

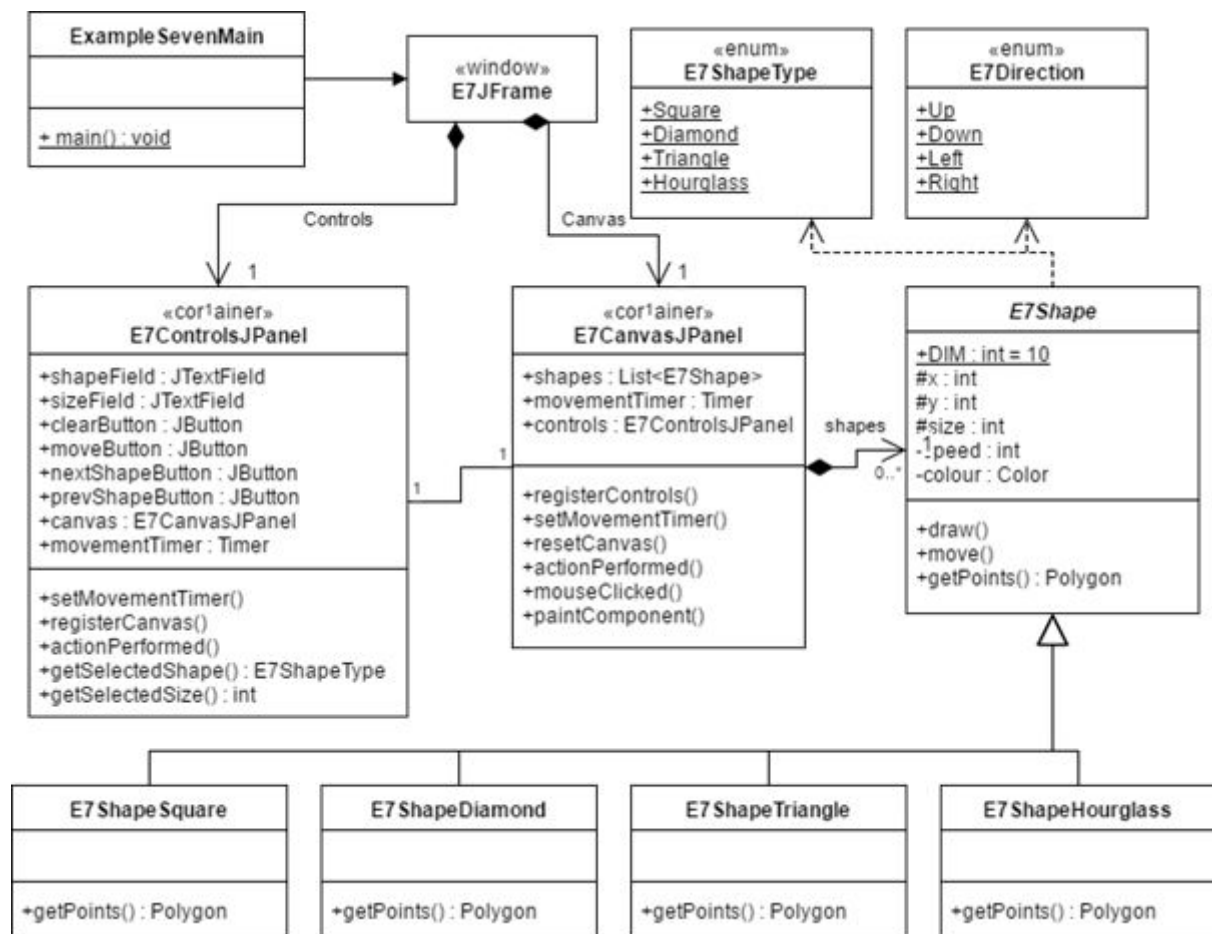


PGCert IT: Programming for Industry

Lab 11: Introduction to Swing

Exercise Zero: A Complete Worked Example

To help understand the Swing Framework, we include a complete worked example of a GUI application that demonstrates most of the features you'll need to complete these labs. The demo is included in the package example within the lab materials project. Please review the (extensive) comments to see how the program works. An overview of the application is shown in this UML.



Exercise One: Body Mass Index (BMI)

Body mass index (BMI) is a measure of body fat based on height and weight. A healthy BMI is considered to be in the range 19 - 24.9.

Declare the instance variables and complete the methods in the `ExerciseOnePanel1` class as per the comments in the source code, so that the program looks and behaves as below.

The user first enters their height in metres and their weight in kilograms, then presses one of the buttons:

- If the user presses the **“Calculate BMI”** button, the BMI is calculated, rounded to 2 decimal places (using the method `roundTo2DecimalPlaces()`) and displayed. The formula for calculating the user’s BMI is:

$$\text{BMI} = \text{weight} / (\text{height} * \text{height})$$



The screenshot shows a Java Swing window titled "Body Mass Index". It contains two input fields: "Height in metres:" with the value "1.64" and "Weight in kilograms:" with the value "63.25". Below these are two buttons: "Calculate BMI" and "Calculate Healthy Weight". The "Calculate BMI" button is highlighted, indicating it has been pressed. To the right of the "Calculate BMI" button is a text field labeled "Your Body Mass Index (BMI) is:" containing the value "23.51". Below the "Calculate Healthy Weight" button is a text field labeled "Maximum Healthy Weight for your Height:" which is currently empty.

- If the user presses the **“Calculate Healthy Weight”** button, the maximum “healthy” weight for the user’s height is calculated, rounded to 2 decimal places, and displayed. The formula for calculating the maximum healthy weight is:

$$\text{Maximum healthy weight} = 24.9 * \text{height} * \text{height}$$



The screenshot shows the same "Body Mass Index" window. The "Height in metres:" field still contains "1.64", but the "Weight in kilograms:" field is now empty. The "Calculate BMI" button is no longer highlighted. The "Calculate Healthy Weight" button is highlighted, indicating it has been pressed. The "Your Body Mass Index (BMI) is:" field is now empty. The "Maximum Healthy Weight for your Height:" field now contains the value "66.97".

Note: The method `roundTo2DecimalPlaces()` has been provided for you to round amounts to 2dp.

Exercise Two: A Simple Calculator

Complete the `ExerciseTwoPanel` class so that the program looks and behaves as follows:

The program has a graphical user interface that consists of three `JTextField`s, a `JLabel` and two `JButtons`:



If the **"Add"** button is clicked, the value from the first `JTextField` is added to the value in the second `JTextField`, and the result displayed in the third `JTextField`, as illustrated in the screenshot below:



If the **"Subtract"** button is clicked, the value from the second `JTextField` is subtracted from the value in the first `JTextField`, and the result displayed in the third `JTextField`:



Complete the code for the `JPanel` class that sets up this graphical user interface, and processes the `ActionEvent` that is generated when one of the buttons is clicked. The size of the first two `JTextField`s should be wide enough to hold at least 10 characters and the result `JTextField` should be wide enough to hold at least 20 characters.

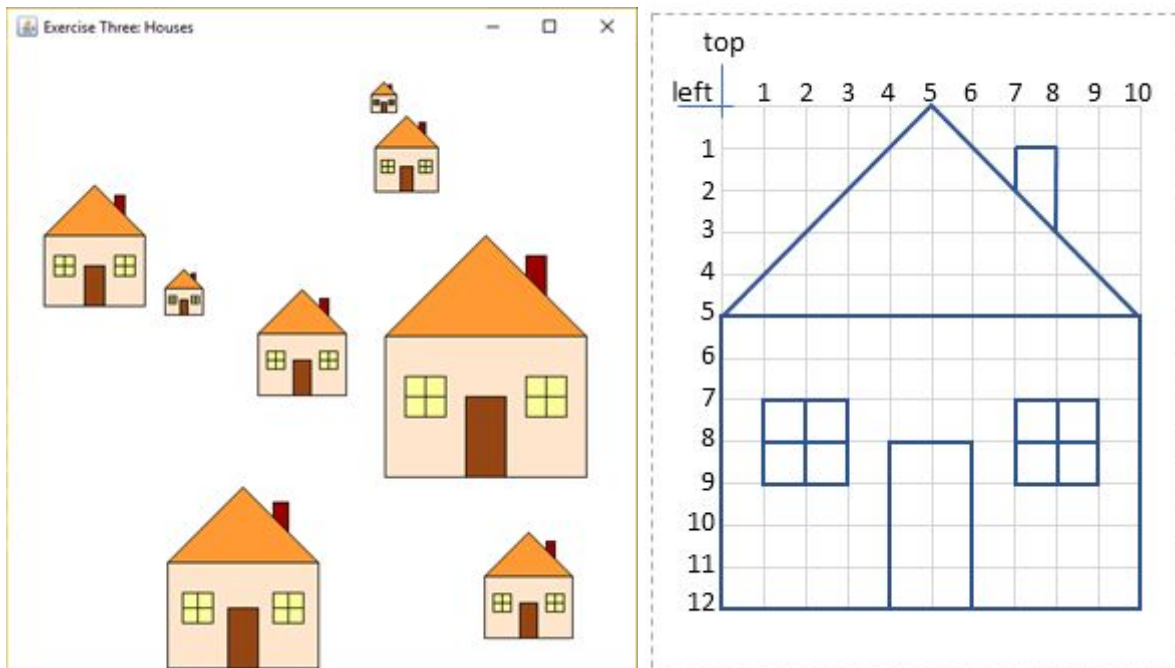
Exercise Three: Drawing in a JPanel

Complete the `drawHouse()` method in the `ExerciseThreePanel` class so that it draws a house. The position and size of the house will be passed in as the parameters `left`, `top`, and `size`.

- The house is to be drawn the same as the house drawn on the grid below.
- The house should be drawn in the position specified by the `left` and `top` parameters.
- The colours of various parts of the house are defined in the `ExerciseThreePanel` class, as constants (e.g. `OUTLINE_COLOR`, `MAIN_COLOR`).
- The size of the house, in pixels, should be the same as in the grid below when the `size` parameter is 1. If the `size` parameter is 2, everything should be twice the size, and so on.
- Your method has been called 8 times from the `paintComponent()` method, using different positions and different sizes, so the drawing window will show 8 differently positioned, different sized houses as shown in the screenshot below.

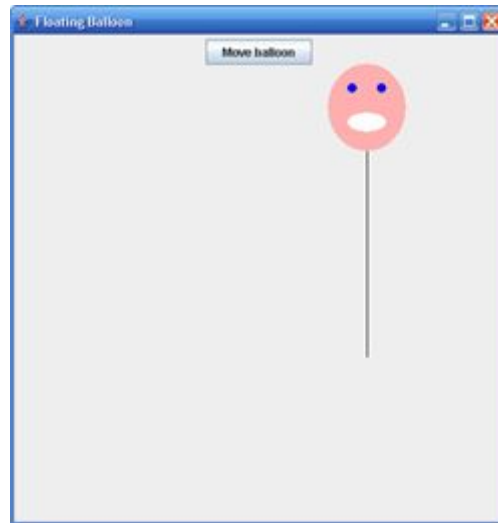
Hint #1: The `Graphics` object can draw many kinds of shapes and lines. Have a look at its [JavaDoc](#) for more information. The house can be drawn using some combination of these methods:

- `setColor()`, `fillRect()`, `drawRect()`, `drawLine()`, `fillPolygon()`, `drawPolygon()`



Exercise Four: Balloons

The `ExerciseFourPanel` class currently contains a `JButton` labelled **"Move balloon"** and a simple drawing of a balloon. When the "Move balloon" button is clicked, the balloon moves one position to the right.



Step 1: Responding to Key events

We want to respond to key events, so that when the user presses one of the arrow keys on the keyboard, our balloon can change direction.

Firstly, make `ExerciseFourPanel` implement the [KeyListener](#) interface. This requires three methods to be implemented, but we'll just use the `keyPressed()` method for now, and leave the others blank.

Next, add code to the `keyPressed()` method so that the balloon changes direction when the user presses one of the arrow keys on the keyboard. The [KeyEvent](#) object you have access to within that method contains a method called `getKeyCode()`, which will return one of the keycode constants defined within the `KeyEvent` class (e.g. `VK_UP` for the up arrow). The direction of the balloon can be changed using its `setDirection()` method.

Next, modify `Balloon`'s `move()` method so it actually knows how to move directions other than `Right`.

Finally, in `ExerciseFourPanel`'s constructor, call its `addKeyListener()` method to add itself as a `KeyListener` on itself, so that the `keyPressed()` method will actually get called appropriately by Swing.

Step 2: Animating the balloon

Instead of clicking a button continually to move the balloon, it would be better if the balloon moved by itself. To do this, we'll use a `Timer` to animate the panel. Consult the provided examples to see how you can do this.

When the program first starts, the balloon should be stationary, and should start moving only when one of the arrow keys is pressed. The balloon should continue to move in the current direction until one of the arrow keys is pressed. When one of the arrow keys is pressed, the balloon should immediately change direction.

Once you've got the timer working as described, you can remove the `JButton` from the panel as it's no longer required.

Step 3: Stopping the balloon

Modify your solution so that when the user presses the "s" key on their keyboard, the balloon stops animating. When they press one of the arrow keys again, the balloon should resume moving in the corresponding direction.

Step 4: Adding more balloons!

Modify the program to add more balloons. Try using an `ArrayList` or a similar collection to hold them. Exactly what those balloons do and how they're animated is up to you, but be creative! Some questions to consider:

- Are all the extra balloons added at once? Or are they added over time?
- Does the user control just their one balloon, or do they control many (or all of them)?
- What happens when the balloons float off-screen. Do they reverse direction? Do they pause? Do they teleport back to the other side?
- Do the balloons ignore each other? Or is there some kind of collision detection that lets them interact?

