**Week 11 Lab – Graphical User Interfaces**

# Lab Intro

In the lecture, we saw how we could create Graphical User Interfaces (GUIs) using **Swing**. This lab will provide you with some hands-on experience of developing your own GUI applications.

**Key Term – Swing  
Swing** is a Java library (collection of classes) that allows developers to create Graphical User Interfaces (GUIs), allowing your programs to have a front-end interface which can be more appealing for users compared to a console application.



1. Launch IntelliJ and create a new **Project** named **Week 11**.
2. Download **Greeter.java** from **Moodle** and import it into your Project (as it’s a single.java file, it’s probably easier just to drag it into your **src** folder)

## Learning Outcomes

* Develop simple GUI programs using the Java Swing classes
* Recap switch statements and enumerated types

## Resources

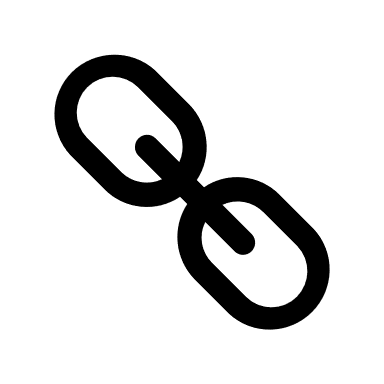
* Week 11 lecture slides
* **Greeter.java** (see Moodle)
* **Doomsday.java** (see Moodle – required for Exercise 2)



# Exercise 1 – Calculator App

Using **Greeter.java** as a reference, write a simple calculator program that allows the user to enter two decimal numbers, and compute & display their sum, difference, product or quotient (depending on the button clicked).

**Unit Link – Team Project**In your second semester, all of you will undertake a unit called **Team Project**, where you will work in groups to develop a product. Although the emphasis of that unit is working as a group and your assessment is presentation, having knowledge of developing front-end user interfaces will be a beneficial skill to have.



You should have two **JTextField** components that allow the user to input two numbers (treat them as floats), and four **JButton** components which are labelled x, +, /, -. The result of the calculation is displayed using a **JLabel**. The development of this program can be broken into two steps: (1) producing the GUI, and (2) implementing the functionality of the different components – detailed below.

## Part 1 – Creating the GUI

Create the application framework (Figure 1) and add all the components. Look at the **Greeter** application from the lecture as an example. You will need to derive (extend) a class from **JPanel**, and add two **JTextField** components, four **JButton** components and a **JLabel** for the result. You can add the **actionPerformed** method, but for now it should be an empty method.

Graphical user interface, application

Description automatically generated

Figure Calculator GUI View

When this stage is complete, you should have a window that looks like the picture above. You should be able to type text into the text fields, and press the buttons, but the inputs will have no effect.

## Part 2 – Adding the Functionality

Now add the functionality. You will need to respond to a button press as follows:

1. Read the strings from the **JTextField** objects and convert them to floating-point numbers (hints below).
2. Calculate the result by applying the appropriate operation and write the output to the JLabel (or an error if the input cannot be converted to a number).

**Hints:**

1. Make sure all the buttons send events to the **JPanel** class (you should call **addActionListener** on each one – see **Greeter.java**).
2. Convert the strings to floats using the **Float.parseFloat( String )** method. This returns a **float**, but throws an exception if it can’t do the conversion. You can use the code below in your solution:

**try** {

// get the operands

**float** val1 = Float.parseFloat( input1.getText() );

**float** val2 = Float.parseFloat( input2.getText() );

**float** result = 0;

// calculate result here...

answer.setText( "Answer: " + result);

}

**catch** (Exception e) {

answer.setText("invalid input");

}

What this does is try to execute all the code in the try block. If it succeeds, the catch block is ignored. If an exception is thrown by any of the code in the try block, the catch block is executed. The details of the exception can be accessed using the local variable e – in this case we don’t query this and just give out a blanket error message.

1. Think carefully about how you use the code in hint **2**. Would it be a good idea to cut and paste this four times into the **actionPerformed** method, once for each button? Probably not. Much better would be to write a single method which reads the input and performs the calculation. You could define an enumeration for the operations, and then pass that into a calculation method. Something like:

enum Operation {

TIMES, PLUS, MINUS, DIVIDE

}

**public** **void** actionPerformed( ActionEvent event ) {

**if** ( event.getSource() == buttonTimes ) {

calculate( Operation.TIMES );

} **else if** ( event.getSource() == buttonPlus ) {

calculate( Operation.PLUS );

} **else if** ( event.getSource() == buttonMinus ) {

calculate( Operation.MINUS );

} **else if** ( event.getSource() == buttonDivide ) {

calculate( Operation.DIVIDE );

}

}

**void** calculate ( Operation op ) {

// ... do stuff. A switch statement may be useful here...

}



# Exercise 2 – DoomsDay Calculator

This task will involve writing a day of the week calculator that allows the user to input a date, and calculates and outputs the day of the week on which that date fell (or will fall).

**Download Doomsday.java from Moodle – don’t change anything in Doomsday.java. You should include it in your project, but you don’t need to modify it.** This is a class that implements John Horton Conway’s ‘doomsday’ algorithm for calculating the day of the week for any date. The class has a static function

**public static** **int** getDayOfWeek( **int** day, **int** month, **int** year )

This returns -1 if the date entered is invalid. For valid dates, it returns 0 for Sunday, 1 for Monday, and so on. Call it like this:

**int** dayOfWeek = Doomsday.getDayOfWeek( 29, 2, 2016 );

System.*out*.println( dayOfWeek );

// outputs '1' - 29/2/2016 was a Monday.

Your task is to write a GUI for this class. It should look something like Figure 2.

Graphical user interface, application, Word

Description automatically generated

Figure Doomsday GUI View

The program should include the following features:

* The day should be given by name, not the integer returned from the function.
* It should output an error if the input cannot be converted into integers.
* It should output a different error if the input can be parsed but is not a valid date.
* The output should include the date.

**Example outputs:**

‘Invalid input’

‘45/32/2020 is not a valid date’

‘29/2/2016 is a Monday’