Assignment Pie Charts

Object Orientation

Spring 2025

1 JavaFX

JavaFX is a framework for creating graphical user interfaces in Java.

2 Learning Goals

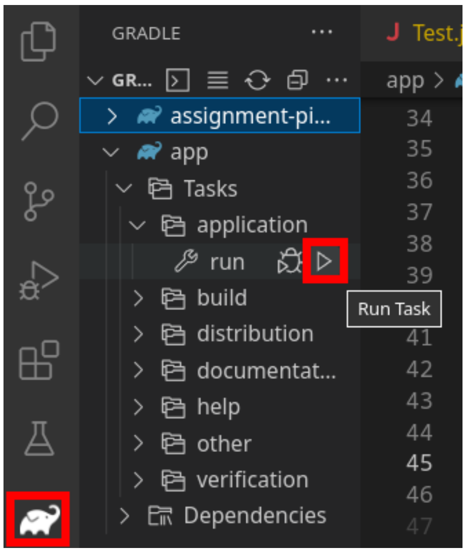
In this exercise you use JavaFX to create an application with a graphical user interface. After doing this exercise you should be able to:

- Create new JavaFX projects
- Use panes to automatically layout GUI elements
- Use properties to automatically update GUI elements

3 Setting Up JavaFX

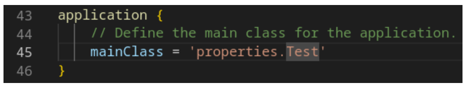
First, extract the assignment zip file. Next, go to File → Open Folder, and select the extracted `assignment-pie-charts-start` folder. At this point, Gradle will handle setting up the project for usage with JavaFX.

You can run the provided `WarmUp.java` by performing the Gradle `app:run` task pictured below:



If you do not run your program this way, you will likely get an error. **For people that use IntelliJ IDEA:** there should be a similar Gradle section on the right which also contains the `app:run` task which you need to start. Alternatively, you can find it by going to View → Tool Windows → Gradle. **For people that use Eclipse:** there should be a 'Gradle Tasks' tab on the bottom pane which contains `application:run`. Alternatively, you can find it by going to Window → Show View → Other → Gradle → Gradle Tasks. **For people that use the command-line:** you can run the program using `./gradlew run --console=plain -q` like before.

If you want to run a file which is different from `WarmUp.java`, you can change this in the Gradle configuration at `app/build.gradle`. For example, the image below shows you how to change the main file to `Test.java` located in the `properties` package:



```
application {
    // Define the main class for the application.
    mainClass = 'properties.Test'
}
```

The rest of this `build.gradle` file describes the dependencies of the project, and can be used in JavaFX projects of your own creation.

4 Warming Up With Property Bindings

This is a warm-up exercise for yourself to practice. It will not be graded.

Properties are values that can be connected to other Properties with bindings. With Properties and bindings you can make formulas of values whose result value automatically gets updated when one of the in-between values changes. For example, you can make an IntegerProperty that is always 2 higher than another IntegerProperty.

Properties can be used without GUIs, but JavaFX makes heavy use of properties to automatically update widgets. You can access many attributes of widgets as properties and bind them to the model of your program. In this way, whenever the model changes, the GUI gets updated automatically.

4.1 Your Tasks

1. In the project template you find a test class PropertyTest. Use only property bindings to make the test cases pass.
2. In the project template you find a class WarmUp. Fill in the code as described in the comments. Again, you only have to use property bindings.

5 Problem Sketch

In this assignment you implement a GUI for a pie chart generator. To keep it simple, the pie chart has a fixed number of four segments. You also do not have to draw the pie chart, but only display the proportions of the segments.

Figure 1 shows a screenshot of the pie chart generator. On the left side are four input fields that only accept positive integers. On the right side are four labels that display the proportion of each integer relative to the total sum. When the user changes one of the input values, all outputs should update automatically.

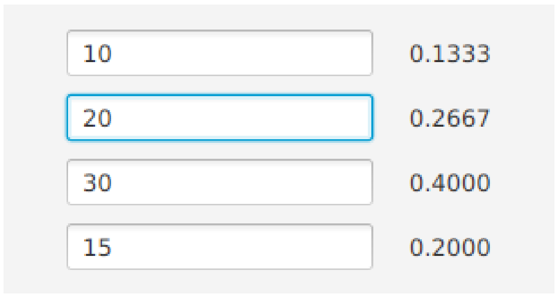

```
10      0.1333
20      0.2667
30      0.4000
15      0.2000
```

Figure 1: Screenshot of the simple pie chart generator.

5.1 Using Panes

In JavaFX, a `Pane` manages the layout of its child elements. There are different panes that arrange their children in different ways. Panes can be nested. In this assignment you should use a `GridPane`. `GridPane`s arrange their children in a tabular fashion. The screenshot in Figure 1 is made with the following settings.

```
GridPane root = new GridPane();
root.setAlignment(Pos.CENTER);
root.setHgap(20);
root.setVgap(10);
```

5.2 Validating Input

JavaFX text fields support a callback mechanism that lets you register a function that is called every time the user changes the input. To make it easy, we provide the code you should use.

```
TextField textField = new TextField();
textField.textProperty().addListener(
  new ChangeListener<String>() {
  @Override
  public void changed(
    ObservableValue<? extends String> observable,
    String oldValue, String newValue) {
    if (!newValue.matches("[1-9]\\d{0,3}")) {
      textField.setText(oldValue);
    } } });
```

This function checks if the string in the text field matches a simple regular expression. If it does not match, the old value is put back. The given regular expression ensures a positive integer of at most 4 digits.

5.3 Using Properties

When the user changes one of the input fields, the output fields should update automatically. For this you should use `Properties`.

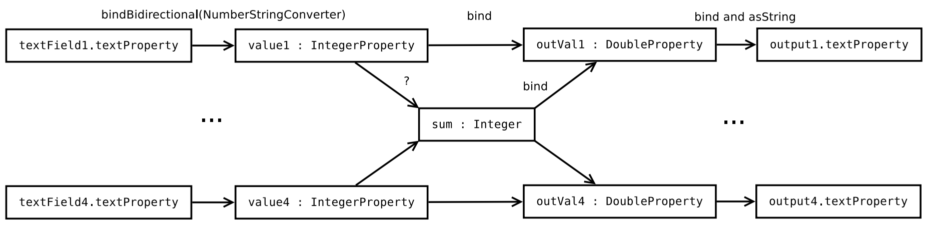


Figure 2: The properties and their dependencies.

The input and output fields have text properties, but you need to do some arithmetic with their values. For this you should build a network of properties that depend on each other and perform the necessary conversions.

For every input field you should create a `SimpleIntegerProperty` that is synchronized with the text of the input field. You can bind it to the text using `Property.bindBidirectional()` and `NumberStringConverter`. This gives you a translation and update from a string property to an integer property.

Next, you should create a `SimpleIntegerProperty` called `sum` that holds the sum of all the text field's integer properties. How to keep the sum updated is part of the exercise.

You should create a `SimpleDoubleProperty` for each output field, that is bound to the ratio of the corresponding input property and the sum. This requires a conversion from integer to double in the binding. The ratio is a real number between 0 and 1, but both sum and input properties are integers. If you divide an integer by a bigger integer, it will be rounded to 0. You have to convert the integer to double before the division. This can be done by using `IntegerProperty.add(0d)`.

The final step is feeding each double property into its respective output label. This can be done with `DoubleProperty.asString`, which takes a format string as argument.

Figure 2 shows the properties in the system. The arrows denote dependencies and are labelled with the way the properties should be updated.

6 Your Tasks

1. Create a new empty Java Application as described in Section 3.
2. Create panes for the layout.
3. Add TextFields and Labels to the panes.
4. Create properties as necessary and implement the update mechanics.

7 Optional Extra Challenge: Variable Number of Segments

Add a plus and a minus button to dynamically add and remove segments. You can put those buttons in the bottom pane of the BorderPane. Hint: Create a new intermediate sum for each input field that depends on the previous intermediate sum.

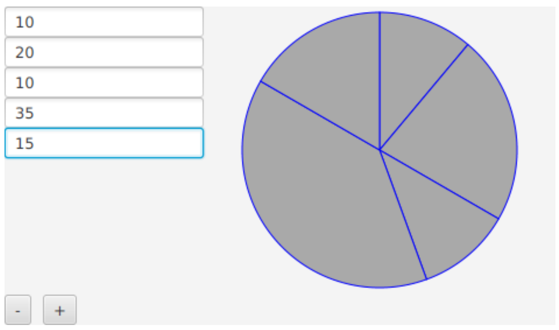


Figure 3: Optional Extra Challenge: draw the pie chart with variable number of segments.

8 Optional Double Extra Challenge: Draw The Pie Chart

If you want, you can try drawing the pie chart. Instead of a `GridPane`, you should use a `BorderPane`. Put the input fields in a `VBox`, which you put in the left area of the BorderPane. Put the pie chart drawing in a simple `Pane`, which you put in the center area of the `BorderPane`. Create `Arcs` with the center and radius bound to the dimensions of the pane. The start and end angle should be bound to the corresponding output properties. When the user changes an input field, the drawing should update automatically.

8.1 Submit Your Project

To submit your project, follow these steps.

1. Find the folder that contains your assignment. In Visual Studio Code, you can find this by going to: File → Open Folder. Add the correct folder to a zip file. At the root level, your zip file should contain only one folder, e.g. `assignment-student-start`, which contain the entire project, i.e. the `app` folder and the Gradle files. *Do not submit only the `.java` files or the `src` folder!*
2. **Submit this zip file on Brightspace**. Do not submit individual Java files. Only one person in your group has to submit it. Submit your project before the deadline, which can be found on Brightspace.
3. **Do not submit any other format.** Do not submit .rar or .tar.gz or .7z files. Only zip files are allowed.