Prof. Dr. Th. Letschert

TECHNISCHE HOCHSCHULE MITTELHESSEN
MNI Mathematik, Naturwissenschaften und Informatik

NVP – Nebenläufige und Verteilte Programme

Aufgabenblatt 3

Aufgabe 1

Betrachten Sie folgende JavaFX-Anwendung im MVC- / Ereignis-getriebenen Stil:

```
package blatt_03.aufgabe_1_1

import javafx.application.Application
import javafx.scene.Scene
import javafx.scene.control.{Label, TextField}
import javafx.scene.layout.GridPane
import javafx.stage.Stage
import javafx.beans.value.{ObservableValue, ObservableValueBase, ChangeListener}

class SumView extends GridPane with ChangeListener[Int] {
  val tfInput_1 = new TextField("")
  val tfInput_2 = new TextField("")
  val tfOutput = new TextField("")

  add(tfInput_1, 0, 0)
  add(tfInput_2, 1, 0)
  add(tfOutput, 1, 1)
  add(new Label("Sum"), 0, 1)

  tfInput_1.textProperty().addListener(
    (obs , oldValue, newValue) =>
       SumControl.set_a(newValue)
    )

  tfInput_2.textProperty().addListener(
    (obs: ObservableValue[_ >: String], oldValue: String, newValue: String) =>
       SumControl.set_b(newValue)
    )

  override def changed( obs: ObservableValue[_ <: Int],
                  oldValue: Int,
                  newValue: Int) = {
    tfOutput.setText(""+newValue)
  }

}

object SumModel extends ObservableValueBase[Int] {
  private var value_a = 0
  private var value_b = 0

  def setA(v: Int) : Unit = {
    value_a = v
```

```
    fireValueChangedEvent()
  }
  def setB(v: Int) : Unit = {
    value_b = v
    fireValueChangedEvent()
  }

  def getValue(): Int = value_a + value_b
}

object SumControl {
  def set_a(str: String): Unit = try {
    val a = str.toInt
    SumModel.setA(a)
  } catch {
    case _:NumberFormatException => /* ignore */
  }
  def set_b(str: String): Unit = try {
    val b = str.toInt
    SumModel.setB(b)
  } catch {
    case _:NumberFormatException => /* ignore */
  }
}

class SumApplication extends Application {
  override def start(primaryStage: Stage): Unit = {
    primaryStage.setTitle("Sum App");
    val view = new SumView
    SumModel.addListener(view)
    val scene = new Scene(view);
    primaryStage.setScene(scene);
    primaryStage.show();
  }
}

object SumApp {
  def main(args: Array[String]) {
    Application.launch(classOf[SumApplication], args: _*)
  }
}
```

Bringen Sie die Anwendung in einen Datenfluss / reaktiven Stil indem Sie die Möglichkeiten der reaktiven Programmierung von JavaFX nutzen.

Aufgabe 2

Implementieren Sie eine GUI-Anwendung zur Fraktorisierung mit folgenden Eigenschaften:

- Die Faktorisierurng wird mit einen Knopf gestartet und kann mit einem Knopf abgebrochen werden.
- Alle Primfaktoren sollen am Ende der Berechnung angezeigt werden. Zusätzlich soll während der Faktorisierung jeder gefundene Primfaktor sofort nach seiner Entdeckung angezeigt werden.
- Ein Fortschrittsbalken soll den Fortschritt der Faktorisierung anzeigen.

Hinweis: Nutzen Sie *javafx.concurrent.Task*.