Tech Workshop January 2024

Java App Development

Summary

Introduction

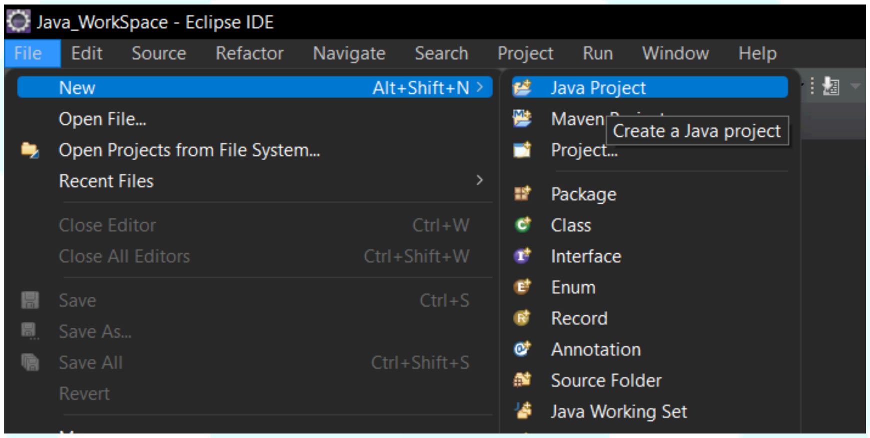
This tech workshop should give you a jump start on Java App Development. We will cover the basics of setting up a Java application and how to build logic in order to achieve successful results. We hope you'll follow along and create your own application as you begin your journey into Java programming.

Initial Set Up

We recommend using Eclipse as your IDE if you'd like to code along with this workshop. Steps shown by the instructor will be in Eclipse and running the project will require installing the Java Development Kit (JDK) from Oracle. For more detail here, feel free to consult the Java Bridge Course.

Starting a Project

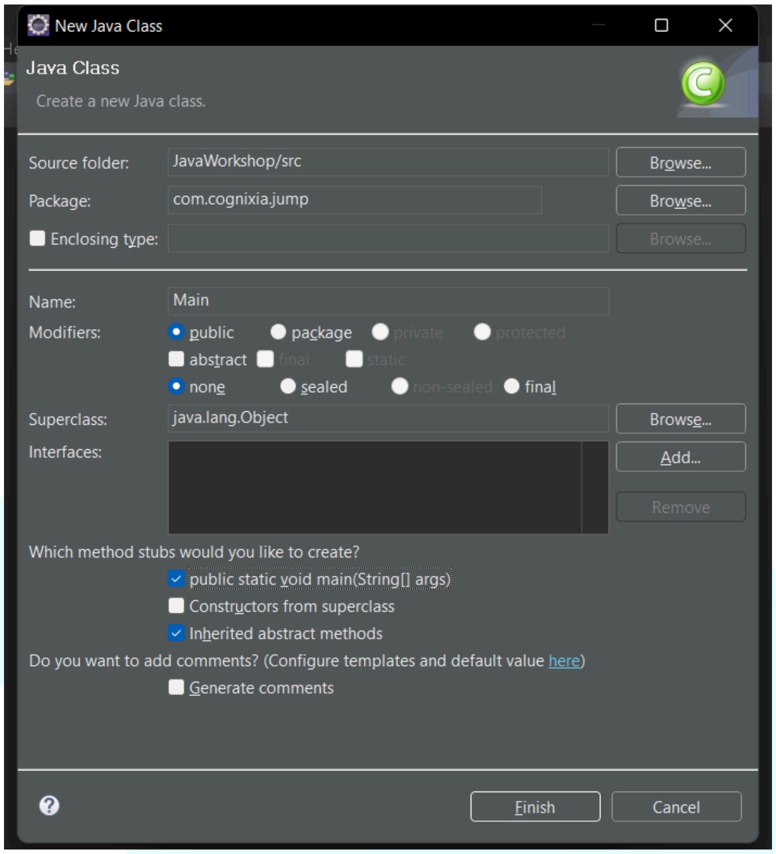
Once in Eclipse, we'll click **File** > **New** > **Java Project**



Now choose a name for your project, such as **JavaWorkshop**, make sure the box for **module-info.java** is unchecked.

Right-click on the **src** folder and select **New** > **Package**. We'll type **com.cognixia.workshop** for the name of our package.

Now we can right click on our package and select **New** > **Class**. We'll name our first class **Main** and check the box for **public static void main(String[] args).**



Source folder: JavaWorkshop/src

Package: com.cognixia.jump

Enclosing type: [ ]

Name: Main

Modifiers: [✓ public] package private protected

[ ] abstract [ ] final [ ] static

[✓ none] sealed non-sealed final

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

[✓] public static void main(String[] args)

[ ] Constructors from superclass

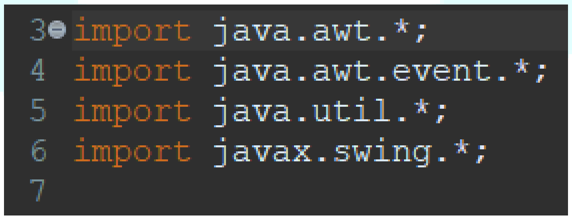
[✓] Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

[ ] Generate comments

Next we'll create our TicTacToe class without checking the box for **public static void main(String[] args).**

Make sure to include the imports that we'll be using for this project. The * at the end will include everything from the specified package.



```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
```

GUI Elements

For this project we'll be using **Java Swing** for our **Graphic User Interface (GUI)** since our use case is a standalone app that runs on our local machines. In the future you may encounter use cases such as web applications that leverage **React** or **Angular** with **JavaScript** for user interface. The scope of this workshop will be Java development only.

Within our **TicTacToe** class, let's begin by initializing elements needed by our GUI.

```
// GUI Objects
JFrame frame = new JFrame();
JPanel titlePanel = new JPanel();
JPanel buttonPanel = new JPanel();
JLabel textField = new JLabel();
JButton[] buttons = new JButton[9];

// Used to randomly select the first player
Random random = new Random();

// don't need more than one boolean, this is enough to determine player 2's turn
boolean player1turn;
```

We'll need to do some configuration of these elements within our TicTacToe constructor. Here's an example of how to begin.

```
TicTacToe(){

     // Setting values for GUI display
     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     frame.setSize(1000, 800);;
     frame.getContentPane().setBackground(new Color(50, 50, 50));
```

An important element of our GUI will be our button grid representing the game board, we can construct a for loop to expedite this process. Once we've added the buttons to our frame, we'll call the firstTurn method, which brings us to our application logic.

```java
for(int i=0; i<9; i++) {
    buttons[i] = new JButton();
    buttonPanel.add(buttons[i]);
    buttons[i].setFont(new Font("Tahoma",Font.BOLD,120));
    buttons[i].setFocusable(false); // can't select it by pressing tab to move the focus
    buttons[i].addActionListener(this); // comment this out to fix a bug at the end
}

frame.add(buttonPanel);

firstTurn();
```

Application Logic

The key to any successful and working application is the logic within it. Sometimes called business logic or game logic, these instructions will be how the program understands our goals.

Here’s an example of how we could begin with our **firstTurn** logic.

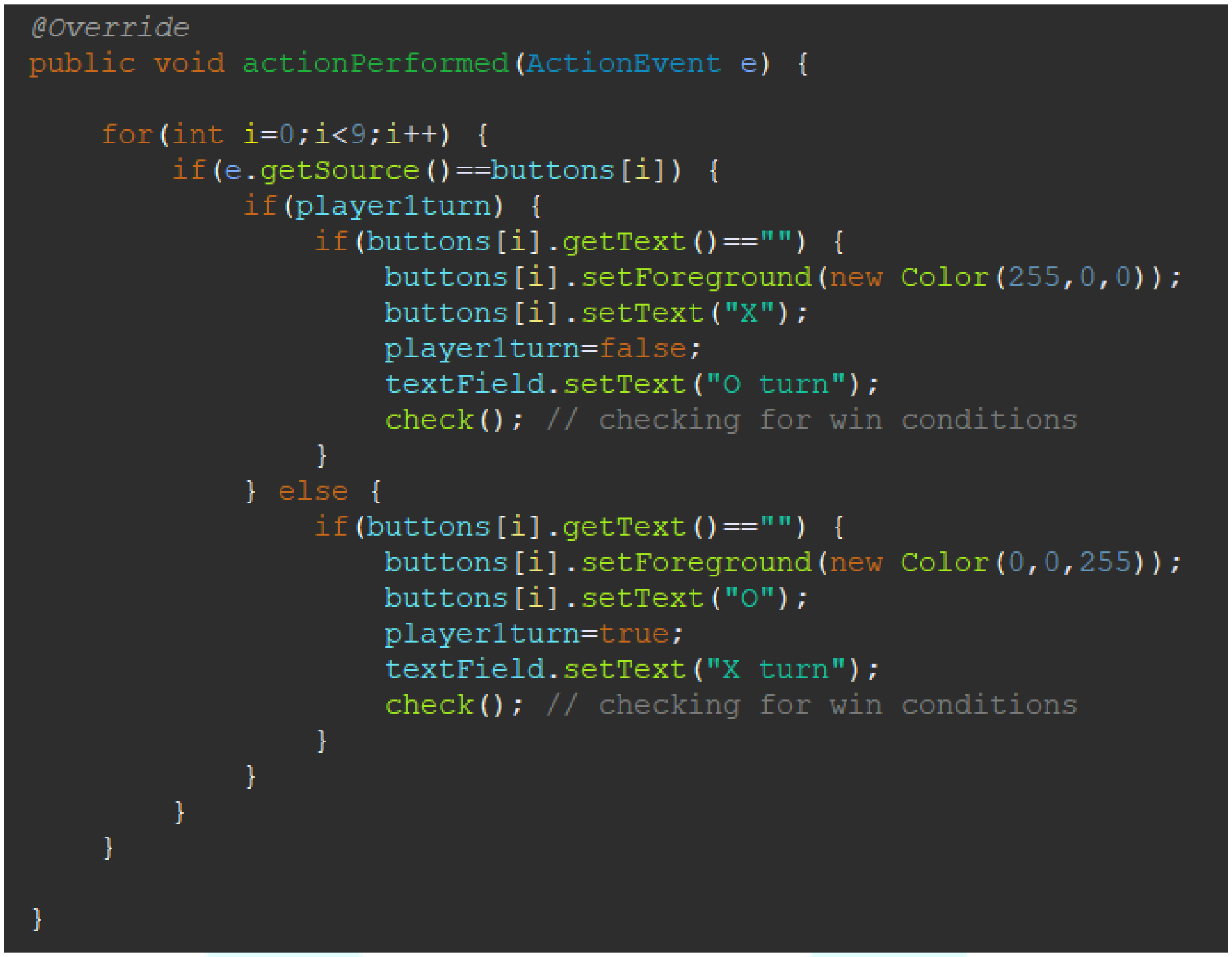
```java
public void firstTurn() {

    if(random.nextInt(2)==0) {
        player1turn=true;
        textField.setText("X turn");
    } else {
        player1turn=false;
        textField.setText("O turn");
    }

}
```

Depending on the random result, the text field will display the player’s turn and set the boolean value accordingly.

For our GUI we need to make sure we implement **ActionListener** in our **TicTacToe** class. This will help the application look for button presses and react to them.

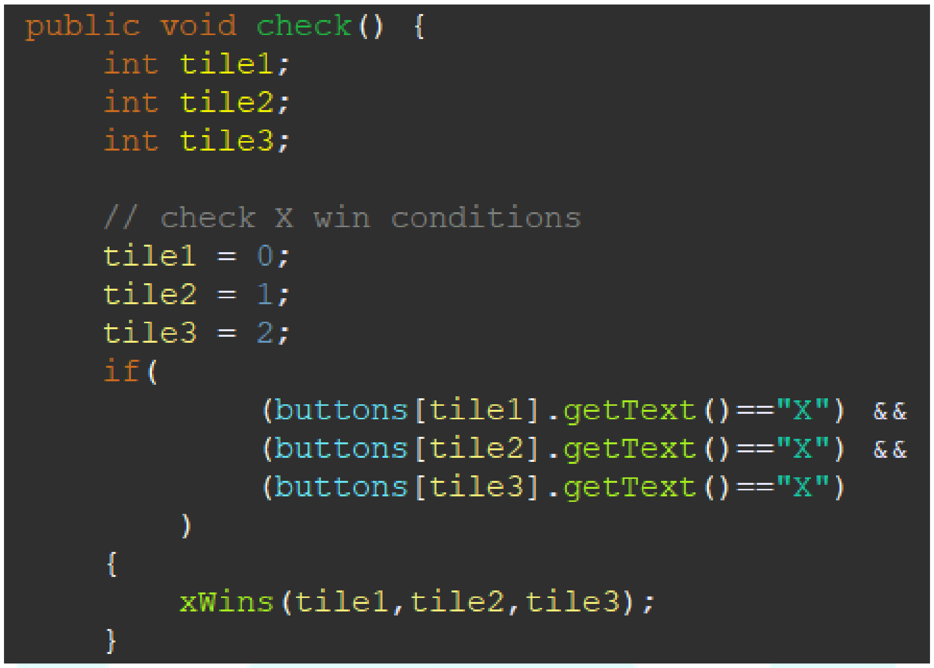
```java
public class TicTacToe implements ActionListener {
```

Since we've implemented this interface, we need to implement the **actionPerformed** method. Here's an example of how we could instruct the application when a button is pressed.

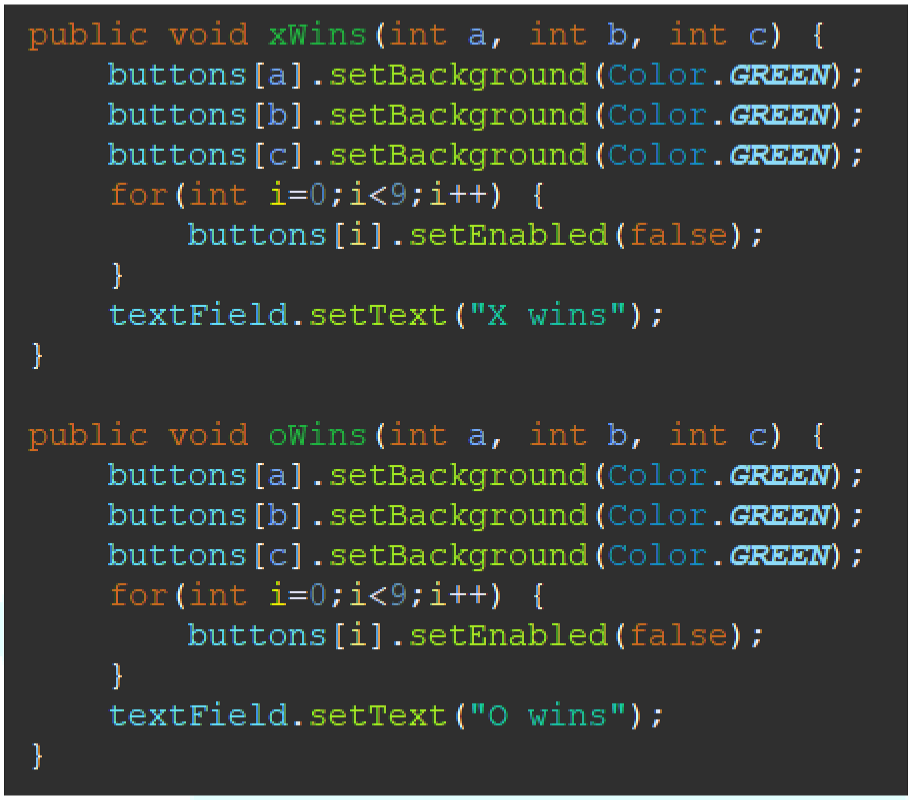


```java
@Override
public void actionPerformed(ActionEvent e) {

    for(int i=0;i<9;i++) {
        if(e.getSource()==buttons[i]) {
            if(player1turn) {
                if(buttons[i].getText()=="") {
                    buttons[i].setForeground(new Color(255,0,0));
                    buttons[i].setText("X");
                    player1turn=false;
                    textField.setText("O turn");
                    check(); // checking for win conditions
                }
            } else {
                if(buttons[i].getText()=="") {
                    buttons[i].setForeground(new Color(0,0,255));
                    buttons[i].setText("O");
                    player1turn=true;
                    textField.setText("X turn");
                    check(); // checking for win conditions
                }
            }
        }
    }

}
```

From here we need our check function to look for winning combinations after a button is pressed. Since there are a number of possible solutions here is just one example.


```
public void check() {
    int tile1;
    int tile2;
    int tile3;

    // check X win conditions
    tile1 = 0;
    tile2 = 1;
    tile3 = 2;
    if(
            (buttons[tile1].getText()=="X")  &&
            (buttons[tile2].getText()=="X")  &&
            (buttons[tile3].getText()=="X")
        )
    {
        xWins(tile1,tile2,tile3);
    }
```

You may note that the tile variables are not hard coded into the if statement and **xWins** function. This will make this code snippet easier to reuse. We will also need an **oWins** function for O player's win conditions although both functions will be very similar.



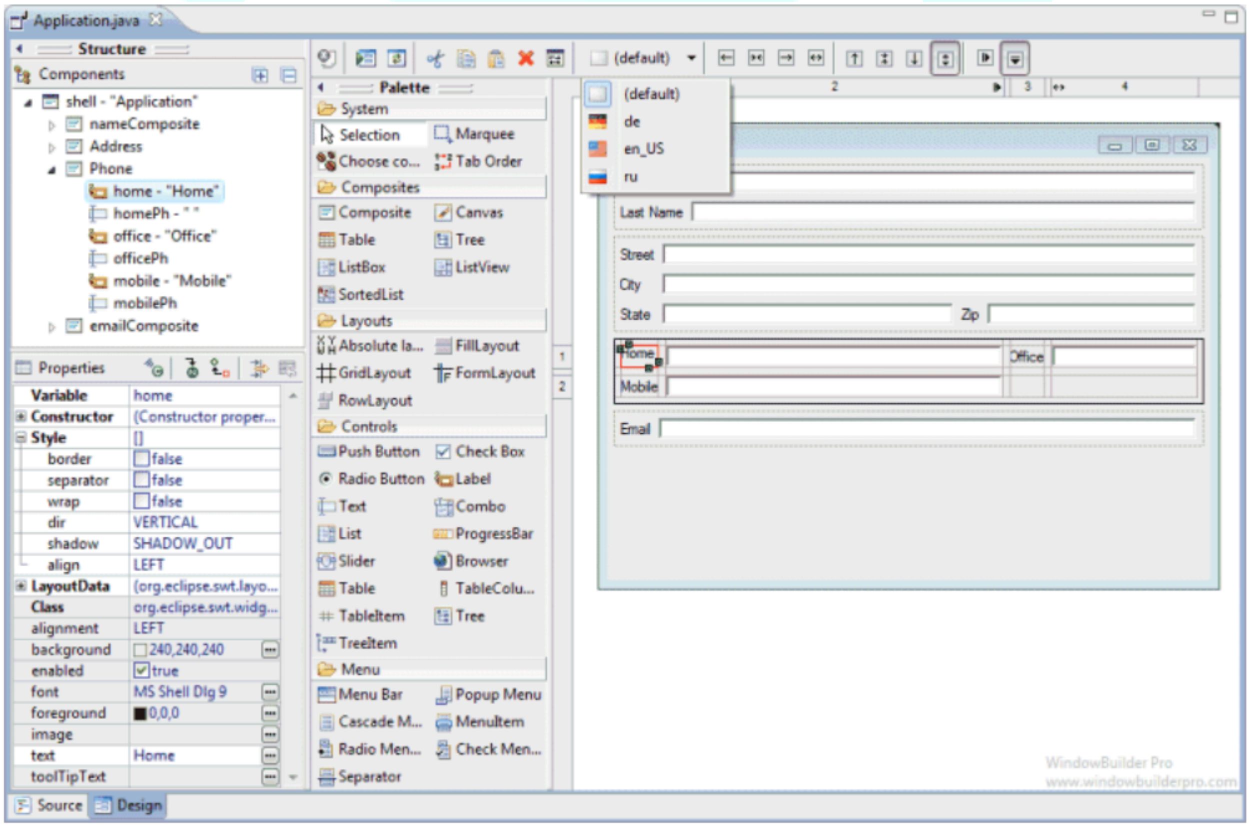
```java
public void xWins(int a, int b, int c) {
    buttons[a].setBackground(Color.GREEN);
    buttons[b].setBackground(Color.GREEN);
    buttons[c].setBackground(Color.GREEN);
    for(int i=0;i<9;i++) {
        buttons[i].setEnabled(false);
    }
    textField.setText("X wins");
}

public void oWins(int a, int b, int c) {
    buttons[a].setBackground(Color.GREEN);
    buttons[b].setBackground(Color.GREEN);
    buttons[c].setBackground(Color.GREEN);
    for(int i=0;i<9;i++) {
        buttons[i].setEnabled(false);
    }
    textField.setText("O wins");
}
```

At this point you have a basic functioning Tic-Tac-Toe application! **Congratulations!** From here there are various additions and improvements we could make. We encourage you to implement whatever changes you feel would create a better experience for the user.

Additional Notes:

- **Creating a JAR file** - FAQ How do I create an executable JAR file for a stand-alone SWT program? - Eclipsepedia
    - If you'd like your program to be more stand-alone, consider creating a JAR file.
    - This JAR file should be runnable in the command line using the command **`java -jar <yourfilename>.jar`**
    - *NOTE: Ensure that your java version used to compile is the same version used to run the JAR file*

- **Eclipse WindowBuilder** - https://eclipse.dev/windowbuilder/
    - If you'd prefer using a GUI to help build your GUI, try using Eclipse WindowBuilder. This will give you a **Design** tab in addition to your **Source** (code) tab and you can see how making changes in one tab affects the other.



Variable	home
Constructor	(Constructor proper...
Style	[]
border	false
separator	false
wrap	false
dir	VERTICAL
shadow	SHADOW_OUT
align	LEFT
LayoutData	(org.eclipse.swt.layo...
Class	org.eclipse.swt.widg...
alignment	LEFT
background	240,240,240
enabled	✓ true
font	MS Shell Dlg 9
foreground	0,0,0
image	
text	Home
toolTipText	