Update readme.md for Scalable images
Jonas Solhaug Kaad authored 1 month ago

Forked from an inaccessible project.

readme.md 8.68 KiB

Exercises for Lecture 12

Task 1 - JavaFX Settings

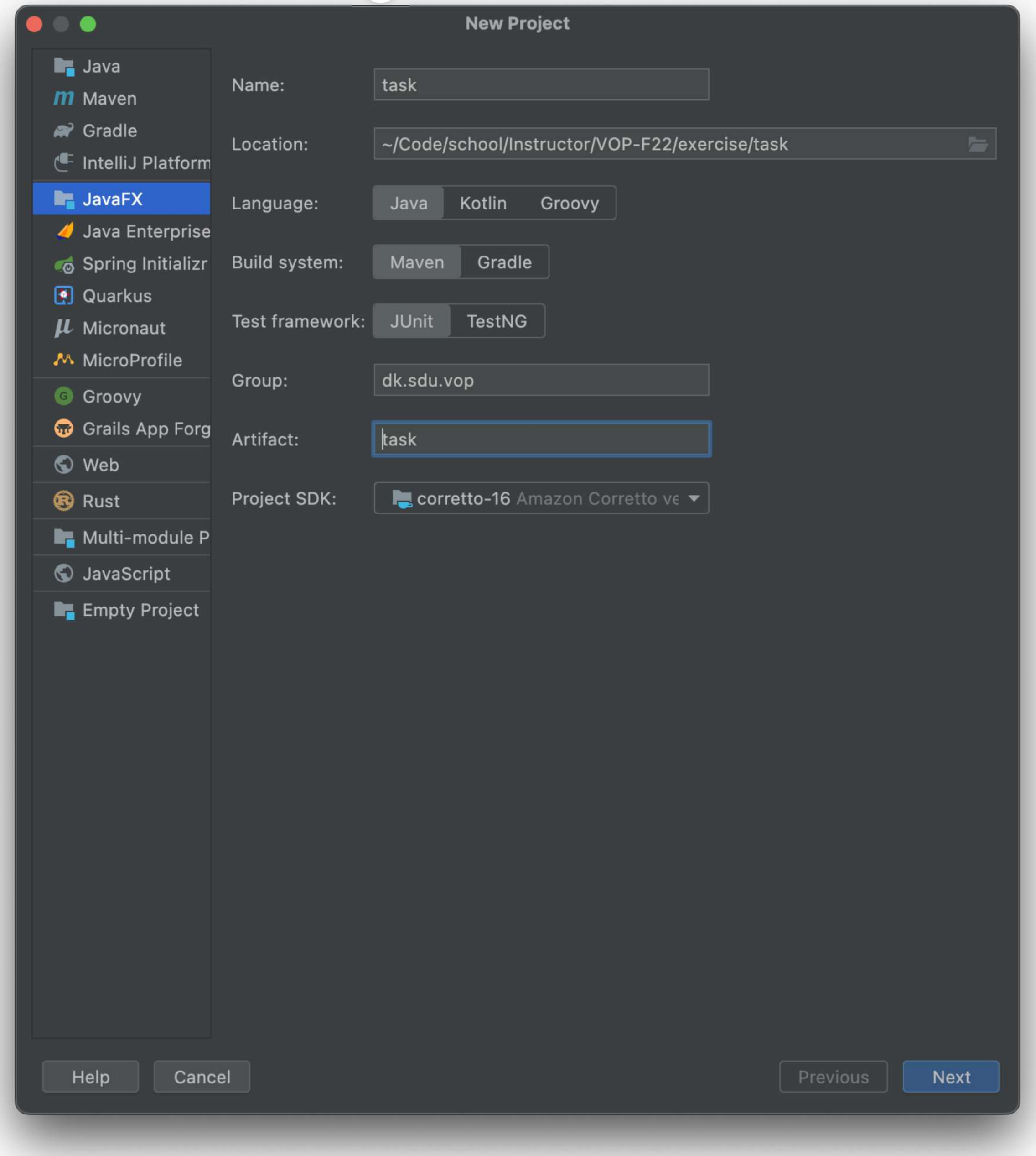
Purpose: To create an exam project as a JavaFX project, correctly named and prepared for the following tasks.

Open IntelliJ and select the JavaFX tab.

Use the following parameters:

- Language: **Java**
- Build system: **Maven**
- Test framework: **JUnit**
- Project SDK: **Your newest one (18-19-20 will be good)**

The additional settings are up to yourself.



Name: task

Location: ~/Code/school/Instructor/VOP-F22/exercise/task

Language: Java Kotlin Groovy

Build system: Maven Gradle

Test framework: JUnit TestNG

Group: dk.sdu.vop

Artifact: task

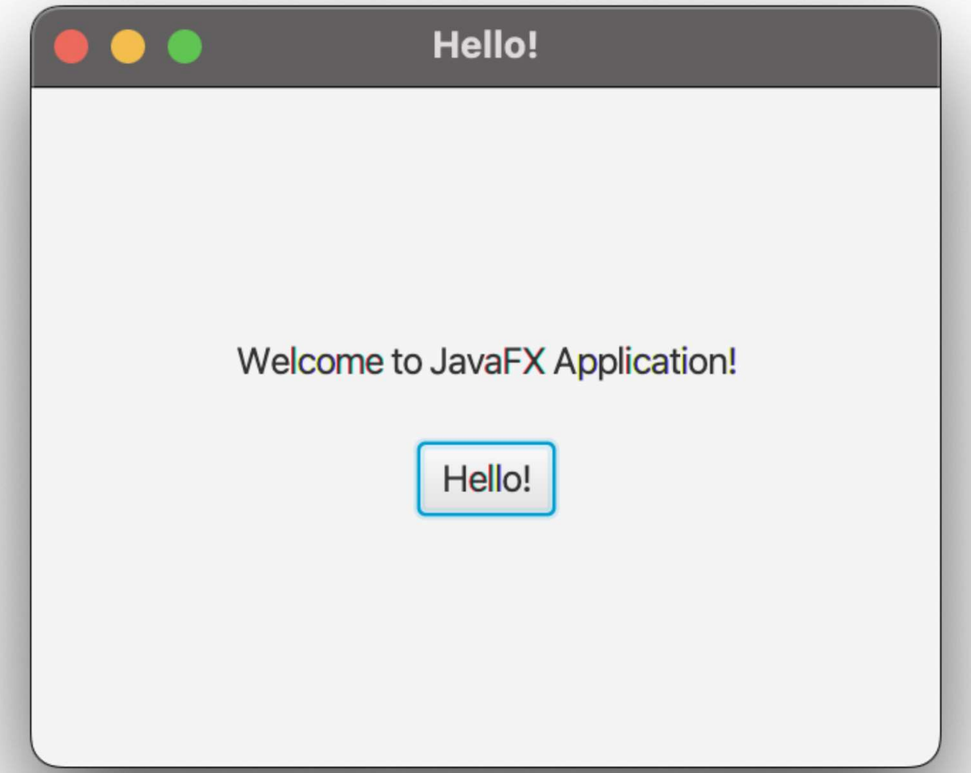
Project SDK: corretto-16 Amazon Corretto ve

When pressing next you will be presented with different dependencies -

none are neeeded.

Press **finish**.

This will initialize the project, verify that everything is working by opening the _"HelloApplication.java" file and starting the main method. This should prompt the following window:



You are now ready to continue with the following exercises 😁.

Task 2 - Facade-pattern and some basic Java

Purpose: To implement a Facade-pattern class, that the GUI layer is able to utilize.

1. Implement a class `Facade.java` in the package *boilerplate*
2. Create a private variable `int[] intArray` within the class
3. Create a private variable of type `java.util.Random`
4. Initialize the Random-generator in the constructor
5. Create a "getter()"-method, that returns `intArray;`

In the following three subtasks, you'll program three algorithms that utilizes arrays, loops and conditionals.

Task 2.1 - public int[] fillArray(int size,int max)

Implement the method, so it initializes `intArray` to the size of `size` and fills it with random numbers in the interval `[0..max]`. The array is then returned.

Task 2.2 - public int sumOfDivisors(int divisor)

Implement the method, so it returns the sum of the numbers within the `intArray` variable that is divisible by the `divisor` argument (`x % divisor == 0`).

Task 2.3 - public int[] fillUniqueArray(int size, int max)

Implement the method, so it initializes `intArray` to the size of the argument `size` and fills it with random unique numbers within the `[0..max]` interval. Ensure that `size < max` and no numbers are repeated.

_Hint: define a private method, that verifies if the numbers from the random-generator exists in the array, before inserting it.

Before the array is returned, it's a good idea to sort it using `Arrays.sort()`, as it makes it much easier to spot if the numbers truly are unique.

Example: Execution of your `main()` -method could give the following:

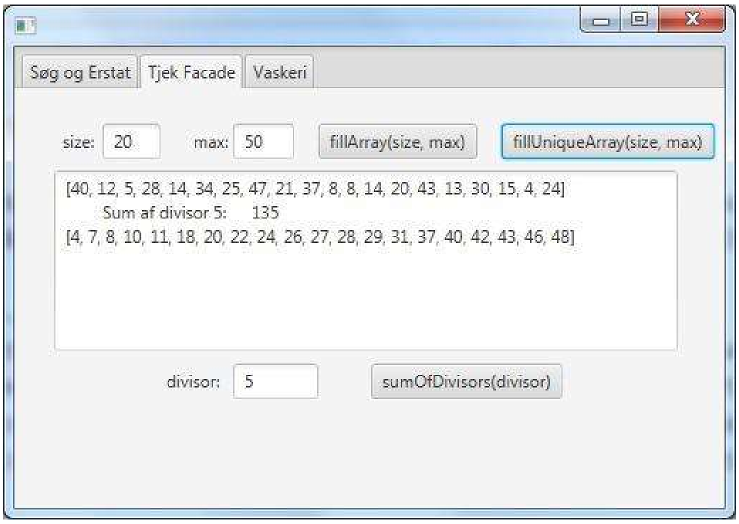
```
fillArray: [0, 4, 5, 5, 4, 5, 1, 5, 6, 2, 7, 5, 4, 2, 1, 8, 1, 4, 9, 8] Divisors of 3 has Sum: 15
fillUnique: [1, 2, 3, 4, 6, 7, 8, 9, 10, 13, 16, 17, 18, 20, 21, 23, 24, 26, 27, 28] size er larger than max!
Error: null
```

Task 2.4 - Calling methods from the GUI layer

Create a new tab panel within your JavaFX application and name it "*Verify Facade*".

Declare the facade as a variable and initialize i within the `initialize()` method in the accompanying controller.

Create components within the tab, such that the three methods in the Facade class can be called from the UI. You can use the following UI as reference:



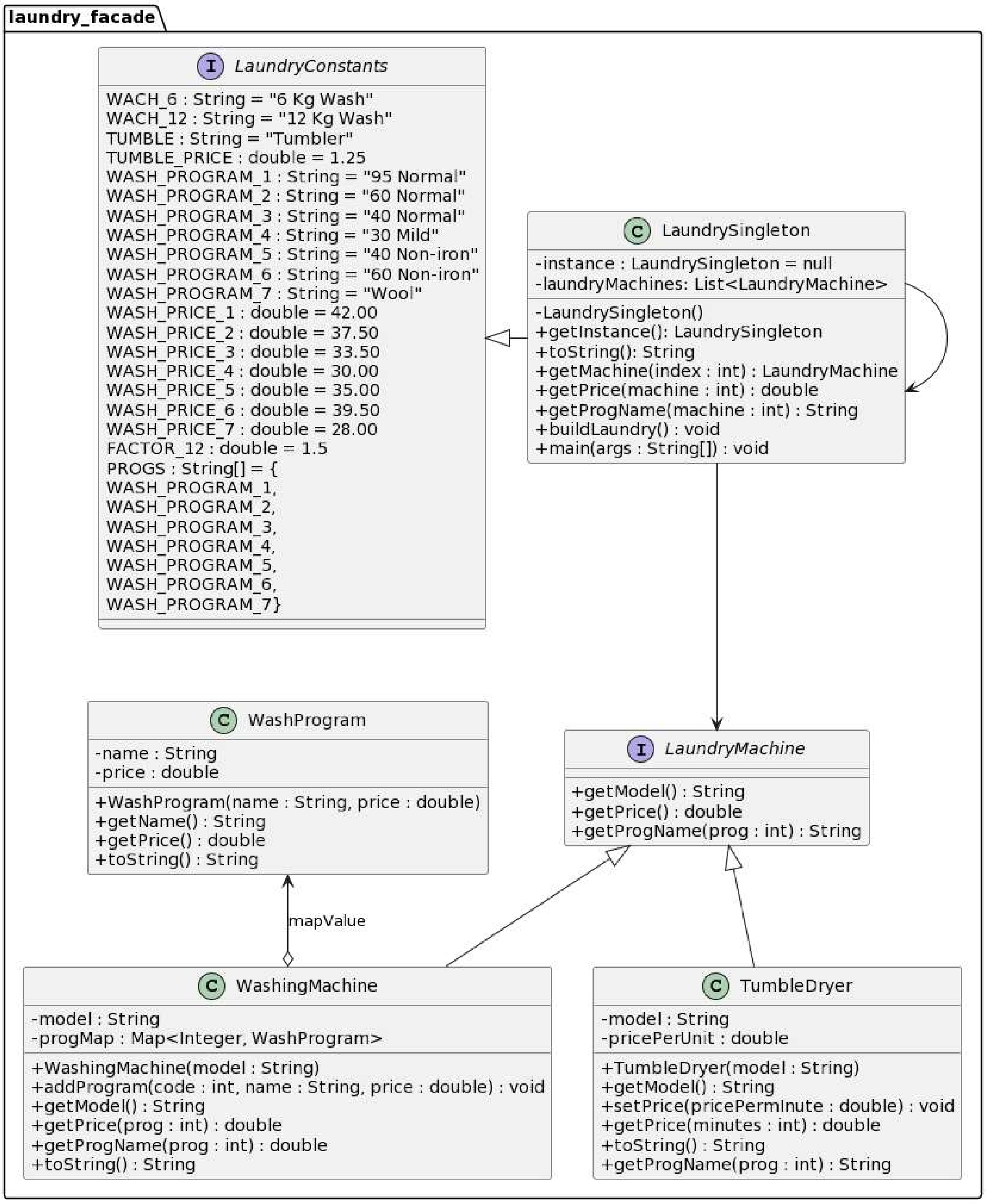
size: 20 max: 50

```
[40, 12, 5, 28, 14, 34, 25, 47, 21, 37, 8, 8, 14, 20, 43, 13, 30, 15, 4, 24]
        Sum af divisor 5:     135
[4, 7, 8, 10, 11, 18, 20, 22, 24, 26, 27, 28, 29, 31, 37, 40, 42, 43, 46, 48]
```

divisor: 5

Task 3 - Polymorphism and a facade that uses the singleton-pattern

This class diagram shows a simplified structure of a laundro mat:


laundry_facade
I LaundryConstants
WACH_6 : String = "6 Kg Wash"
WACH_12 : String = "12 Kg Wash"
TUMBLE : String = "Tumbler"
TUMBLE_PRICE : double = 1.25
WASH_PROGRAM_1 : String = "95 Normal"
WASH_PROGRAM_2 : String = "60 Normal"
WASH_PROGRAM_3 : String = "40 Normal"
WASH_PROGRAM_4 : String = "30 Mild"
WASH_PROGRAM_5 : String = "40 Non-iron"
WASH_PROGRAM_6 : String = "60 Non-iron"
WASH_PROGRAM_7 : String = "Wool"
WASH_PRICE_1 : double = 42.00
WASH_PRICE_2 : double = 37.50
WASH_PRICE_3 : double = 33.50
WASH_PRICE_4 : double = 30.00
WASH_PRICE_5 : double = 35.00
WASH_PRICE_6 : double = 39.50
WASH_PRICE_7 : double = 28.00
FACTOR_12 : double = 1.5
PROGS : String[] = {
WASH_PROGRAM_1,
WASH_PROGRAM_2,
WASH_PROGRAM_3,
WASH_PROGRAM_4,
WASH_PROGRAM_5,
WASH_PROGRAM_6,
WASH_PROGRAM_7}
C LaundrySingleton
-instance : LaundrySingleton = null
-laundryMachines: List<LaundryMachine>
-LaundrySingleton()
+getInstance(): LaundrySingleton
+toString(): String
+getMachine(index : int) : LaundryMachine
+getPrice(machine : int) : double
+getProgName(machine : int) : String
+buildLaundry() : void
+main(args : String[]) : void
C WashProgram
-name : String
-price : double
+WashProgram(name : String, price : double)
+getName() : String
+getPrice() : double
+toString() : String
I LaundryMachine
+getModel() : String
+getPrice() : double
+getProgName(prog : int) : String
mapValue
C WashingMachine
-model : String
-progMap : Map<Integer, WashProgram>
+WashingMachine(model : String)
+addProgram(code : int, name : String, price : double) : void
+getModel() : String
+getPrice(prog : int) : double
+getProgName(prog : int) : double
+toString() : String
C TumbleDryer
-model : String
-pricePerUnit : double
+TumbleDryer(model : String)
+getModel() : String
+setPrice(pricePermInute : double) : void
+getPrice(minutes : int) : double
+toString() : String
+getProgName(prog : int) : String

The interface `LaundryMachine` represents all machines, that can be used inside a laundromat (wash, drying, spin drying, soap-vendor, coffee machine and more.). However, in this task we're solely focusing on washing machines and dryers:

`LaundryMachine.java` (supplied boilerplate code) defines the methods:

Method	Return
`String getModel()`	Returns the model of the machine
`double getPrice(int program)`	Returns the price when using the machine
`String getProgName(int program)`	Returns a description of a program that can be used

`TumbleDryer.java` represents dryers:

- Implements the interface `LaundryMachine` in the class (Hint: how do we normally implement interfaces?)
- The constructor takes a `model` as an argument

Method	Info
`void setPrice(double pricePerMinute)`	Sets the minut price when using the machine

Method	Info
`double getPrice(int program)`	Returns the price of a given program.
`String getProgName(int program)`	Returns for example "Drying for 30 minutes", if prog = 30.

`WashingMachine.java` represents washing machines:

- implements the `LaundryMachine` interface
- the constructor takes a model description as an argument
- contains an attribute of the type `Map<Integer, WashProgram>` : *Contains the programs that are available.*

Method	Info
`void addProgram(int prog, String name, double price)`	Defines a washing program and puts it into the map with `prog` as key
`double getPrice(int program)`	Returns the price of use of program `program`
`String getProgName(int program)`	Returns the name of the program `prog`

`WashProgram.java` (supplied code) represents a single wash program including its name and price.

`LaundryConstants.java` (supplied code) is an interface containing model definitions, program names, program prices and an array of program-names. These constants can be used to create a test laundro mat, and also used within the user interface in task 3.2.

`LaundrySingleton.java` represents a test laundromat, which partially can be run through the `main()` method and can be partially used as a *Facade* within the JavaFX GUI. Only the part that makes this class a singleton isn't implemented.

It contains a method `public void buildLaundry()` , that creates a laundromat with the values from `LaundryConstants` .

Task 3.1 - Singleton

Purpose: To implement a class, which uses the singleton pattern.

Create code within the `LaundrySingleton.java` , such that it becomes a singleton class. The singleton pattern dictates that instance methods only are available through a static method, such as the `public static LaundrySingleton getInstance();` and that whenever you interact with the class, it is through the same instance every time

Task 3.2 - Implementations of the LaundryMachine interface

Purpose: Implementation of polymorphism methods defined within the interface.

Implement the missing code in:

- `TumbleDryer.java`
- `WashingMachine.java`

When executing the `main()` -method within `LaundrySingleton` , the output should look like so:

```
Washing machine max 6 kg:
40 Iron-free        35,00

Washing machine max 12 kg:
40 Iron-free        52,50

Drying Machine:
Drying for 5 minutter 6,25
```

Task 3.3 - GUI

Purpose: Implementation of a simple user interface that uses the `LaundrySinleton` class to make polymorphism calls to the `LaundryMachine` interface implementations.

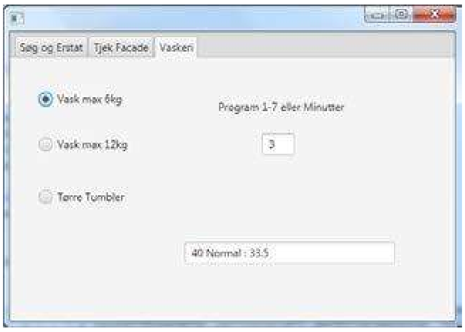
Create a new `Tab` within your user interface named "*Laundromat*".

From the UI, it should be possible to test the laundro mat. For each of the three machines, generated by the `LaundrySingleton.buildLaundry()` , it should be possible to select a program (and pick minutes for the dryer) and get a price quote:

- Add the line `LaundrySingleton.getInstance().buildLaundry()` ; to the `initialize()` -method within your JavaFX controller such that the test laundro mat is available.

- The test user interface contains the following:
  - Three `RadioButton` s sharing a common `ToggleGroup` , that lets the user pick a washing machine.
  - A `Label` and an accompanying `TextField` allowing the user to enter the program number or the drying time.
  - A `Textfield` that shows the selected program and price.
  - When a washing machine is selected, use the program name and the price mthods defined in `LaundrySingleton` .

The following three examples that shows a possible solution to the UI (in the last example the value -1.0 is used as an error code when a program is missing):



● Vask max 6kg

○ Vask max 12kg

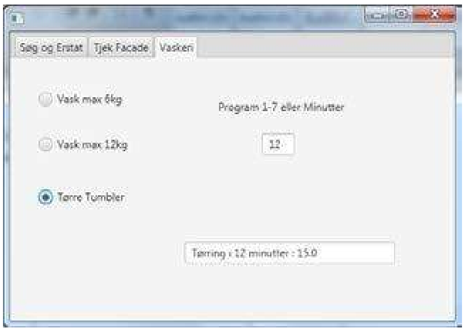
○ Tørre Tumbler

Program 1-7 eller Minutter

3

40 Normal : 33.5

Program 3 selected on machine 0.



○ Vask max 6kg

○ Vask max 12kg

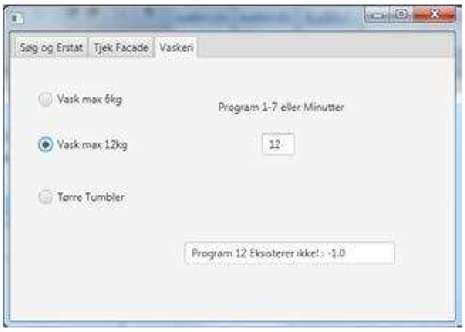
● Tørre Tumbler

Program 1-7 eller Minutter

12

Tørring i 12 minutter : 15.0

12 minutes of drying selected on machine 2.



○ Vask max 6kg

● Vask max 12kg

○ Tørre Tumbler

Program 1-7 eller Minutter

12

Program 12 Eksisterer ikke! : -1.0

Non-existent program selected on machine 1.

```java
package laundry_facade;

import java.util.Random;

public class Facade {
    //Vi starter med de simple OOP-Steps.
    private int[] intArray;

    Random random;

    public Facade(){
        random = new Random();
    }

    //Her laver vi vores inkapsulation :)
    public int[] getIntArray(){
        return intArray;
    }

    //Vi har startet med, at definere parametrene.
    public int[] fillArray(int size,int max){
        //Vi har her sagt, at vi ønsker at danne en array.
        intArray = new int[size];
        //Herefter har vi lavet en for-lykke, hvor vi har gjort således at den kører igennem indeksernes plads.
        //Så skal der i forlykken placeres tilfældige tal således at de starter fra indeks 0 til en maksimale array.længde.
        for(int i = 0; i< intArray.length;i++){
            intArray[i] = random.nextInt(0,max);
        }
        return intArray;
    }

    //Her i tilfældet, kan det ses at vi har summen af divisorer.
    //Vi har en placeringsvariabel (sum) som er 0.
    //Når vi triller igennem for-lykken kan vi se at vi finder modulus og derefter returner summen ved
```

```java
tillæggelsen af array-indekser.
    public int sumOfDivisors(int divisor){
        int sum = 0;
        for(int i = 0; i<intArray.length;i++){
            if(intArray[i]%divisor==0){
                return sum+= intArray[i];
            }
        }
    }

    //Her kan det ses, at vi har fill unique array hvor vi bruger de samme koncepter, men følger nøje efter opgavens beskrivelser.
    public int[] fillUniqueArray(int size, int max){
        if(!(size<max)){
            System.out.println("Size is bigger than Max");
            return null;
        }
        intArray = new int[size];
        for(int i =0; i<intArray.length;i++){
            if(true){
                int randomnumber = random.nextInt(0, max);
                if(!contains(randomnumber, i)){
                    intArray[i] = randomnumber;
                }
            }
        }
    }



}

```

```java
package laundry_facade;


public interface LaundryMachine {

    String getModel();

    double getPrice(int prog);

    String getProgName(int prog);

}

```

```java
package laundry_facade;

import java.util.Arrays;


public class LaundrySingleton implements LaundryConstants extends Application {
    private static LaundryMachine[] laundryMachines;
    private static LaundrySingleton instance;

    //Her har vi lavet getinstance metoder for at kunne sørge for at der ikke blev instansieret en objekt udenfor klassen.
    public static LaundrySingleton getInstance() {
        if(instance == null){
            instance = new LaundrySingleton();
        }
        return instance;

    }

    private LaundrySingleton(){

    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        LaundrySingleton.getInstance().buildLaundry();

        System.out.println("Laundry:\n" + LaundrySingleton.getInstance());

        System.out.println("\n" + LaundrySingleton.getInstance().getMachine(0).getModel() + ":");
        System.out.print(LaundrySingleton.getInstance().getProgName(0, 5) + "\t");
        System.out.printf("%.2f%n", LaundrySingleton.
getInstance().getPrice(0, 5));

        System.out.println("\n" + LaundrySingleton.getInstance().getMachine(1).getModel() + ":");
        System.out.print(LaundrySingleton.getInstance().getProgName(1, 5) + "\t");
        System.out.printf("%.2f%n", LaundrySingleton.getInstance().getPrice(1, 5));

        System.out.println("\n" + LaundrySingleton.getInstance().getMachine(2).getModel() + ":");
        System.out.print(LaundrySingleton.getInstance().getProgName(2, 5) + "\t");
        System.out.printf("%.2f%n", LaundrySingleton.getInstance().getPrice(2, 5));

    }

    @Override
    public String toString() {
        return "laundryMachines:\n" + Arrays.toString(laundryMachines);
    }

    public LaundryMachine getMachine(int index) {
        if (index < laundryMachines.length)
            return laundryMachines[index];
        else {
            System.out.println("Maskine findes ikke!");
            return null;
        }
    }

    public double getPrice(int machine, int program) {
        LaundryMachine lm = getMachine(machine);
        if (lm != null) {
            return lm.getPrice(program);
        }
        return 0.0;
    }

    public String getProgName(int machine, int prog) {
        LaundryMachine lm = getMachine(machine);
        if (lm != null) {
            return lm.getProgName(prog);
        }
        return "Maskine findes ikke!";

    }

    public void buildLaundry() {
        laundryMachines = new LaundryMachine[3];

        WashingMachine w1 = new WashingMachine("Vaskemaskine max 6 kg");
        w1.addProgram(1, WASH_PROGRAM_1, WASH_PRICE_1);
        w1.addProgram(2, WASH_PROGRAM_2, WASH_PRICE_2);
        w1.addProgram(3, WASH_PROGRAM_3, WASH_PRICE_3);
        w1.addProgram(4, WASH_PROGRAM_4, WASH_PRICE_4);
        w1.addProgram(5, WASH_PROGRAM_5, WASH_PRICE_5);
        w1.addProgram(6, WASH_PROGRAM_6, WASH_PRICE_6);
        w1.addProgram(7, WASH_PROGRAM_7, WASH_PRICE_7);

        laundryMachines[0] = w1;

        WashingMachine w2 = new WashingMachine("Vaskemaskine max 12 kg");
        w2.addProgram(1, WASH_PROGRAM_1, WASH_PRICE_1 * FACTOR_12);
        w2.addProgram(2, WASH_PROGRAM_2, WASH_PRICE_2 * FACTOR_12);
        w2.addProgram(3, WASH_PROGRAM_3,
WASH_PRICE_3 * FACTOR_12);
        w2.addProgram(4, WASH_PROGRAM_4, WASH_PRICE_4 * FACTOR_12);
        w2.addProgram(5, WASH_PROGRAM_5, WASH_PRICE_5 * FACTOR_12);
        w2.addProgram(6, WASH_PROGRAM_6, WASH_PRICE_6 * FACTOR_12);
        w2.addProgram(7, WASH_PROGRAM_7, WASH_PRICE_7 * FACTOR_12);

        laundryMachines[1] = w2;

        TumbleDryer t = new TumbleDryer("Tørre Tumbler");
        t.setPrice(TUMBLE_PRICE);

        laundryMachines[2] = t;

    }


}

```

```java
package laundry_facade;

public class TumbleDryer implements LaundryMachine {

    private final double pricePerMinute;
    private final String model;

    //Her skal vi bare bruge this-keyword fra OPP-timen.
    public TumbleDryer(String model) {
        this.model = model;
    }

    //Her skal vi bare bruge vores this-keyword fra OOP.
    public void setPrice(double pricePerMinute) {
        this.pricePerMinute = pricePerMinute;
    }

    //Her skal vi bare implemente return model ligesom i OOP.
    @Override
    public String getModel() {
        return model;
    }

    //Program (prog) definerer tiden, og prisen køres udefra tidsforbruget.
    //Derfor er det oplagt at gange prisen med tidsforbruget som er (prog).
    @Override
    public double getPrice(int prog) {
        return pricePerMinute * prog;
    }

    //Dette er udtrykket for programmet, mens vaskemaskinene kører.
    @Override
    public String getProgName(int prog) {
        return "Tørring i" + prog + "minutter";
    }

    @Override
    public String toString() {
        return getModel() + " Minutpris: " + pricePerMinute + "\n";
    }

}

```

```java
package laundry_facade;

import java.util.HashMap;
import java.util.Map;

public class WashingMachine implements LaundryMachine {

    private final Map<Integer, WashProgram> progMap;
    private final String model;

    public WashingMachine(String model) {
        progMap = new HashMap<>();
        this.model = model;
    }

    public void addProgram(int code, String name, double price) {
        progMap.put(code, new WashProgram(name, price));

    }

    @Override
    public String getModel() {
        return model;
    }


    @Override
    public double getPrice(int prog) {
        return progMap.get(prog).getPrice();
    }

    @Override
    public String getProgName(int prog) {
        return progMap.get(prog).getName();
    }

    @Override
    public String toString() {
        return getModel() + "\n" + progMap + "\n";
    }


}

```

Her har jeg tilføjet opgavebesvarelsen fra Gitlab som vores Instruktør har lavet. Jeg har haft udfordringer med, at lave den selv eftersom IntelliJ ikke kunne acceptere IntelliJ imports fra Applikation osv.

APP-Klassen

```
package com.example.exercise;

//Dette her er en kopi fra Gitlab, som jeg havde problemer med at lave inde på
IntelliJ.
//Jeg kan ikke forklare årsagen, men jeg tror at fordi vores opgave var ikke
oprettet som JavaFX-Program i starten så jeg kunne ikke løse opgaven selv.

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.io.IOException;

public class App extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(App.class.getResource("primary.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 600, 400);
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch();
    }
}
```

PrimaryController-Klassen

```
package com.example.exercise;

import facade.Facade;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.*;
import javafx.scene.input.KeyEvent;
import laundry_facade.LaundrySingleton;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;

public class PrimaryController {
    @FXML
    private Label machineLabel;
    @FXML
    private RadioButton wash6RadioButton;
    @FXML
    private ToggleGroup options;
    @FXML
    private RadioButton wash12RadioButton;
    @FXML
    private RadioButton dryRadioButton;
    @FXML
    private TextField resultTF;
    @FXML
    private TextField programTF;
    @FXML
    private Button fillButton;
    @FXML
    private Button fillUniqueButton;
    @FXML
    private TextField sizeTF;
    @FXML
    private TextField maxTF;
    @FXML
    private TextArea textArea;
    @FXML
    private TextField divTF;
    @FXML
    private Button sumButton;

    int program;
    private Facade facade;

    private final LaundrySingleton instance = LaundrySingleton.getInstance();

    @FXML
    public void initialize(){
        facade = new Facade();
        instance.buildLaundry();
    }


    public void buttonHandler(ActionEvent actionEvent) {
        int size = Integer.parseInt(sizeTF.getText());
        int max = Integer.parseInt(maxTF.getText());

        if(actionEvent.getSource() == fillButton){
            textArea.appendText(Arrays.toString(facade.fillArray(size, max)) +
"\n");
        }
        if(actionEvent.getSource() == fillUniqueButton){
            textArea.appendText(Arrays.toString(facade.fillUniqueArray(size,
max))+ "\n");
        }
        if(actionEvent.getSource() == sumButton){
            int div = Integer.parseInt(divTF.getText());
            textArea.appendText("Sum of divisor " + div + ": " +
facade.sumOfDivisors(div)+ "\n");
        }
    }

    public void washHandler() {
        try {
            program = Integer.parseInt(programTF.getText());
        } catch (NumberFormatException ex){
            System.out.println("Not a number");
        }

        Integer[] integers = {1, 2, 3, 4, 5, 6, 7};
        ArrayList<Integer> legalPrograms = new ArrayList<>();
        Collections.addAll(legalPrograms, integers);

        if (options.getSelectedToggle() == wash6RadioButton){
            machineLabel.setText(instance.getMachine(0).getModel() +":");
            if(legalPrograms.contains(program)) {
                resultTF.setText(instance.getProgName(0, program) + "  :  " +
String.format("%.2f", instance.getPrice(0, program)));
            }
            else{
                resultTF.setText("Program " + program + " does not exist: " + -
1);
            }
        }

        if (options.getSelectedToggle() == wash12RadioButton){
            machineLabel.setText(instance.getMachine(1).getModel() +":");
            if(legalPrograms.contains(program)) {
                resultTF.setText(instance.getProgName(1, program) + "  :  " +
String.format("%.2f", instance.getPrice(1, program)));
            }
            else{
                resultTF.setText("Program " + program + " does not exist: " + -
1);
            }
        }
        if (options.getSelectedToggle() == dryRadioButton){
            machineLabel.setText(instance.getMachine(2).getModel() +":");
            resultTF.setText(instance.getProgName(2, program) + "  :  " +
String.format("%.2f", instance.getPrice(2, program)));
        }
    }
}
```