WOJSKOWA AKADEMIA TECHNICZNA

im. Jarosława Dąbrowskiego

PROGRAMOWANIE

WSPÓŁBIEŻNE



SPRAWOZDANIE Z

PROJEKTU

Temat : Symulacja ruchu na stoku narciarskim

Prowadzący: dr inż. Jarosław Rulka

Wykonał: Mikołaj Szabrański

Grupa: WCY19IY1S1

Data wykonania: 18.06.2021

1. Treść zadania

Podstawowe wymagania:

1. liczba procesów sekwencyjnych powinna być dobrana z wyczuciem tak, aby zachować czytelność interfejsu i jednocześnie umożliwić zobrazowanie reprezentatywnych przykładów,
2. kod źródłowy programu musi być tak skonstruowany, aby można było „swobodnie” modyfikować liczbę procesów sekwencyjnych (za wyjątkiem zadań o ściśle określonej liczbie procesów),
3. graficzne zobrazowanie działania procesów współbieżnych,
4. odczyt domyślnych danych wejściowych ze sformatowanego, tekstowego pliku danych (xml, properties, inne),
5. możliwość modyfikacji danych wejściowych poprzez GUI.

Problem do rozwiązania: symulacja ruchu na stoku narciarskim.

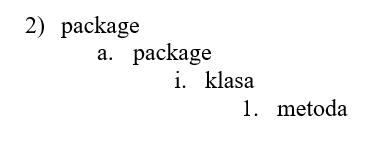
1. Założenia:
   1. 3 trasy zjazdowe obsługiwane przez 3 wyciągi:
      1. wyciąg ze stacji bazowej na szczyt;
      2. wyciąg ze stacji bazowej do stacji w połowie stoku;
      3. wyciąg ze stacji w połowie stoku na szczyt;
   2. Każdy wyciąg ma określoną pojemność i częstotliwość kursowania;
   3. Po upływie pewnego czasu dany wyciąg jest zamykany w celu prac serwisowych.
2. Opis problemu
3. Przyjęte założenia
   1. Narciarze wjeżdżają „na górę” i zjeżdżają na nartach. Nikt nie wraca kolejką górską na dół.
   2. Jeśli liczba osób w kolejce jest nie mniejsza od pojemności wyciągu to wsiada do niego dokładnie tyle osób ile sugeruje pojemność. Jeśli osób w kolejce jest mniej niż może jechać wyciągiem maksymalnie to w takim przypadku jadą one wszystkie.
4. Współdzielone zasoby
5. Wyróżnione punkty synchronizacji
   1. Metoda add() w klasie Queue
   2. Metoda subNumOfPpl (int sub) w klasie Queue
6. Wykaz obiektów synchronizacji
   1. Queue.numOfPpl – liczba osób w kolejce do wyciągu
   2. Queue.current – liczba osób jadących obecnie kolejką linową
7. Wykaz procesów sekwencyjnych

Ropeway.run():

* 1. wsiadanie do wyciągu
  2. rozpoczęcie ruchu wyciągu po odczekaniu ustalonego czasu
  3. jazda na „górę”
  4. zatrzymanie się wyciągu
  5. wysiadanie i odczekanie
  6. powrót na „dół”

1. Struktura projektu

Legenda:



Struktura:

1. com
   1. sample
      1. Controller

Metody:

* + - 1. fromFile – odzczyt wartości danych z pliku
      2. initialize – ustawienie wartości początkowych progressBar’ów oraz utworzenie i uruchomienie wątków
      3. bazaSzczyt – akcja przycisku inkrementująca liczbę osób w kolejce do wyciągu ze stacji bazowej na szczyt
      4. bazaSrodek - akcja przycisku inkrementująca liczbę osób w kolejce do wyciągu ze stacji bazowej do stacji w połowie stoku
      5. srodekSzczyt - akcja przycisku inkrementująca liczbę osób w kolejce do wyciągu ze stacji w połowie stoku na szczyt
    1. Main

Metody:

* + - 1. start – utworzenie okna GUI
      2. main – launch(args)
    1. sample.fxml - wygląd początkowy GUI
  1. Stok
     1. Queue

Metody

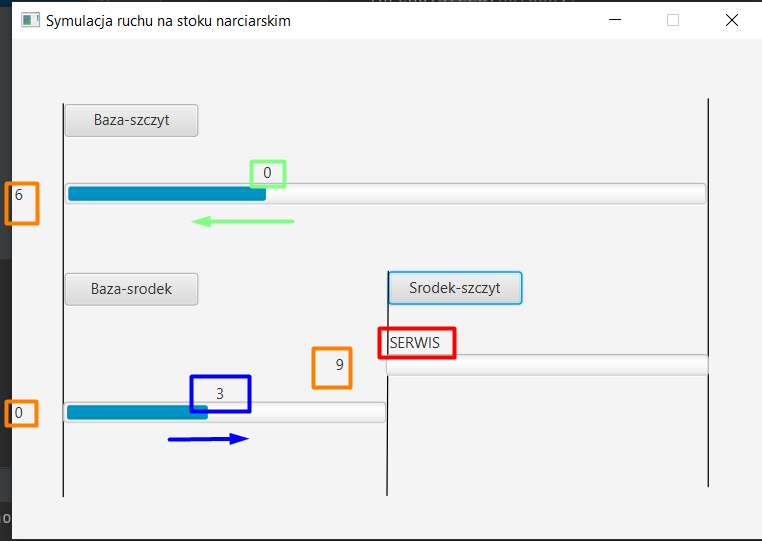
* + - 1. update – aktualizacja liczby osób w kolejce w GUI
      2. update2 - aktualizacja liczby osób w wyciągu w GUI
      3. service – wyświetlenie informacji o serwisie
      4. add – inkrementacja liczby osób w kolejce
      5. subNumOfPpl – pomniejszenie liczby osób w kolejce po tym jak część z nich wsiadła do wyciągu
    1. Ropeway
       1. run – odpowiada za poruszanie się kolejek oraz ich oczekiwanie i przerwę na serwis

1. dane.txt – plik z wartościami danych

Struktura pliku

* + - 1. wiersz – ładowności kolejek
      2. wiersz – czasy kursów wyciągów (w jednym kierunku)
      3. wiersz – czasy serwisowania wyciągów
      4. wiersz – czasy oczekiwania na odjazd po przyjeździe
      5. wiersz - liczba kursów po których odbywa się serwis

1. GUI



Liczba osób w kolejce do wyciągu.

Komunikat o serwisowaniu wyciągu.

Nad wyciągiem wyświetla się liczba osób które nią jadą:

Jeśli kolejka jedzie w górę („w prawo”) to jest to liczba wynikająca z zadania, a jeśli w dół („w lewo”) to jest to 0 zgodnie z przyjętym założeniem.

1. Listing programu
   1. com
      1. sample
         1. Controller

package com.sample;  
  
import com.Stok.Queue;  
import com.Stok.Ropeway;  
import javafx.application.Platform;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.Initializable;  
import javafx.scene.control.Label;  
import javafx.scene.control.ProgressBar;  
import java.io.File;  
import java.io.FileNotFoundException;  
import java.net.URL;  
import java.util.Arrays;  
import java.util.ResourceBundle;  
import java.util.Scanner;  
import java.util.concurrent.atomic.AtomicInteger;  
  
  
public class Controller implements Initializable {  
 Thread[] UDM = new Thread[3];  
 Queue[] que = new Queue[3];  
 AtomicInteger pplInQueue[] = { //chętni do wjechania poszczególnymi kolejkami  
 new AtomicInteger(15),  
 new AtomicInteger(19),  
 new AtomicInteger(16)  
 };  
 int capacity[] = {0,0,0};//ładowności kolejek  
 int travelTime[] = {0,0,0};//czas kursu wyciągu  
 int serviceTime[] = {0,0,0};//czas serwisowania wyciągu  
 int holdOn[] = {0,0,0};//czas oczekiwania po przyjezdzie  
 int toService = 1; //liczba przejazdów do serwisowania  
 int numOfObj = 3; //liczba wyciągów  
  
 @FXML  
 private ProgressBar progressBar1;  
 @FXML  
 private ProgressBar progressBar2;  
 @FXML  
 private ProgressBar progressBar3;  
 @FXML  
 private Label label1 = new Label("" + pplInQueue[0]);  
 @FXML  
 private Label label2 = new Label("" + pplInQueue[1]);  
 @FXML  
 private Label label3 = new Label("" + pplInQueue[2]);  
 @FXML  
 private Label label1Move;  
 @FXML  
 private Label label2Move;  
 @FXML  
 private Label label3Move;  
  
 private void fromFile() throws FileNotFoundException {  
 File file = new File("./src/dane.txt");  
 Scanner scanner = new Scanner(file);  
 for (int i = 0; i < 3; i++) {  
 capacity[i] = scanner.nextInt();  
 }  
 for (int i = 0; i < 3; i++) {  
 travelTime[i] = scanner.nextInt();  
 }  
 for (int i = 0; i < 3; i++) {  
 serviceTime[i] = scanner.nextInt();  
 }  
 for (int i = 0; i < 3; i++) {  
 holdOn[i] = scanner.nextInt();  
 }  
 toService = scanner.nextInt();  
 }  
  
 @Override  
 public void initialize(URL url, ResourceBundle rb) {  
 progressBar1.setProgress(0.0);  
 progressBar2.setProgress(0.0);  
 progressBar3.setProgress(0.0);  
  
 try {  
 fromFile();  
 } catch (FileNotFoundException e) {}  
  
 Platform.*runLater*(() -> {  
 que[0] = new Queue(0, pplInQueue[0], capacity[0], label1, label1Move);  
 que[1] = new Queue(1, pplInQueue[1], capacity[1], label2, label2Move);  
 que[2] = new Queue(2, pplInQueue[2], capacity[2], label3, label3Move);  
  
 UDM[0] = new Ropeway(que[0], travelTime[0], serviceTime[0], holdOn[0], toService, progressBar1, label1, label1Move);  
 UDM[1] = new Ropeway(que[1], travelTime[1], serviceTime[1], holdOn[1], toService, progressBar2, label2, label2Move);  
 UDM[2] = new Ropeway(que[2], travelTime[2], serviceTime[2], holdOn[2], toService, progressBar3, label3, label3Move);  
 Arrays.*stream*(UDM).forEach(ropeway -> ropeway.setDaemon(true));  
  
 for (int i = 0; i < numOfObj; i++) {  
 UDM[i].start();  
 }  
 });  
 }  
  
 public void bazaSczyt(ActionEvent e) {  
 que[0].add();  
 }  
  
 public void bazaSrodek(ActionEvent e) {  
 que[1].add();  
 }  
  
 public void srodekSczyt(ActionEvent e) {  
 que[2].add();  
 }  
}

* + - 1. Main

package com.sample;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
  
public class Main extends Application {  
  
 @Override  
 public void start(Stage stage) throws Exception {  
 Parent root = FXMLLoader.*load*(getClass().getResource("sample.fxml"));  
 Scene scene = new Scene(root);  
 stage.setScene(scene);  
 stage.setTitle("Symulacja ruchu na stoku narciarskim");  
 stage.setResizable(false);  
 stage.show();  
 }  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
}

* + - 1. Sample.fxml

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.scene.control.Button?>  
<?import javafx.scene.control.Label?>  
<?import javafx.scene.control.ProgressBar?>  
<?import javafx.scene.layout.AnchorPane?>  
<?import javafx.scene.shape.Line?>  
  
<AnchorPane prefHeight="400.0" prefWidth="600.0" xmlns:fx="http://javafx.com/fxml" fx:controller="com.sample.Controller">  
 <children>  
 <Button layoutX="42.0" layoutY="52.0" mnemonicParsing="false" onAction="#bazaSczyt" prefHeight="26.0" prefWidth="107.0" text="Baza-szczyt" />  
 <Button layoutX="42.0" layoutY="187.0" mnemonicParsing="false" onAction="#bazaSrodek" prefHeight="26.0" prefWidth="107.0" text="Baza-srodek" />  
 <Button layoutX="301.0" layoutY="186.0" mnemonicParsing="false" onAction="#srodekSczyt" prefHeight="26.0" prefWidth="107.0" text="Srodek-szczyt" />  
 <Line endX="-100.0" endY="-314.0" layoutX="141.0" layoutY="366.0" startX="-100.0" />  
 <Line endX="-100.0" endY="-310.0" layoutX="657.0" layoutY="358.0" startX="-100.0" />  
 <Line endX="-99.0" endY="-179.00001525878906" layoutX="400.0" layoutY="365.0" startX="-100.0" />  
 <ProgressBar fx:id="progressBar1" layoutX="42.0" layoutY="115.0" prefHeight="18.0" prefWidth="513.0" progress="0.0" />  
 <ProgressBar fx:id="progressBar2" layoutX="41.0" layoutY="290.0" prefHeight="18.0" prefWidth="258.0" progress="0.0" />  
 <ProgressBar fx:id="progressBar3" layoutX="299.0" layoutY="252.0" prefHeight="18.0" prefWidth="258.0" progress="0.0" />  
 <Label fx:id="label1" layoutX="2.0" layoutY="116.0" prefHeight="18.0" prefWidth="40.0" text="Label" />  
 <Label fx:id="label2" layoutX="2.0" layoutY="290.0" prefHeight="18.0" prefWidth="40.0" text="Label" />  
 <Label fx:id="label3" layoutX="259.0" layoutY="252.0" prefHeight="18.0" prefWidth="40.0" text="Label" />  
 <Label fx:id="label1Move" layoutX="42.0" layoutY="98.0" text="Label" />  
 <Label fx:id="label2Move" layoutX="42.0" layoutY="275.0" text="Label" />  
 <Label fx:id="label3Move" layoutX="302.0" layoutY="234.0" text="Label" />  
 </children>  
</AnchorPane>

* + 1. Stok
       1. Queue

package com.Stok;  
  
import javafx.application.Platform;  
import javafx.scene.control.Label;  
  
import java.util.concurrent.atomic.AtomicInteger;  
import java.util.concurrent.locks.Lock;  
import java.util.concurrent.locks.ReentrantLock;  
  
public class Queue {  
 AtomicInteger numOfPpl;  
 int capacity;  
 int current;  
 int nr;  
 Label lab;  
 Label labMove;  
  
 private final Lock lock = new ReentrantLock();  
  
 public Queue(int nr, AtomicInteger numOfPpl, int capacity, Label lab, Label labMove) {  
 this.nr = nr;  
 this.numOfPpl = numOfPpl;  
 this.capacity = capacity;  
 this.lab = lab;  
 this.labMove = labMove;  
 }  
  
 public void update() {  
 Platform.*runLater*(()->{  
 lab.setText(Integer.*toString*(numOfPpl.get()));  
 });  
 }  
  
 public void update2() {  
 Platform.*runLater*(()->{  
 labMove.setText(Integer.*toString*(current));  
 });  
 }  
  
 public void service(){  
 Platform.*runLater*(()->{  
 labMove.setText("SERWIS");  
 });  
 }  
  
 public synchronized void add() {  
 numOfPpl.incrementAndGet();  
 update();  
 }  
  
 public synchronized void subNumOfPpl(int sub) {  
 if (numOfPpl.get() > sub) {  
 current = sub;  
 numOfPpl.set(numOfPpl.get() - capacity);  
 } else {  
 current = numOfPpl.get();  
 numOfPpl.set(0);  
 }  
 update();  
 update2();  
 }  
}

* + - 1. Ropeway

package com.Stok;  
  
import javafx.application.Platform;  
import javafx.scene.control.Label;  
import javafx.scene.control.ProgressBar;  
  
import java.util.concurrent.locks.Lock;  
import java.util.concurrent.locks.ReentrantLock;  
  
public class Ropeway extends Thread {  
 private final Queue qu;  
 private final int travelTime;  
 private final int capacity;  
 private final int serviceTime;  
 private final int holdOn;  
 private final int toService;  
 private double x;  
 private double addLength;  
 ProgressBar progressBar;  
 Label label;  
 Label labelMove;  
  
 private final Lock lock = new ReentrantLock();  
  
 public Ropeway(Queue qu, int travelTime, int serviceTime, int holdOn, int toService, ProgressBar progressBar, Label label, Label labelMove) {  
 this.qu = qu;  
 this.holdOn = holdOn;  
 this.serviceTime = serviceTime;  
 this.travelTime = travelTime;  
 this.toService = toService;  
 this.capacity = qu.capacity;  
 this.progressBar = progressBar;  
 this.label = label;  
 this.labelMove = labelMove;  
 this.x = labelMove.getLayoutX();  
 this.addLength = (progressBar.getWidth() / 100);  
 }  
  
 public void run() {  
 while (true) {  
 for (int i = 0; i < toService; i++) {  
 lock.lock();  
 try {  
 //wsiadanie  
 qu.subNumOfPpl(capacity);  
 //czekanie na odjazd  
 Thread.*sleep*(holdOn);  
 //jazda w górę  
 for (int j = 0; j < 100; j++) {  
 try {  
 progressBar.setProgress(j / 100.0);  
 Platform.*runLater*(()->labelMove.setLayoutX(x+=addLength));  
 Thread.*sleep*(travelTime / 100);  
 } catch (InterruptedException ie) {  
 }  
 }  
 //odczekanie i powrót  
 qu.current = 0;  
 qu.update2();  
 Thread.*sleep*(holdOn);  
 for (int j = 100; j > 0; j--) {  
 try {  
 progressBar.setProgress(j / 100.0);  
 Platform.*runLater*(()->labelMove.setLayoutX(x-=addLength));  
 Thread.*sleep*(travelTime / 100);  
 } catch (InterruptedException ie) {  
 }  
 }  
 } catch (InterruptedException e) {  
 } finally {  
 lock.unlock();  
 }  
 }  
 //SERWIS  
 try {  
 qu.service();  
 Thread.*sleep*(serviceTime);  
 qu.update2();  
 } catch (InterruptedException e) {  
 }  
 }  
 }  
}

* 1. dane.txt

3 4 2

3000 2000 1500

10000 12000 11000

300 200 200

5