# GUI tools in Java

## What is AWT?

Abstract Window Toolkit - a java library for creating GUI applications. It is a part of the Java standard library. It is a low level library, it is not very flexible. It is not very good for creating modern GUI applications.

## What is Swing?

This is a java framework, that is build on the foundation of AWT. It is more flexible than AWT, and was build to fix the issues from AWT. It is a part of the Java standard library. Swing uses the same handling of events as AWT.

The two main issues it solves are that it had lighter weight components (making them more efficient), and it had a better layout manager.

A swing ui is made up of two elements, the components and the containers. The components are the buttons, labels, text fields, etc. The containers holds group of components like panels, frames, etc.

---

## Example of a Swing app

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

public class Main {
    public static void main(String[] args) {
        JFrame frame = new JFrame("title"); //This is the window of the
app, and the thing inside is the title

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // This
stops the app, by default
        // if you press exit it closes the window but the app might still
be running

        frame.setSize(400,300); // The size of the window

        JLabel text = new JLabel("Hello"); //TO get text to screen

        frame.add(text); //Adds the text to canvas


        JButton button = new JButton("Click me");
        button.setSize(20,10);
        frame.add(button);


        TextField editText = new TextField();
        frame.add(editText);
```

```
        frame.setVisible(true); //Sets the visablty true


    }
}

---
```

## What is JavaFx?

JavaFx was a improvement over Swing because it allowed app to be created for mobile as well. This also allowed the app to have a customizable look and feel by allowing CSS to be used. JavaFx was designed to replace Swing but because Swing was in use for such a long time it is still used.

JavaFx also supports FXML which allows you to get a better control of the user interface.

JavaFx was a part of the Java SDK but in java 8 they made it a separate, library that you have to install.

JavaFx works by having a stage this is the actual window and the size of it, the scene is the container for the components. So the stage is a container for the scene. So all JavaFx apps have to have one scene and one stage.

In JavaFx the components are called nodes. However, a node can also be a group of nodes. A node can have a child node.

The scene graph is called the root node.

JavaFx has layout panels that manage the process of placing elements to the scene (Swing also has this).

A JavaFx application must be a sub-class of Application. The application class defines the life cycle of the application such as

---

## Example of a JavaFx app

```
public class CalLauncher{
    public static void main(String[] args){
        Application.launch(CalApp.class);
    }
}
```

```
public class CalApp extends Application{
    @Override
    public void start(Stage stage) throw Exception{
```

```
        stage.setScence(new Scene(new CalView(new CalModel), 300, 300));
        stage.show();
    }
}
```

```
public class CalModel{
    public void add(int a, int b){
        return a + b;
    }

    public void subtract(int a, int b){
        return a - b;
    }

    public void multiply(int a, int b){
        return a * b;
    }

    public void divide(int a, int b){
        return a / b;
    }

}
```

```
public class CalView extends Pane{

    public CalView(CalController controller){
        var textField = new TextField();
        textField.setFont(Font.font(70));

        var textFiled2 = new TextField();
        textFiled2.setFont(Font.font(70));

        var btnAdd = new Button("Add");
        outField.setFont(Font.font(70));


        btn.setOnAction(e -> {
            try{
                var num1 = Integer.parseInt(textField.getText());
                var num2 = Integer.parseInt(textField2.getText());
                var result = controller.add(num1, num2);
                outField.setText(result + "");
            }catch(NumberFormatException ex){
                outField.setText("Invalid input");
            }
        });
```

```
        var outField = new TextField();
        outField.setFont(Font.font(70));


        getChildren().addAll(
            new VBox(15 // spacing between elements,
            textField, textField2, btnAdd, outField));
    }

}
```

```
public class CalController{
    private CalModel model;
    private CalView view;

    public CalController(CalModel model, CalView view){
        this.model = model;
        this.view = view;
    }

    public void process(String button){
        if(button.equals("Add")){
            model.add();
        }
    }

}
```