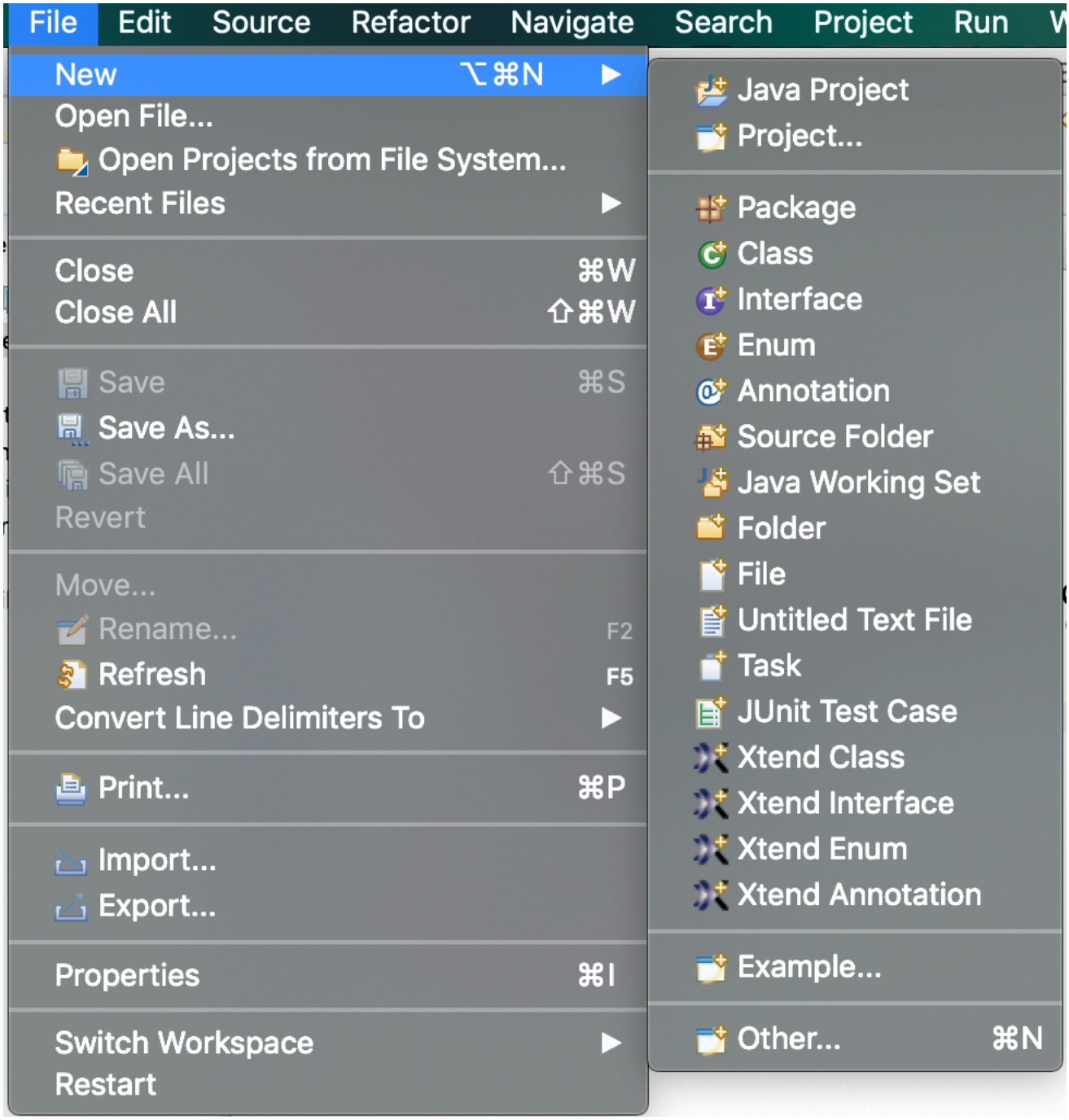
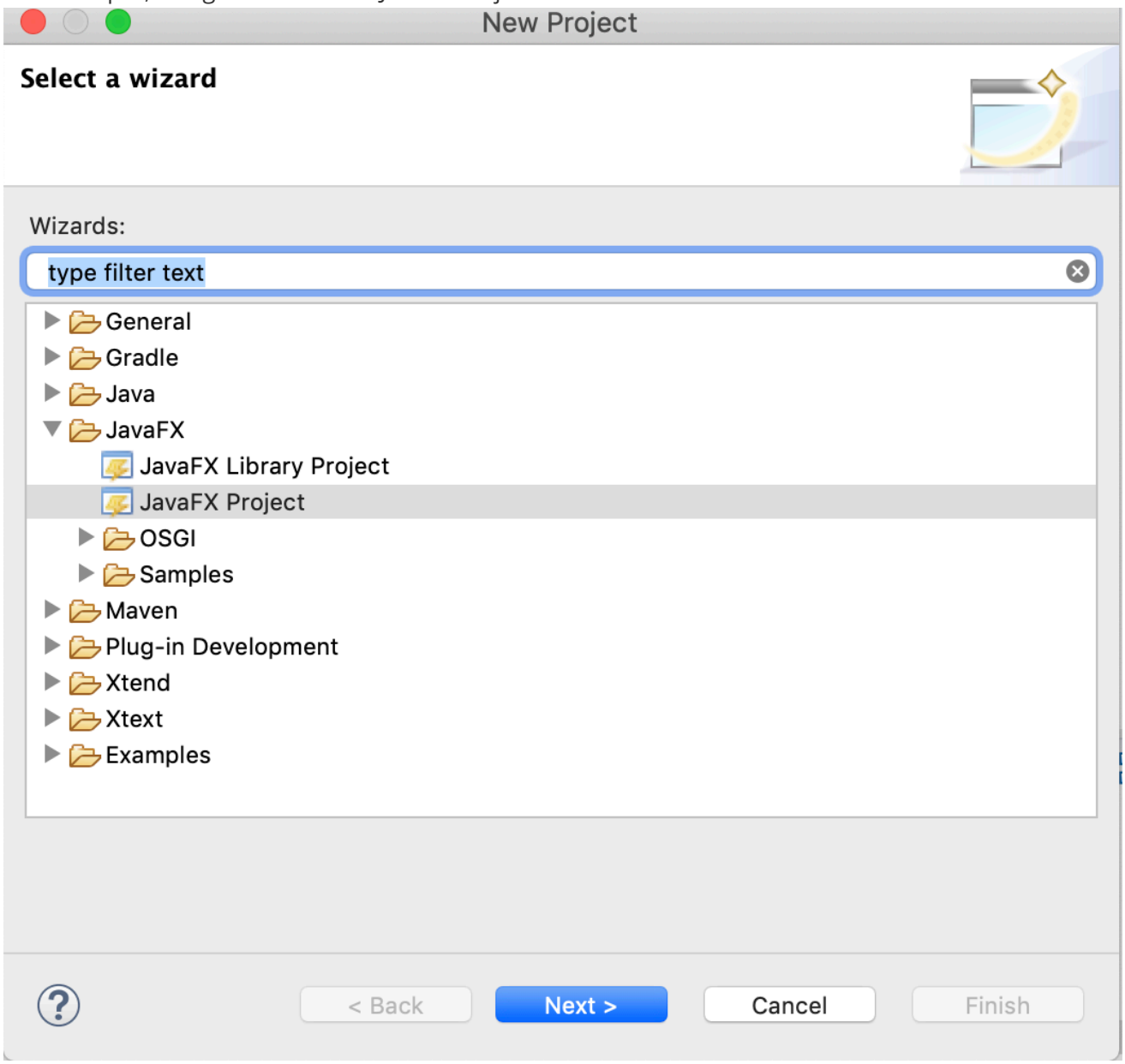
Tic Tac Toe:

Creating the project:

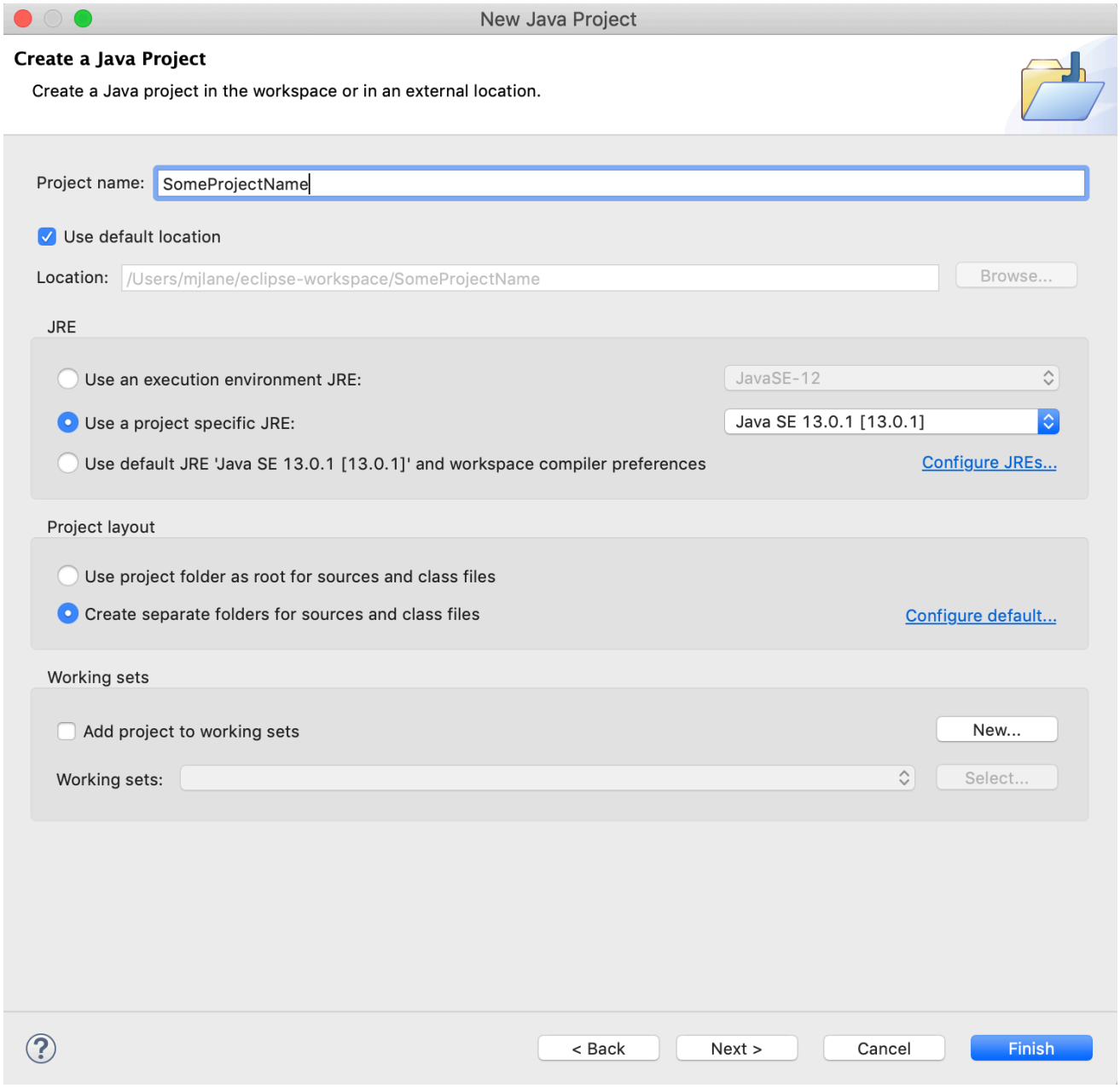
Click into the File >> New Project:



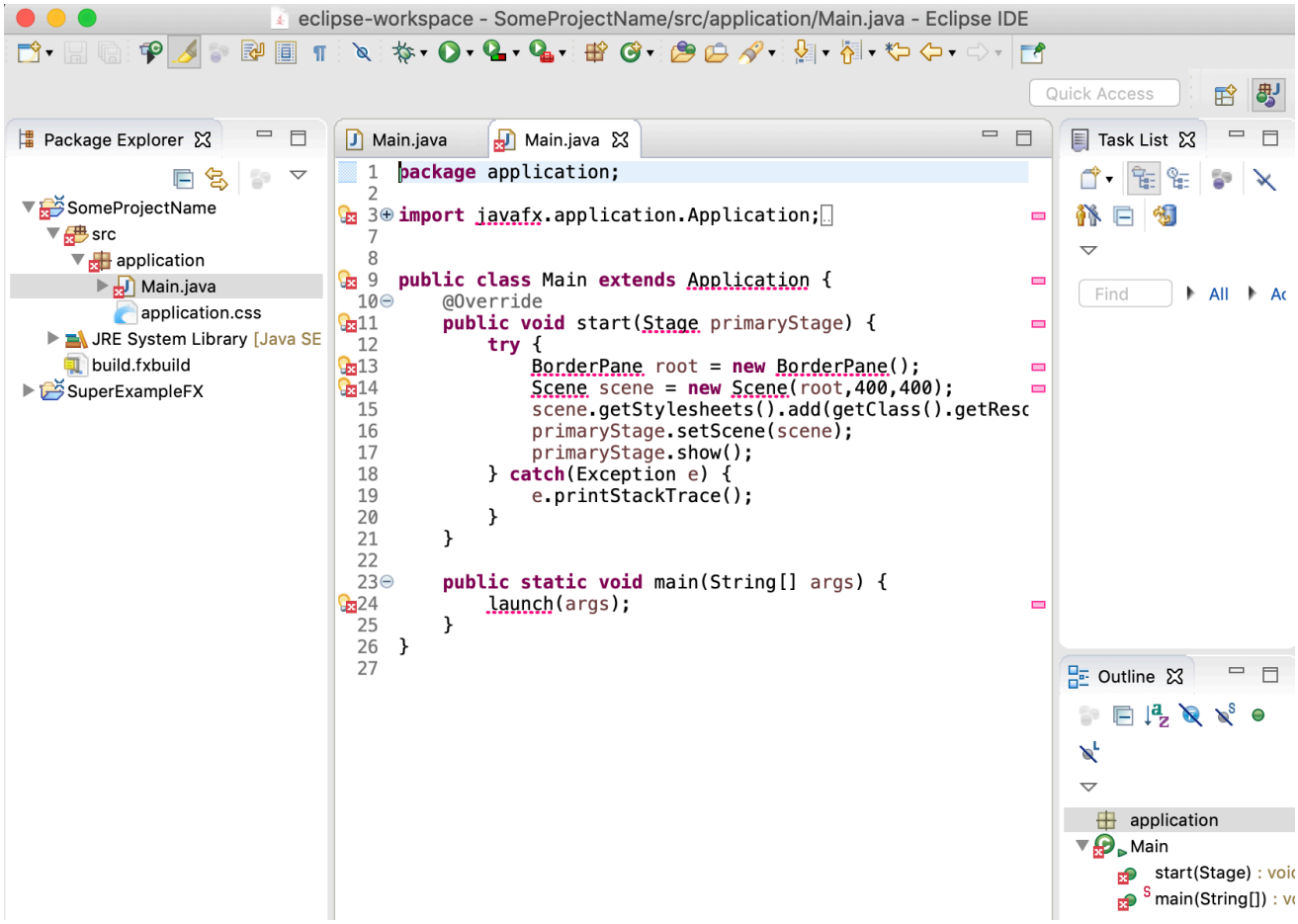
In the helper, navigate to the new JavaFX Project:



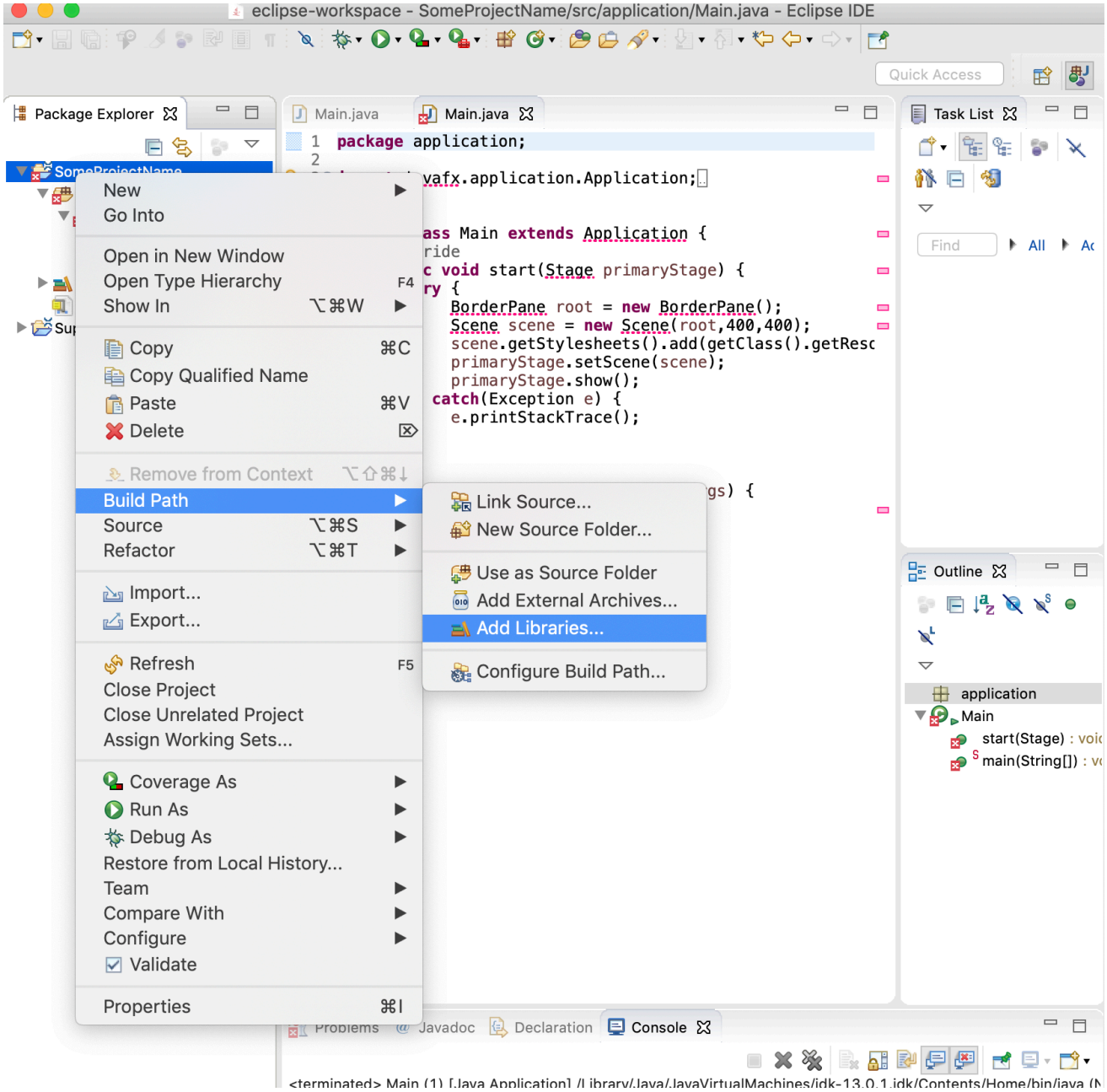
Select a name for your java project. Make sure you're using the right JRE (how to do this is noted in the installation guide):



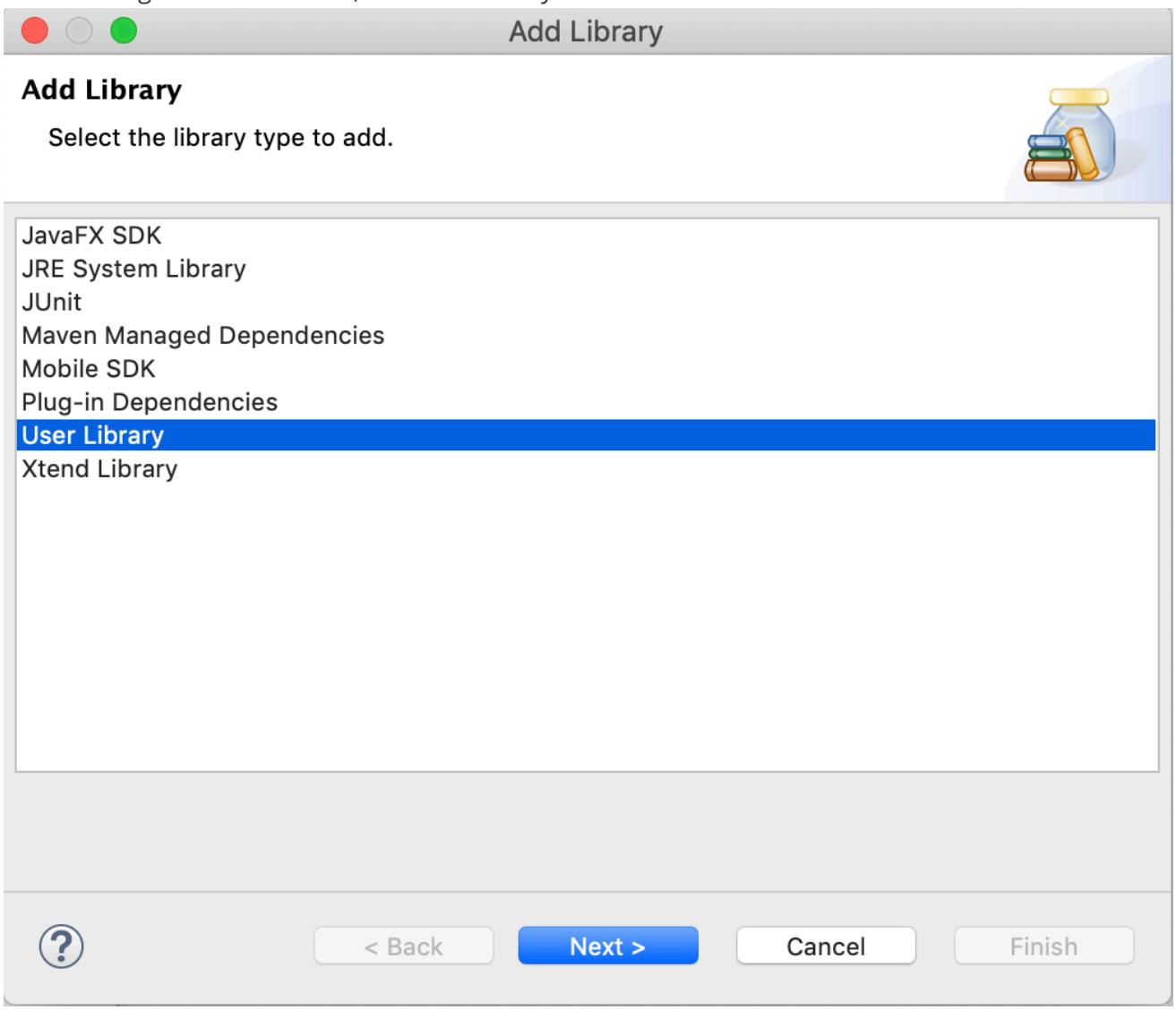
You'll notice that there are a ton of errors:



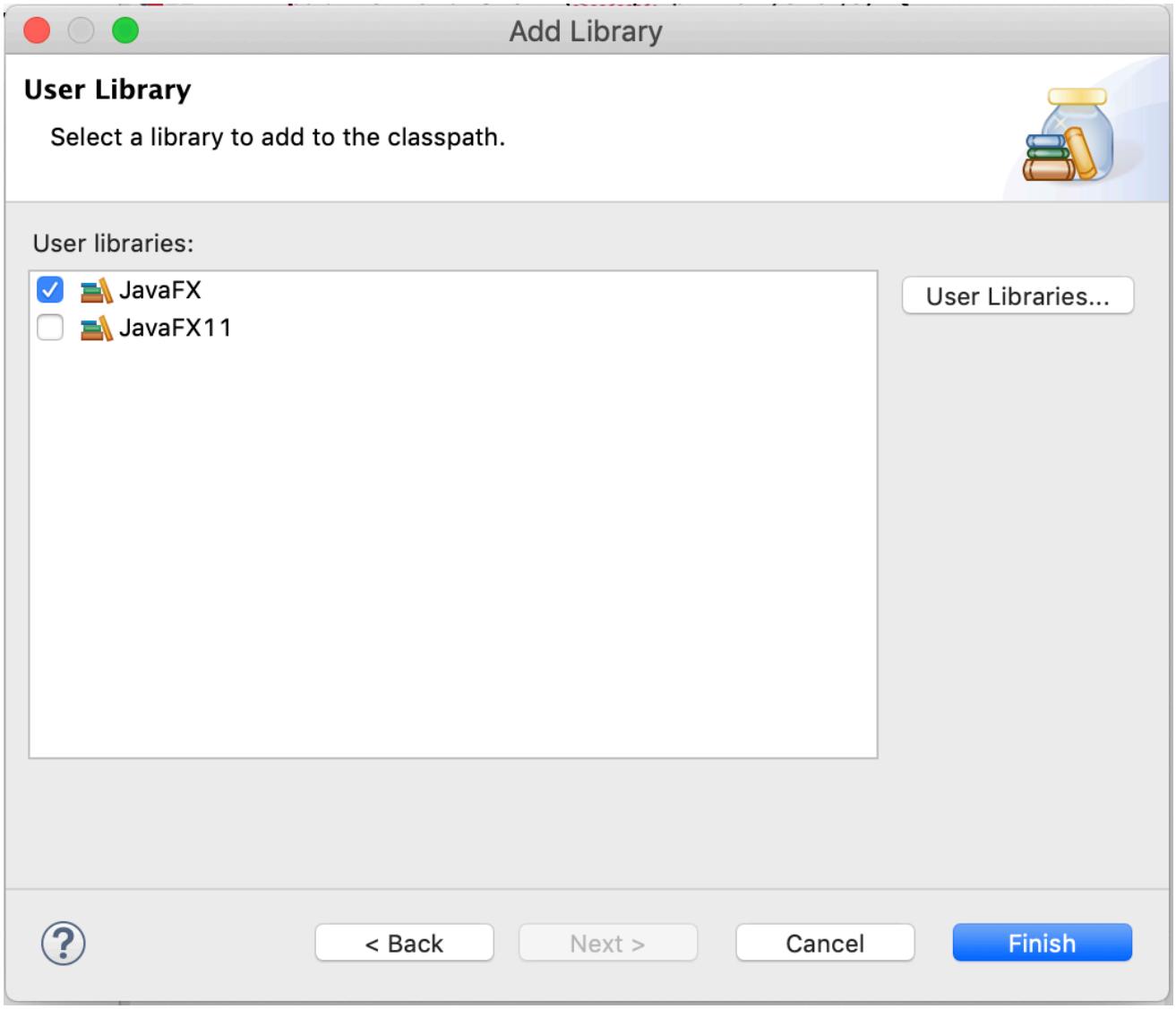
To fix this, right click your project and navigate to build path, and click `Add Libraries`:


eclipse-workspace - SomeProjectName/src/application/Main.java - Eclipse IDE
Quick Access
Package Explorer
Main.java
Main.java
Task List
Find
All
Outline
application
Main
start(Stage) : voi
main(String[]) : v
Problems
Javadoc
Declaration
Console
<terminated> Main (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk-13.0.1.jdk/Contents/Home/bin/java (N
1 package application;
2
vafx.application.Application;
ass Main extends Application {
ride
c void start(Stage primaryStage) {
ry {
BorderPane root = new BorderPane();
Scene scene = new Scene(root,400,400);
scene.getStylesheets().add(getClass().getResc
primaryStage.setScene(scene);
primaryStage.show();
catch(Exception e) {
e.printStackTrace();
gs) {
New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In ⌥⌘W
Copy ⌘C
Copy Qualified Name
Paste ⌘V
Delete
Remove from Context ⌥⇧⌘↓
Build Path
Source ⌥⌘S
Refactor ⌥⌘T
Import...
Export...
Refresh F5
Close Project
Close Unrelated Project
Assign Working Sets...
Coverage As
Run As
Debug As
Restore from Local History...
Team
Compare With
Configure
Validate
Properties ⌘I
Link Source...
New Source Folder...
Use as Source Folder
Add External Archives...
Add Libraries...
Configure Build Path...

After clicking into add libraries, click user library:



Now select the desired JavaFX library:

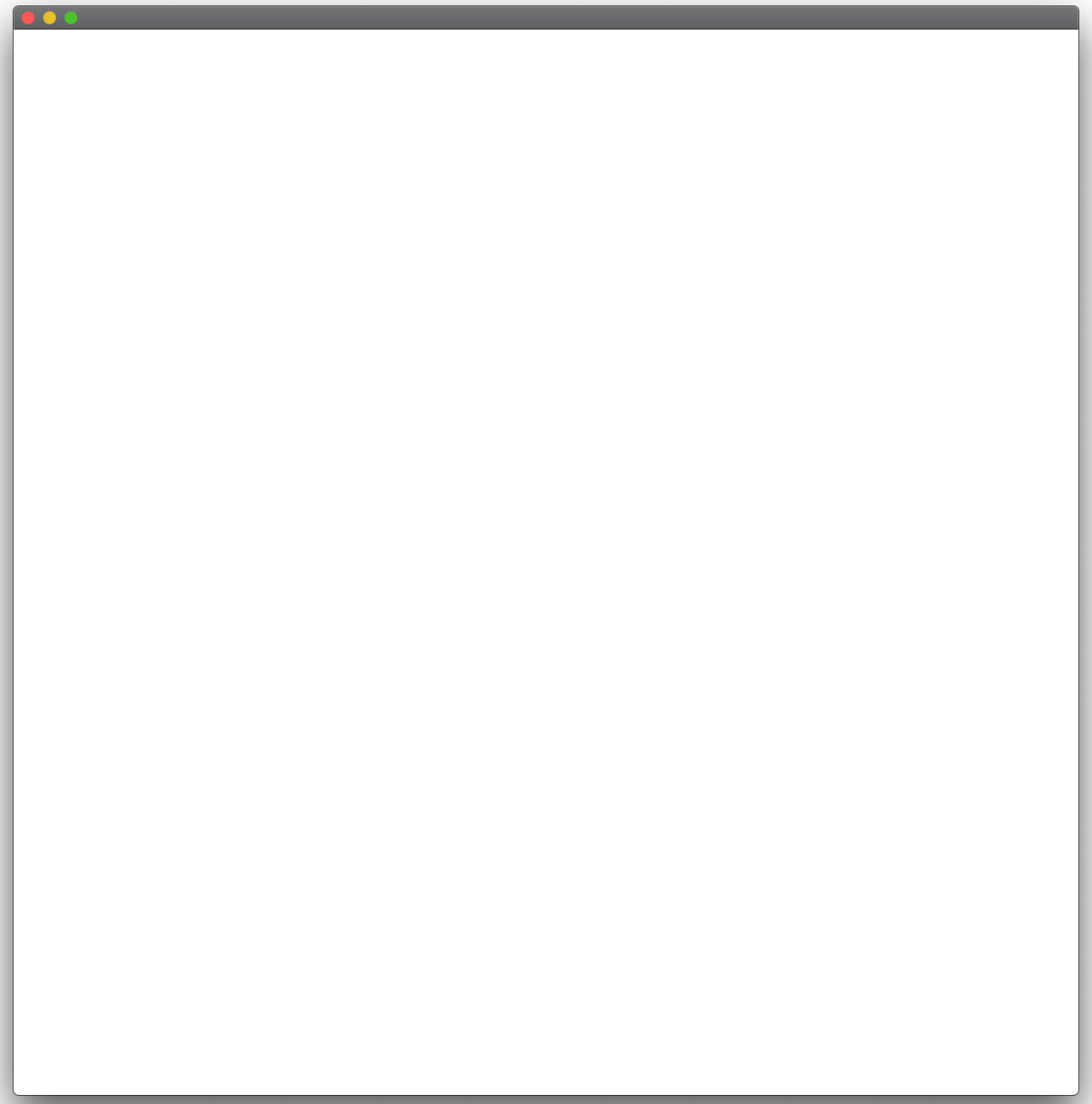


Now your app ought to be able to run! Make sure that your code looks like the below code with no errors:

```
1  package application;
2
3  import javafx.application.Application;
4  import javafx.stage.Stage;
5  import javafx.scene.Scene;
6  import javafx.scene.layout.BorderPane;
7
8
9  public class Main extends Application {
10   @Override
11   public void start(Stage primaryStage) {
12     try {
13       BorderPane root = new BorderPane();
```

```
      Scene scene = new Scene(root,1000,1000);

scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
      primaryStage.setScene(scene);
      primaryStage.show();

    } catch(Exception e) {
      e.printStackTrace();
    }
  }

  public static void main(String[] args) {
    launch(args);
  }
}

```

Now click the run button and you should see a blank pane (very exciting):



Writing the code!

Let's first start out by cleaning up our `start` method and making our pane a bit smaller by changing the size of our window to 500 by 500:

```
1  package application;
2
3  import javafx.application.Application;
4  import javafx.stage.Stage;
5  import javafx.scene.Scene;
6  import javafx.scene.layout.Pane;
7
```

```
8
9  public class Main extends Application {
10
11   public Scene buildPrimaryStageScene() {
12     Pane root = new Pane();
13     Scene scene = new Scene(root,500,500);
14
   scene.getStylesheets().add(getClass().getResource("application.css").toExtern
   alForm());
15
16     return scene;
17   }
18
19
20   @Override
21   public void start(Stage primaryStage) {
22     try {
23       Scene myScene = buildPrimaryStageScene();
24
25
26       primaryStage.setScene(myScene);
27       primaryStage.show();
28
29     } catch(Exception e) {
30       e.printStackTrace();
31     }
32   }
33
34   public static void main(String[] args) {
35     launch(args);
36   }
37 }
38
```

Tic Tac Toe has a three by three grid of `x`s and `o`s, so let's create each individual grid by creating a private class called Tile:

```
1  package application;
2
3  import javafx.application.Application;
4  import javafx.geometry.Pos;
5  import javafx.stage.Stage;
6  import javafx.scene.Scene;
7  import javafx.scene.layout.Pane;
```

```
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;


public class Main extends Application {

  public Scene buildPrimaryStageScene() {
    Pane root = new Pane();
    Scene scene = new Scene(root,600,600);

scene.getStylesheets().add(getClass().getResource("application.css").toExtern
alForm());

    return scene;
  }


  private class Tile extends StackPane {
    public Tile() {
      Rectangle border = new Rectangle(200,200);
      border.setFill(null);
      border.setStroke(Color.BLACK);

      setAlignment(Pos.CENTER);
      getChildren().addAll(border);
    }

  }

  @Override
  public void start(Stage primaryStage) {
    try {
      Scene myScene = buildPrimaryStageScene();

      primaryStage.setScene(myScene);
      primaryStage.show();

    } catch(Exception e) {
      e.printStackTrace();
    }
  }

  public static void main(String[] args) {
    launch(args);
  }
```

```
}

```

Now let's add our tiles to our border!

```
package application;

import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;


public class Main extends Application {

  public Scene buildPrimaryStageScene() {
    Pane root = new Pane();
    Scene scene = new Scene(root,600,600);

scene.getStylesheets().add(getClass().getResource("application.css").toExtern
alForm());

    // Create new tiles within the scene!
    for (int i = 0 ; i < 3; i++) {
      for (int j = 0 ; j < 3; j++ ) {
        Tile tile = new Tile();
        tile.setTranslateX(j*200);
        tile.setTranslateY(i*200);

        root.getChildren().add(tile);
      }
    }

    return scene;
  }


  private class Tile extends StackPane {
```

```
    public Tile() {
      Rectangle border = new Rectangle(200,200);
      border.setFill(null);
      border.setStroke(Color.BLACK);

      setAlignment(Pos.CENTER);
      getChildren().addAll(border);
    }
  }

  @Override
  public void start(Stage primaryStage) {
    try {
      Scene myScene = buildPrimaryStageScene();

      primaryStage.setScene(myScene);
      primaryStage.show();

    } catch(Exception e) {
      e.printStackTrace();
    }
  }

  public static void main(String[] args) {
    launch(args);
  }
}
```

Now, when a user clicks a specific tile, that tile ought to show either `x` or `o`. To do that, we'll need to add text to our tiles:

```
package application;

import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.input.MouseButton;
import javafx.scene.layout.Pane;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Text;

```

```
14
15  public class Main extends Application {
16
17    public Scene buildPrimaryStageScene() {
18      Pane root = new Pane();
19      Scene scene = new Scene(root,600,600);
20
    scene.getStylesheets().add(getClass().getResource("application.css").toExtern
    alForm());
21
22      // Create new tiles within the scene!
23      for (int i = 0 ; i < 3; i++) {
24        for (int j = 0 ; j < 3; j++ ) {
25          Tile tile = new Tile();
26          tile.setTranslateX(j*200);
27          tile.setTranslateY(i*200);
28
29          root.getChildren().add(tile);
30        }
31      }
32
33      return scene;
34    }
35
36
37    class Tile extends StackPane {
38      Text text = new Text();
39
40      public Tile() {
41        Rectangle border = new Rectangle(200,200);
42        border.setFill(null);
43        border.setStroke(Color.BLACK);
44
45        setAlignment(Pos.CENTER);
46        getChildren().addAll(border);
47
48      }
49
50      private void drawX() {
51        text.setText("X");
52      }
53
54      private void drawO() {
55        text.setText("O");
56      }
57    }
```

```
58
59    @Override
60    public void start(Stage primaryStage) {
61      try {
62        Scene myScene = buildPrimaryStageScene();
63
64        primaryStage.setScene(myScene);
65        primaryStage.show();
66
67      } catch(Exception e) {
68        e.printStackTrace();
69      }
70    }
71
72    public static void main(String[] args) {
73      launch(args);
74    }
75  }
```

We've added text to our tile, but there's nothing calling setText for `x` or `o`. In order to do that, we'll need to add in some event driven programming. To do that, we'll need to add a function to our tile constructor called `setOnMouseClicked` that takes in a function. In this function, we'll use a lambda to actually set our text by calling those methods:

```
1  package application;
2
3  import javafx.application.Application;
4  import javafx.geometry.Pos;
5  import javafx.stage.Stage;
6  import javafx.scene.Scene;
7  import javafx.scene.input.MouseButton;
8  import javafx.scene.layout.Pane;
9  import javafx.scene.layout.StackPane;
10 import javafx.scene.paint.Color;
11 import javafx.scene.shape.Rectangle;
12 import javafx.scene.text.Text;
13
14
15 public class Main extends Application {
16
17   public Scene buildPrimaryStageScene() {
18     Pane root = new Pane();
19     Scene scene = new Scene(root,600,600);
```

```
scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());

    // Create new tiles within the scene!
    for (int i = 0 ; i < 3; i++) {
      for (int j = 0 ; j < 3; j++ ) {
        Tile tile = new Tile();
        tile.setTranslateX(j*200);
        tile.setTranslateY(i*200);

        root.getChildren().add(tile);
      }
    }

    return scene;
  }


  class Tile extends StackPane {
    Text text = new Text();

    public Tile() {
      Rectangle border = new Rectangle(200,200);
      border.setFill(null);
      border.setStroke(Color.BLACK);

      setAlignment(Pos.CENTER);
      getChildren().addAll(border, text);

      setOnMouseClicked(eventClick -> {
        System.out.println("What is the event?" + eventClick);
        if (eventClick.getButton() == MouseButton.PRIMARY) {
          drawX();
        } else {
          drawO();
        }
      });
    }

    private void drawX() {
      text.setText("X");
    }

    private void drawO() {
      text.setText("O");
```

```
    }
  }

  @Override
  public void start(Stage primaryStage) {
    try {
      Scene myScene = buildPrimaryStageScene();

      primaryStage.setScene(myScene);
      primaryStage.show();

    } catch(Exception e) {
      e.printStackTrace();
    }
  }

  public static void main(String[] args) {
    launch(args);
  }
}
```

Clearly we need to increase the size of our `x`s and `o`s. Let's do that by using the `text.setFont` method inside of our tile constructor:

```
package application;

import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.input.MouseButton;
import javafx.scene.layout.Pane;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;
import javafx.scene.text.Text;


public class Main extends Application {

  public Scene buildPrimaryStageScene() {
    Pane root = new Pane();
    Scene scene = new Scene(root,600,600);
```

```
scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());

    // Create new tiles within the scene!
    for (int i = 0 ; i < 3; i++) {
      for (int j = 0 ; j < 3; j++ ) {
        Tile tile = new Tile();
        tile.setTranslateX(j*200);
        tile.setTranslateY(i*200);

        root.getChildren().add(tile);
      }
    }

    return scene;
  }


  class Tile extends StackPane {
    Text text = new Text();

    public Tile() {
      Rectangle border = new Rectangle(200,200);
      border.setFill(null);
      border.setStroke(Color.BLACK);

      text.setFont(Font.font(72));

      setAlignment(Pos.CENTER);
      getChildren().addAll(border, text);

      setOnMouseClicked(eventClick -> {
        System.out.println("What is the event?" + eventClick);
        if (eventClick.getButton() == MouseButton.PRIMARY) {
          drawX();
        } else {
          drawO();
        }
      });
    }

    private void drawX() {
      text.setText("X");
    }

```

```
    private void drawO() {
      text.setText("O");
    }
  }

  @Override
  public void start(Stage primaryStage) {
    try {
      Scene myScene = buildPrimaryStageScene();

      primaryStage.setScene(myScene);
      primaryStage.show();

    } catch(Exception e) {
      e.printStackTrace();
    }
  }

  public static void main(String[] args) {
    launch(args);
  }
}
```

You may have noticed, though, that you can literally click any single spot on the game board and change it to either `X` or `O`. So let's create a method that makes it so that you absolutely must change the turn of who's playing:

```
package application;

import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.input.MouseButton;
import javafx.scene.layout.Pane;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;
import javafx.scene.text.Text;


public class Main extends Application {

  private boolean turnX = true;
```

```
19
20    public Scene buildPrimaryStageScene() {
21      Pane root = new Pane();
22      Scene scene = new Scene(root,600,600);
23
scene.getStylesheets().add(getClass().getResource("application.css").toExtern
alForm());
24
25      // Create new tiles within the scene!
26      for (int i = 0 ; i < 3; i++) {
27        for (int j = 0 ; j < 3; j++ ) {
28          Tile tile = new Tile();
29          tile.setTranslateX(j*200);
30          tile.setTranslateY(i*200);
31
32          root.getChildren().add(tile);
33        }
34      }
35
36      return scene;
37    }
38
39
40    class Tile extends StackPane {
41      Text text = new Text();
42
43      public Tile() {
44        Rectangle border = new Rectangle(200,200);
45        border.setFill(null);
46        border.setStroke(Color.BLACK);
47
48        text.setFont(Font.font(72));
49
50        setAlignment(Pos.CENTER);
51        getChildren().addAll(border, text);
52
53        setOnMouseClicked(eventClick -> {
54          System.out.println("What is the event?" + eventClick);
55          if (eventClick.getButton() == MouseButton.PRIMARY) {
56            if(!turnX) {
57              return;
58            }
59
60            drawX();
61            turnX = false;
62          } else {
```

```
63            if(turnX) {
64              return;
65            }
66            drawO();
67            turnX = true;
68          }
69        });
70      }
71
72      private void drawX() {
73        text.setText("X");
74      }
75
76      private void drawO() {
77        text.setText("O");
78      }
79    }
80
81    @Override
82    public void start(Stage primaryStage) {
83      try {
84        Scene myScene = buildPrimaryStageScene();
85
86        primaryStage.setScene(myScene);
87        primaryStage.show();
88
89      } catch(Exception e) {
90        e.printStackTrace();
91      }
92    }
93
94    public static void main(String[] args) {
95      launch(args);
96    }
97  }
```

When we run the program, now we can't play multiple runs of `x` or `o`! We're not out of the woods yet, you can still overwrite spaces! So let's make sure that our spaces are playable before we set the text!

It seems though, now, that we need a way to determine whether the space is playable or not. A good way to see if a space is playable is to determine whether or not there's any text in the space itself, so let's write a getter function for the tile's text, and a function that returns a boolean if the tile already has text in it:

```
1  package application;
```

```
2
3  import javafx.application.Application;
4  import javafx.geometry.Pos;
5  import javafx.stage.Stage;
6  import javafx.scene.Scene;
7  import javafx.scene.input.MouseButton;
8  import javafx.scene.layout.Pane;
9  import javafx.scene.layout.StackPane;
10 import javafx.scene.paint.Color;
11 import javafx.scene.shape.Rectangle;
12 import javafx.scene.text.Font;
13 import javafx.scene.text.Text;
14
15
16 public class Main extends Application {
17   private boolean turnX = true;
18
19   public Scene buildPrimaryStageScene() {
20     Pane root = new Pane();
21     Scene scene = new Scene(root,600,600);
22
   scene.getStylesheets().add(getClass().getResource("application.css").toExter
   nalForm());
23
24     // Create new tiles within the scene!
25     for (int i = 0 ; i < 3; i++) {
26       for (int j = 0 ; j < 3; j++ ) {
27         Tile tile = new Tile();
28         tile.setTranslateX(j*200);
29         tile.setTranslateY(i*200);
30
31         root.getChildren().add(tile);
32       }
33     }
34
35     return scene;
36   }
37
38
39   class Tile extends StackPane {
40     Text text = new Text();
41
42     public Tile() {
43       Rectangle border = new Rectangle(200,200);
44       border.setFill(null);
45       border.setStroke(Color.BLACK);
```

```
46
47        text.setFont(Font.font(72));
48
49        setAlignment(Pos.CENTER);
50        getChildren().addAll(border, text);
51
52        setOnMouseClicked(eventClick -> {
53          if(!isPlayableSpace()) return;
54
55          System.out.println("What is the event?" + eventClick);
56          if (eventClick.getButton() == MouseButton.PRIMARY) {
57            if(!turnX) {
58              return;
59            }
60            drawX();
61            turnX = false;
62          } else {
63            if(turnX) {
64              return;
65            }
66            drawO();
67            turnX = true;
68          }
69        });
70      }
71
72      private void drawX() {
73        text.setText("X");
74      }
75
76      private void drawO() {
77        text.setText("O");
78      }
79
80      public String getTextValue() {
81        return text.getText();
82      }
83
84      public boolean isPlayableSpace() {
85        if( getTextValue().isEmpty() ) {
86          return true;
87        }
88        return false;
89      }
90    }
91
```

```
  @Override
  public void start(Stage primaryStage) {
    try {
      Scene myScene = buildPrimaryStageScene();

      primaryStage.setScene(myScene);
      primaryStage.show();

    } catch(Exception e) {
      e.printStackTrace();
    }
  }

  public static void main(String[] args) {
    launch(args);
  }
}
```

So now that we've build the basic game board, we need to find a way to determine who the winner is! However, before we do that, let's try refactoring our code first. Refactoring doesn't just include renaming things, it also includes moving bits of code around, rewriting functions to be more abstract, etc. So, let's start by moving our Tile class into it's own class outside of our main application file.

`Tile.java`

```
package application;

import javafx.geometry.Pos;
import javafx.scene.input.MouseButton;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;
import javafx.scene.text.Text;


public class Tile extends StackPane {
  public Tile() {
    Rectangle border = new Rectangle(200,200);
    border.setFill(null);
    border.setStroke(Color.BLACK);

    text.setFont(Font.font(72));

```

```
    setAlignment(Pos.CENTER);
    getChildren().addAll(border, text);

    setOnMouseClicked(eventClick -> {
      if(!isPlayableSpace()) return;

      System.out.println("What is the event?" + eventClick);
      if (eventClick.getButton() == MouseButton.PRIMARY) {
        if(!Turn.turnX) {
          return;
        }

        drawX();
        turnX = false;
      } else {
        if(turnX) {
          return;
        }
        drawO();
        turnX = true;
      }
    });
  }

  private void drawX() {
    text.setText("X");
  }

  private void drawO() {
    text.setText("O");
  }

  public String getTextValue() {
    return text.getText();
  }

  public boolean isPlayableSpace() {
    if( getTextValue().isEmpty() ) {
      return true;
    }
    return false;
  }
}
```

The above is great! Though we do seem to have run into a snag, since our turnX was an application level boolean value. How do we share something between a whole number of references? If we tried to place turnX within our Tile class we would be creating a new `turnX` boolean every single time a new class was instantiated. UNLESS we make it static! So let's do that:

`Tile.java`:

```
1  package application;
2
3  import javafx.geometry.Pos;
4  import javafx.scene.input.MouseButton;
5  import javafx.scene.layout.StackPane;
6  import javafx.scene.paint.Color;
7  import javafx.scene.shape.Rectangle;
8  import javafx.scene.text.Font;
9  import javafx.scene.text.Text;
10
11 public class Tile extends StackPane {
12   static boolean turnX = true;
13
14   Text text = new Text();
15
16   public Tile() {
17     Rectangle border = new Rectangle(200,200);
18     border.setFill(null);
19     border.setStroke(Color.BLACK);
20
21     text.setFont(Font.font(72));
22
23     setAlignment(Pos.CENTER);
24     getChildren().addAll(border, text);
25
26     setOnMouseClicked(eventClick -> {
27       if(!isPlayableSpace()) return;
28
29       System.out.println("What is the event?" + eventClick);
30       if (eventClick.getButton() == MouseButton.PRIMARY) {
31         if(!turnX) {
32           return;
33         }
34
35         drawX();
36         turnX = false;
37       } else {
38         if(turnX) {
39           return;
```

```
        }
        drawO();
        turnX = true;
      }
    });
  }

  private void drawX() {
    text.setText("X");
  }

  private void drawO() {
    text.setText("O");
  }

  public String getTextValue() {
    return text.getText();
  }

  public boolean isPlayableSpace() {
    if( getTextValue().isEmpty() ) {
      return true;
    }
    return false;
  }
}

```

Deleting the tile code out of our main function, it looks a lot cleaner:

`Main.java`:

```
package application;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;


public class Main extends Application {


  public Scene buildPrimaryStageScene() {
    Pane root = new Pane();
    Scene scene = new Scene(root,600,600);
```

```
15
    scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
16
17      // Create new tiles within the scene!
18      for (int i = 0 ; i < 3; i++) {
19        for (int j = 0 ; j < 3; j++ ) {
20          Tile tile = new Tile();
21          tile.setTranslateX(j*200);
22          tile.setTranslateY(i*200);
23
24          root.getChildren().add(tile);
25        }
26      }
27
28      return scene;
29    }
30
31    @Override
32    public void start(Stage primaryStage) {
33      try {
34        Scene myScene = buildPrimaryStageScene();
35
36        primaryStage.setScene(myScene);
37        primaryStage.show();
38
39      } catch(Exception e) {
40        e.printStackTrace();
41      }
42    }
43
44    public static void main(String[] args) {
45      launch(args);
46    }
47  }
```

Now that our code's a bit cleaner, let's write some code to determine if there are three spaces in a row that all have the same exact text! We could write a method to do this, but let's instead create a class to take in 3 cards as a parameter for its constructor, and then determine whether those three cards are all the same type!

`ThreeInARow.java`

```
1  package application;
```

```
2
3
4  class ThreeInARow {
5    private Tile[] tiles;
6
7    public ThreeInARow(Tile... tiles) {  // varargs - VARANEAT
8      this.tiles = tiles;
9    }
10
11   public boolean hasWinningCondition() {
12     if (tiles[0].getTextValue().isEmpty()) return false;
13 System.out.println(tiles[0].getTextValue() + " " + tiles[1].getTextValue() +
   " " + tiles[2].getTextValue());
14     boolean isWinner =
   tiles[0].getTextValue().equals(tiles[1].getTextValue()) &&
15         tiles[0].getTextValue().equals(tiles[2].getTextValue());
16
17     return isWinner;
18   }
19 }
20
```

In the above code, you'll notice a brand new construction that we haven't talked about before. That's the varargs operator. That operator essentially says that there will be some kind of list of objects coming in of some type (in this case, tile).

Now let's some code to determine a winner! There are multiple possible winning conditions. Let's write a class that has a list of all possible winning conditions AND a way to check for a winner! We'll do so by having an arrayList of `ThreeInARow` objects and have a method that checks if any of those `ThreeInARow` objects return true on `hasWinningCondition`.

`CheckWinningCondition.java`

```
1  package application;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  public class CheckWinningCondition {
7    public static List<ThreeInARow> threeInARowList = new
   ArrayList<ThreeInARow>();
8
9    public static void checkWinner() {
10     // Go through all possible combinations!
```

```
    for(ThreeInARow combination : threeInARowList ) {
      if(combination.hasWinningCondition()) {
        System.out.println("OH GOODNESS THERE'S A WINNER!");

        break;
      }
    }
  }
}
```

Now that we have a way to create these possible winning conditions, let's add all the possible winning conditions to our list in main! The first being all three possible rows, all three possible columns, and the two diagonals!

`Main.java`

```
package application;

import java.util.ArrayList;
import java.util.List;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;


public class Main extends Application {
  private Tile[][] board = new Tile[3][3];

  private boolean gameWinner = false; /// ???????

  public Scene buildPrimaryStageScene() {
    Pane root = new Pane();
    Scene scene = new Scene(root,600,600);

scene.getStylesheets().add(getClass().getResource("application.css").toExtern
alForm());

    // Create new tiles within the scene!
    for (int i = 0 ; i < 3; i++) {
      for (int j = 0 ; j < 3; j++ ) {
        Tile tile = new Tile();
        tile.setTranslateX(j*200);
```

```
        tile.setTranslateY(i*200);

        root.getChildren().add(tile);

        board[j][i] = tile;  // this way we assign an element to the array's value.
      }
    }


    // for all horizontal rows:
    for (int y = 0 ; y < 3; y++) {
      ThreeInARow threeInARowObject = new ThreeInARow( board[0][y], board[1][y], board[2][y]);
      CheckWinningCondition.threeInARowList.add(threeInARowObject);
    }

    // for all vertical columns:
    for (int x = 0 ; x < 3; x++) {
      ThreeInARow threeInARowObject = new ThreeInARow( board[x][0], board[x][1], board[x][2]);
      CheckWinningCondition.threeInARowList.add(threeInARowObject);
    }

    // for the diagonals:
    ThreeInARow diagonalThreeInARow = new ThreeInARow( board[0][0], board[1][1], board[2][2]);
    CheckWinningCondition.threeInARowList.add(diagonalThreeInARow);


    ThreeInARow diagonalThreeInARowOpposite = new ThreeInARow( board[2][0], board[1][1], board[0][2]);
    CheckWinningCondition.threeInARowList.add(diagonalThreeInARow);
    return scene;
  }



  @Override
  public void start(Stage primaryStage) {
    try {
      Scene myScene = buildPrimaryStageScene();

      primaryStage.setScene(myScene);
      primaryStage.show();

```

```
    } catch(Exception e) {
      e.printStackTrace();
    }
  }

  public static void main(String[] args) {
    launch(args);
  }
}
```

If we were tor run the program now, we'd wind up having some difficulty, finding out who our winner is because we need to do one more step... and that's calling the `checkWinner` method! You may have wondered why we made the check winning condition and the list static, and that's because we want only a single instance of it so that we can access it easily across all classes!

`Tile.java`

```
package application;

import javafx.geometry.Pos;
import javafx.scene.input.MouseButton;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;
import javafx.scene.text.Text;

public class Tile extends StackPane {
  static boolean turnX = true;

  Text text = new Text();

  public Tile() {
    Rectangle border = new Rectangle(200,200);
    border.setFill(null);
    border.setStroke(Color.BLACK);

    text.setFont(Font.font(72));

    setAlignment(Pos.CENTER);
    getChildren().addAll(border, text);

    setOnMouseClicked(eventClick -> {
      if(!isPlayableSpace()) return;
```

```
28
29        System.out.println("What is the event?" + eventClick);
30        if (eventClick.getButton() == MouseButton.PRIMARY) {
31          if(!turnX) {
32            return;
33          }
34
35          drawX();
36          turnX = false;
37          CheckWinningCondition.checkWinner();
38
39        } else {
40          if(turnX) {
41            return;
42          }
43          drawO();
44          turnX = true;
45          CheckWinningCondition.checkWinner();
46        }
47      });
48    }
49
50    private void drawX() {
51      text.setText("X");
52    }
53
54    private void drawO() {
55      text.setText("O");
56    }
57
58    public String getTextValue() {
59      return text.getText();
60    }
61
62    public boolean isPlayableSpace() {
63      if( getTextValue().isEmpty() ) {
64        return true;
65      }
66      return false;
67    }
68  }
```

So now when we run the program, you should try to get a winning condition! If the game has one (that is, if you get three in a row), "OH GOODNESS THERE'S A WINNER" will be printed to our console! However, the game doesn't seem to stop at all, so let's add one more static boolean so that we can see if the game's won across all of our classes and set it to true when we reach our winning condition.

`CheckWinningCondition.java`

```
1  package application;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  public class CheckWinningCondition {
7    public static List<ThreeInARow> threeInARowList = new
   ArrayList<ThreeInARow>();
8    public static boolean gameIsWon = false;
9
10   public static void checkWinner() {
11     // Go through all possible combinations!
12     for(ThreeInARow combination : threeInARowList ) {
13       if(combination.hasWinningCondition()) {
14         System.out.println("OH GOODNESS THERE'S A WINNER!");
15         gameIsWon = true;
16         break;
17       }
18     }
19   }
20 }
```

To make sure we stop the game, we'll need to access this boolean in our Tile class and make sure we don't allow for `setOnMouseClicked` to run if there has been a winner. We've already done so for `isPlayableSpace`, so let's just add our boolean check into that:

`Tile.java`

```
1  package application;
2
3  import javafx.geometry.Pos;
4  import javafx.scene.input.MouseButton;
5  import javafx.scene.layout.StackPane;
6  import javafx.scene.paint.Color;
7  import javafx.scene.shape.Rectangle;
8  import javafx.scene.text.Font;
9  import javafx.scene.text.Text;
```

```
10
11  public class Tile extends StackPane {
12    static boolean turnX = true;
13
14    Text text = new Text();
15
16    public Tile() {
17      Rectangle border = new Rectangle(200,200);
18      border.setFill(null);
19      border.setStroke(Color.BLACK);
20
21      text.setFont(Font.font(72));
22
23      setAlignment(Pos.CENTER);
24      getChildren().addAll(border, text);
25
26      setOnMouseClicked(eventClick -> {
27        if(!isPlayableSpace() || CheckWinningCondition.gameIsWon) return;
28
29        System.out.println("What is the event?" + eventClick);
30        if (eventClick.getButton() == MouseButton.PRIMARY) {
31          if(!turnX) {
32            return;
33          }
34
35          drawX();
36          turnX = false;
37          CheckWinningCondition.checkWinner();
38
39        } else {
40          if(turnX) {
41            return;
42          }
43          drawO();
44          turnX = true;
45          CheckWinningCondition.checkWinner();
46        }
47      });
48    }
49
50    private void drawX() {
51      text.setText("X");
52    }
53
54    private void drawO() {
55      text.setText("O");
```

```
  }

  public String getTextValue() {
    return text.getText();
  }

  public boolean isPlayableSpace() {
    if( getTextValue().isEmpty() ) {
      return true;
    }
    return false;
  }
}
```

Before we move to our animation, let's refactor our main class a little bit just to make it a bit more clean:

```
package application;

import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.input.MouseButton;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;
import javafx.scene.text.Text;


public class Main extends Application {
  private Tile[][] board = new Tile[3][3];



  public void addTilesToTIARList() {

    for (int i = 0 ; i < 3; i++ ) {
      ThreeInARow threeInARowObject = new ThreeInARow(
        board[0][i],
        board[1][i],
```

```
        board[2][i]
      );

      ThreeInARow threeInARowObjectColumn = new ThreeInARow(
        board[i][0],
        board[i][1],
        board[i][2]
      );

      CheckWinningCondition.threeInARowList.add(threeInARowObjectColumn);
      CheckWinningCondition.threeInARowList.add(threeInARowObject);
    }

    ThreeInARow diagonalThreeInARow = new ThreeInARow(
        board[0][0],
        board[1][1],
        board[2][2]
    );

    ThreeInARow reverseDiagonalThreeIAR = new ThreeInARow(
        board[0][2],
        board[1][1],
        board[2][0]
    );

    CheckWinningCondition.threeInARowList.add(diagonalThreeInARow);
    CheckWinningCondition.threeInARowList.add(reverseDiagonalThreeIAR);
  }

  public Scene buildPrimaryStageScene() {
    Pane root = new Pane();
    Scene scene = new Scene(root, 600, 600);

scene.getStylesheets().add(getClass().getResource("application.css").toExtern
alForm());

    for(int i = 0 ; i < 3; i++) {
      for(int j = 0 ; j < 3; j++){
        Tile tile = new Tile();
        tile.setTranslateX(j*200);
        tile.setTranslateY(i*200);

        root.getChildren().add(tile);

        board[j][i] = tile;
      }
```

```
    }

    addTilesToTIARList();
    return scene;
  }


  @Override
  public void start(Stage primaryStage) {
    try {
      Scene scene = buildPrimaryStageScene();

      primaryStage.setScene(scene);
      primaryStage.show();
    } catch(Exception e) {
      e.printStackTrace();
    }
  }

  public static void main(String[] args) {
    launch(args);
  }
}

```

Adding Animations:

Fun story, nobody will be looking at the console to see if there's a winner, and if the game just stops, they might not know there's a winner, so we'll need to add one more step. When someone wins the game, we'll need to add an animation to show that we have a winning condition! For the sake of easiness, let's just make it a big line through the winning tiles!

To do this, we'll need to create an animation that goes through all three points in our tiles. First, though, we'll need to get the center of each of our tiles so that we'll have a starting and ending point for our lines:

`Tile.java`:

```
package application;

import javafx.geometry.Pos;
import javafx.scene.input.MouseButton;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
```

```
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;
import javafx.scene.text.Text;


class Tile extends StackPane {
  static boolean turnX = true;
  Text text = new Text();

  public Tile() {
    Rectangle border = new Rectangle(200,200);
    border.setFill(null);
    border.setStroke(Color.BLACK);
    text.setFont(Font.font(120));


    setAlignment(Pos.CENTER);
    getChildren().addAll(border, text);

    setOnMouseClicked(eventClick -> {
      if( !isPlayableSpace() || CheckWinningCondition.gameIsWon ) return;

      System.out.println("eventClick?" + eventClick);
      if(eventClick.getButton() == MouseButton.PRIMARY) {
        if (! turnX ) return;

        drawX();
        CheckWinningCondition.checkWinner();
        turnX = false;
      } else {
        if ( turnX ) return;

        drawO();
        CheckWinningCondition.checkWinner();
        turnX = true;
      }

    });
  }


  private void drawX() {
    text.setText("X");
  }

  private void drawO() {
```

```
    text.setText("O");
  }

  public String getTextValue() {
    return text.getText();
  }

  public boolean isPlayableSpace() {
    if( getTextValue().isEmpty() ) {
      return true;
    }

    return false;
  }

  public double getCenterX() {
    return getTranslateX() + (200 / 2);
  }

  public double getCenterY() {
    return getTranslateY() + (200 / 2);
  }
}
```

Now that we have our center values, we can use those to create our line! Though because we're doing an animation, we'll want to have both the start and the end of our line be on the very first tile's center values. Then, after establishing that our line exists, we'll create a couple of Key values with our end X and end Y coordinates from our tile. We then want to put those key values into a key frame with a duration of how long we want the animation to last. Finally, we'll add that key frame to our timeline, and then play the animation!

Before that will actually work though, we'll need to add our line to the child nodes of our root pane! So, we'll create a function to do that as well!

`CheckWinningCondtition.java`

```
package application;

import java.util.ArrayList;
import java.util.List;

import javafx.animation.Interpolator;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
```

```
import javafx.animation.Timeline;
import javafx.scene.layout.Pane;
import javafx.scene.shape.Line;
import javafx.util.Duration;

public class CheckWinningCondition {
  public static List<ThreeInARow> threeInARowList = new
ArrayList<ThreeInARow>();
  public static boolean gameIsWon = false;
  public static Line winningLine = new Line();


  public static void checkWinner() {
    for (ThreeInARow combination : threeInARowList) {

      if(combination.hasWinningCondition()){
        System.out.println("OH GOODNESS! THERE'S A WINNER!");

        gameIsWon = true;
        playWinningAnimation(combination);
        break;
      }
    }
  }

  private static void playWinningAnimation(ThreeInARow threeInARow) {
    for (Tile tile : threeInARow.tiles) {
      double xCoords = tile.getCenterX();
      double yCoords = tile.getCenterY();
      System.out.println( " X : " + xCoords + " Y: " + yCoords);
    }

    winningLine.setStartX(threeInARow.tiles[0].getCenterX());
    winningLine.setStartY(threeInARow.tiles[0].getCenterY());
    winningLine.setEndX(threeInARow.tiles[0].getCenterX());
    winningLine.setEndY(threeInARow.tiles[0].getCenterY());
    winningLine.setStrokeWidth(15);


    KeyValue endX = new KeyValue(winningLine.endXProperty(),
threeInARow.tiles[2].getCenterX());
        KeyValue endY = new KeyValue(winningLine.endYProperty(),
threeInARow.tiles[2].getCenterY());

        KeyFrame timeLineKeyFrameAnimation = new
KeyFrame(Duration.seconds(3), endX, endY);
```

```
51
52          Timeline timeline =  new Timeline( timeLineKeyFrameAnimation);
53          timeline.play();
54    }
55
56
57    public static void addLinesToRoot(Pane rootPane) {
58      rootPane.getChildren().add(winningLine);
59    }
60  }
61
```

And then finally, we'll want to add it to our (very finely refactored) Main class:

`Main.java`:

```
1  package application;
2
3  import javafx.application.Application;
4  import javafx.geometry.Pos;
5  import javafx.stage.Stage;
6  import javafx.scene.Scene;
7  import javafx.scene.input.MouseButton;
8  import javafx.scene.layout.BorderPane;
9  import javafx.scene.layout.Pane;
10 import javafx.scene.layout.StackPane;
11 import javafx.scene.paint.Color;
12 import javafx.scene.shape.Rectangle;
13 import javafx.scene.text.Font;
14 import javafx.scene.text.Text;
15
16
17 public class Main extends Application {
18   private Tile[][] board = new Tile[3][3];
19
20
21
22   public void addTilesToTIARList() {
23
24     for (int i = 0 ; i < 3; i++ ) {
25       ThreeInARow threeInARowObject = new ThreeInARow(
26         board[0][i],
27         board[1][i],
28         board[2][i]
```

```
      );

      ThreeInARow threeInARowObjectColumn = new ThreeInARow(
        board[i][0],
        board[i][1],
        board[i][2]
      );

      CheckWinningCondition.threeInARowList.add(threeInARowObjectColumn);
      CheckWinningCondition.threeInARowList.add(threeInARowObject);
    }

    ThreeInARow diagonalThreeInARow = new ThreeInARow(
        board[0][0],
        board[1][1],
        board[2][2]
    );

    ThreeInARow reverseDiagonalThreeIAR = new ThreeInARow(
        board[0][2],
        board[1][1],
        board[2][0]
    );

    CheckWinningCondition.threeInARowList.add(diagonalThreeInARow);
    CheckWinningCondition.threeInARowList.add(reverseDiagonalThreeIAR);
  }




  public Scene buildPrimaryStageScene() {
    Pane root = new Pane();
    Scene scene = new Scene(root, 600, 600);

scene.getStylesheets().add(getClass().getResource("application.css").toExter
nalForm());



    for(int i = 0 ; i < 3; i++) {
      for(int j = 0 ; j < 3; j++){
        Tile tile = new Tile();
        tile.setTranslateX(j*200);
        tile.setTranslateY(i*200);

```

```
        root.getChildren().add(tile);

        board[j][i] = tile;
      }
    }

    addTilesToTIARList();
    CheckWinningCondition.addLinesToRoot(root);
    return scene;
  }


  @Override
  public void start(Stage primaryStage) {
    try {
      Scene scene = buildPrimaryStageScene();

      primaryStage.setScene(scene);
      primaryStage.show();
    } catch(Exception e) {
      e.printStackTrace();
    }
  }

  public static void main(String[] args) {
    launch(args);
  }
}
```