ship = new Ship(oceanMap);
        loadShipImage();

        scene = new Scene(root,500,500);
        mapStage.setTitle("Christopher Columbus Sails the Ocean Blue");
        mapStage.setScene(scene);
        mapStage.show();
        startSailing();
    }
```