WOJSKOWA AKADEMIA TECHNICZNA

im. Jarosława Dąbrowskiego

WYDZIAŁ CYBERNETYKI

Laboratorium

Przedmiot: Programowanie Współbieżne

Wykonał: Filip Stańczak

Grupa: WCY19IJ3S1

Data wykonania: 16.01.2023r.

Prowadzący: dr inż. Jarosław Rulka

Treść zadania:

Zadanie nr: **PW-1/2022**

Język implementacji: **Java**
Środowisko implementacyjne: **Eclipse, Intelij IDEA, Netbeans**
Termin wykonania: **ostatnie zajęcia**

Podstawowe wymagania:
a. liczba procesów sekwencyjnych powinna być dobrana z wyczuciem tak, aby zachować czytelność interfejsu i jednocześnie umożliwić zobrazowanie reprezentatywnych przykładów,
b. kod źródłowy programu musi być tak skonstruowany, aby można było „swobodnie" modyfikować liczbę procesów sekwencyjnych (za wyjątkiem zadań o ściśle określonej liczbie procesów),
c. graficzne zobrazowanie działania procesów współbieżnych,
d. odczyt domyślnych danych wejściowych ze sformatowanego, tekstowego pliku danych (xml, properties, inne),
e. możliwość modyfikacji danych wejściowych poprzez GUI.

Sprawozdanie (w formie elektronicznej) powinno zawierać następujące elementy:
1) stronę tytułową,
2) niniejszą treść zadania,
3) syntetyczny opis problemu – w tym wszystkie przyjęte założenia,
4) wykaz współdzielonych zasobów,
5) wykaz wyróżnionych punktów synchronizacji,
6) wykaz obiektów synchronizacji,
7) wykaz procesów sekwencyjnych,
8) listing programu.

Problem do rozwiązania:
Transport cegieł.
Założenia:
Przy taśmie transportowej pracuje trzech pracowników oznaczonych przez P1, P2 i P3. Pracownicy wrzucają na taśmę cegły o masach odpowiednio 1, 2 i 3 jednostki. Na końcu taśmy stoi ciężarówka o ładowności C jednostek, którą należy zawsze załadować do pełna. Wszyscy pracownicy starają się układać cegły na taśmie najszybciej jak to możliwe. Taśma może przetransportować w danej chwili maksymalnie K sztuk cegieł. Jednocześnie jednak taśma ma ograniczony udźwig: maksymalnie M jednostek masy, tak, że niedopuszczalne jest położenie np. samych tylko cegieł najcięższych (3K>M). Po zapełnieniu ciężarówki na jej miejsce pojawia się natychmiast nowa o takich samych parametrach. Cegły „zjeżdżające" z taśmy muszą od razu trafić na samochód dokładnie w takiej kolejności jak zostały położone na taśmie.

Przyjęte założenia:

Istnieje taśma transportowa na której trzech pracowników układa cegły. Na końcu taśmy znajduje się ciężarówka. Każdy pracownik układa cegły jak najszybciej, czas potrzebny na przeniesienie cegły jest zależny od jej masy. Taśma ma ograniczoną nośność a także maksymalną ilość cegieł znajdujących się na niej w danym momencie. Z taśmy cegły lądują do ciężarówki która ma określoną pojemność. W momencie jej całkowitego zapełnienia podstawiona jest nowa ciężarówka a poprzednia odjeżdża z towarem. Jeśli masa cegieł znajdujących się na taśmie wypełni ciężarówkę to pracownicy muszą poczekać aż dana ciężarówka się zapełni i podjedzie nowa. Ciężarówka nie może odebrać cegły w momencie dokładania nowej na taśmę. Pracownik nie może położyć cegły na taśmie gdy ciężarówka zabiera cegłę z taśmy.

Wykaz współdzielonych zasobów:

Taśma transportowa:

```
public class beltOfBricks {
    5 usages
    public LinkedList<Brick> bricksList = new LinkedList<~>();
    1 usage  Filip Stańczak
    public synchronized void add(Brick brick) { bricksList.add(brick); }
    1 usage  Filip Stańczak
    public synchronized Brick remove() { return bricksList.removeFirst(); }
    1 usage  Filip Stańczak
    public Boolean isEmpty() { return bricksList.isEmpty(); }
    2 usages  Filip Stańczak
    public int getSize() { return bricksList.size(); }
}
```

Ciężarówka:

```
public class Truck {
    5 usages
    public int container;
    2 usages
    public int id = 1;
    2 usages
    public static int nextId = 1;
    2 usages  Filip Stańczak
    public Truck(int container) {
        this.container = container;
        this.id = nextId;
        nextId++;
    }
    1 usage  Filip Stańczak
    public void loadTruck() {
        synchronized (ConvoyerBelt.bricksOnBelt) {
            if (!ConvoyerBelt.bricksOnBelt.isEmpty()) {
                Brick brick = ConvoyerBelt.bricksOnBelt.remove();
                Filip Stańczak
                Platform.runLater(new Runnable() {
                    Filip Stańczak
                    @Override
                    public void run() { ConvoyerBelt.belt.getItems().remove( index: 0); }
                });
                container += brick.mass;
                ConvoyerBelt.updateLabels( truck: this);
                //System.out.println("Truck id: "+truck.id+" container: "+truck.container);
                if (container == ConvoyerBelt.TRUCK_CAPACITY) {
                    Truck newTruck = new Truck( container: 0);
                    ConvoyerBelt.P1.setTruck(newTruck);
                    ConvoyerBelt.P2.setTruck(newTruck);
                    ConvoyerBelt.P3.setTruck(newTruck);
                    //System.out.println("Truck is full, new truck is ready.");
                }
            }
        }
    }
}
```

Wykaz wyróżnionych punktów synchronizacji:

Położenie cegły na taśmę produkcyjną:

```
private void layBrick(int mass) {
    synchronized (bricksOnBelt) {
        int weight = 0;
        for(Brick brick : bricksOnBelt.bricksList) {
            weight += brick.mass;
        }
        //System.out.println("Belt weight: "+weight+" Worker P"+mass+" working..");
        if (bricksOnBelt.getSize() < BELT_CAPACITY &&
                weight + mass <= BELT_WEIGHT_LIMIT &&
                truck.container + weight + mass <= TRUCK_CAPACITY) {
            Brick brick = new Brick(brickType);
            bricksOnBelt.add(brick);
            Filip Stańczak
            Platform.runLater(new Runnable() {
                Filip Stańczak
                @Override
                public void run() { ConvoyerBelt.belt.getItems().add(String.valueOf(mass)); }
            });
            ConvoyerBelt.updateLabels(truck);
            //System.out.println("brick layed by P"+ brickType+", belt weight: "+(weight+mass)+"");
        }else {
            truck.loadTruck();
        }
        try{
            Thread.sleep( millis: 200);
        }
        catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Zdjęcie cegły z taśmy i załadowanie do ciężarówki:

```
public void loadTruck() {
    synchronized (ConvoyerBelt.bricksOnBelt) {
        if (!ConvoyerBelt.bricksOnBelt.isEmpty()) {
            Brick brick = ConvoyerBelt.bricksOnBelt.remove();
            Filip Stańczak
            Platform.runLater(new Runnable() {
                Filip Stańczak
                @Override
                public void run() { ConvoyerBelt.belt.getItems().remove( index: 0); }
            });
            container += brick.mass;
            ConvoyerBelt.updateLabels( truck: this);
            //System.out.println("Truck id: "+truck.id+" container: "+truck.container);
            if (container == ConvoyerBelt.TRUCK_CAPACITY) {
                Truck newTruck = new Truck( container: 0);
                ConvoyerBelt.P1.setTruck(newTruck);
                ConvoyerBelt.P2.setTruck(newTruck);
                ConvoyerBelt.P3.setTruck(newTruck);
                //System.out.println("Truck is full, new truck is ready.");
            }
        }
    }
}
```

Wykaz obiektów synchronizacji:

Semafor pozwalający na skorzystanie z taśmy produkcyjnej:

```
public static Semaphore lock = new Semaphore( permits: 1);
```

Wykaz procesów sekwencyjnych:

-Wątek główny sterujący całą aplikacją, powołujący do życia pracowników oraz wyświetlający symulację

-Wątki pracowników

Listing programu:

Truck:

```
package com.example.projektpw;

import javafx.application.Platform;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;

public class Truck {
    public int container;
    public int id = 1;
    public static int nextId = 1;
    public Truck(int container) {
        this.container = container;
        this.id = nextId;
        nextId++;
    }
    public void loadTruck() {
        synchronized (Main.bricksOnBelt) {
            if (!Main.bricksOnBelt.isEmpty()) {
                Brick brick = Main.bricksOnBelt.remove();
                Platform.runLater(new Runnable() {
                    @Override
                    public void run() {
                        Main.belt.getItems().remove(0);
                    }
                });
                container += brick.mass;
                Main.updateLabels(this);
                //System.out.println("Truck id: "+truck.id+" container:
"+truck.container);
                if (container == Main.TRUCK_CAPACITY) {
                    Truck newTruck = new Truck(0);
                    Main.P1.setTruck(newTruck);
                    Main.P2.setTruck(newTruck);
                    Main.P3.setTruck(newTruck);
                    //System.out.println("Truck is full, new truck is
ready.");
                }
            }
        }
    }
}
```

Convoyer Belt:

```
package com.example.projektpw;

import java.util.LinkedList;

public class ConvoyerBelt {
    public LinkedList<Brick> bricksList = new LinkedList<Brick>();
    public synchronized void add(Brick brick){
        bricksList.add(brick);
    }
    public synchronized Brick remove(){
        return bricksList.removeFirst();
    }
    public Boolean isEmpty(){
        return bricksList.isEmpty();
    }
    public int getSize(){
        return bricksList.size();
    }
}
```

Worker:

```
package com.example.projektpw;

import javafx.application.Platform;

import static com.example.projektpw.Main.bricksOnBelt;

public class Worker implements Runnable{
    public String name;
    public int brickType;
    public int BELT_CAPACITY;
    public int BELT_WEIGHT_LIMIT;
    public int TRUCK_CAPACITY;
    private int takeBrickTime;
    public void setTruck(Truck truck) {
        this.truck = truck;
    }
    private Truck truck;
    public Worker(int brickType, Truck truck, int BELT_CAPACITY, int
BELT_WEIGHT_LIMIT, int TRUCK_CAPACITY) {
        this.brickType = brickType;
        this.truck = truck;
        this.BELT_CAPACITY = BELT_CAPACITY;
        this.BELT_WEIGHT_LIMIT = BELT_WEIGHT_LIMIT;
        this.TRUCK_CAPACITY = TRUCK_CAPACITY;
        this.takeBrickTime = brickType*100;
    }
    @Override
    public void run() {
        while (true) {
            //System.out.println("-------
"+HelloApplication.bricksOnBelt.size());
            try {
```

```
                Thread.sleep(takeBrickTime);
                Main.lock.acquire();
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
            layBrick(brickType);
            Main.lock.release();
        }
    }
    private void layBrick(int mass) {
        synchronized (bricksOnBelt) {
            int weight = 0;
            for(Brick brick : bricksOnBelt.bricksList) {
                weight += brick.mass;
            }
            //System.out.println("Belt weight: "+weight+" Worker P"+mass+"
working..");
            if (bricksOnBelt.getSize() < BELT_CAPACITY &&
                    weight + mass <= BELT_WEIGHT_LIMIT &&
                    truck.container + weight + mass <= TRUCK_CAPACITY) {
                Brick brick = new Brick(brickType);
                bricksOnBelt.add(brick);
                Platform.runLater(new Runnable() {
                    @Override
                    public void run() {
                        Main.belt.getItems().add(String.valueOf(mass));
                    }
                });
                Main.updateLabels(truck);
                //System.out.println("brick layed by P"+ brickType+", belt
weight: "+(weight+mass)+"");
            }else {
                truck.loadTruck();
            }
            try{
                Thread.sleep(200);
            }
            catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```

Main:

```
package com.example.projektpw;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.geometry.Insets;
import javafx.geometry.Orientation;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.ListView;
```

```
import javafx.scene.layout.*;
import javafx.stage.Stage;

import java.io.*;
import java.util.Properties;
import java.util.concurrent.Semaphore;

public class Main extends Application {
    public static Semaphore lock = new Semaphore(1);
    public Truck truck;
    public static Worker P1;
    public static Worker P2;
    public static Worker P3;
    public static int BELT_CAPACITY;
    public static int BELT_WEIGHT_LIMIT;
    public static int TRUCK_CAPACITY;
    public static Label truckLoadLabel = new Label("Truck load: 0");
    public static Label truckCountLabel = new Label("Truck count: 0");
    public static Label bricksOnBeltLabel = new Label("Bricks on belt: 0");
    private Label P1label = new Label("P1");
    private Label P2label = new Label("P2");
    private Label P3label = new Label("P3");
    public static VBox road = new VBox();
    public static ListView<String> belt = new ListView<>();
    public static final ConvoyerBelt bricksOnBelt = new ConvoyerBelt();
    public static void main(String[] args) {
        launch(args);
    }

    public void start(Stage primaryStage) {
        readConfig();
        primaryStage.setTitle("Symulator taśmy transportowej");

        HBox root = new HBox();
        root.setPadding(new Insets(10));
        root.setSpacing(50);
        root.setAlignment(Pos.CENTER);

        HBox truckBoxMain = createTruckBoxMain();
        VBox bricksOnBeltBoxMain = createBricksOnBeltBoxMain();
        VBox workerBox = createWorkerBox();

        root.getChildren().addAll(truckBoxMain, bricksOnBeltBoxMain,
workerBox);

        Scene scene = new Scene(root, 800, 200);
        primaryStage.setScene(scene);
        primaryStage.show();

        Truck truck = new Truck(0);
        P1 = new Worker(1, truck, BELT_CAPACITY, BELT_WEIGHT_LIMIT,
TRUCK_CAPACITY);
        P2 = new Worker(2, truck, BELT_CAPACITY, BELT_WEIGHT_LIMIT,
TRUCK_CAPACITY);
        P3 = new Worker(3, truck, BELT_CAPACITY, BELT_WEIGHT_LIMIT,
TRUCK_CAPACITY);
        Thread p1 = new Thread(P1);
```

```
        Thread p2 = new Thread(P2);
        Thread p3 = new Thread(P3);

        p1.start();
        p2.start();
        p3.start();
    }

    private HBox createTruckBoxMain(){
        HBox box = new HBox();
        road.setStyle("-fx-border-color: black; " +
                "-fx-border-style: solid; " +
                "-fx-border-width: 0 2px 0 2px;");
        road.setMinSize(100, 100);
        road.setAlignment(Pos.CENTER_RIGHT);
        VBox newBox = new VBox();
        HBox truckCountBox = new HBox();
        truckCountBox.setAlignment(Pos.TOP_LEFT);
        truckCountBox.getChildren().addAll(truckCountLabel);
        HBox truckLoadBox = new HBox();
        truckLoadBox.setAlignment(Pos.BOTTOM_LEFT);
        truckLoadBox.getChildren().addAll(truckLoadLabel);
        newBox.getChildren().addAll(truckCountBox,truckLoadBox);
        newBox.setSpacing(10);
        newBox.setAlignment(Pos.CENTER_LEFT);
        box.setSpacing(20);
        box.getChildren().addAll(newBox, road);
        return box;
    }

    private VBox createBricksOnBeltBoxMain(){
        VBox newBox = new VBox();
        HBox beltBox = new HBox();
        beltBox.setStyle("-fx-border-color: brown; " +
                "-fx-border-style: solid; " +
                "-fx-border-width: 2px; " +
                "-fx-border-radius: 5px;");
        beltBox.setAlignment(Pos.BASELINE_CENTER);
        beltBox.setMinSize(400, 100);
        belt.setOrientation(Orientation.HORIZONTAL);
        beltBox.getChildren().add(belt);
        VBox beltLabel = new VBox();
        beltLabel.setSpacing(50);
        beltLabel.setAlignment(Pos.BOTTOM_CENTER);
        beltLabel.setPadding(new Insets(30));
        beltLabel.getChildren().addAll(bricksOnBeltLabel);
        newBox.getChildren().addAll(beltBox, beltLabel);
        return newBox;
    }

    private VBox createWorkerBox(){
        VBox newBox = new VBox();
        newBox.setSpacing(50);
        newBox.setAlignment(Pos.CENTER_RIGHT);
        newBox.getChildren().addAll(P1label, P2label, P3label);
        return newBox;
    }
```

```
    private void readConfig(){
        try (InputStream input = new
FileInputStream("src/main/resources/config.properties")) {
            Properties prop = new Properties();
            prop.load(input);
            BELT_CAPACITY =
Integer.parseInt(prop.getProperty("BELT_CAPACITY"));
            BELT_WEIGHT_LIMIT =
Integer.parseInt(prop.getProperty("BELT_WEIGHT_LIMIT"));
            TRUCK_CAPACITY =
Integer.parseInt(prop.getProperty("TRUCK_CAPACITY"));

        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
    public static void updateLabels(Truck truck) {
        Platform.runLater(new Runnable() {
            @Override
            public void run() {
                Main.truckLoadLabel.setText("Truck load: " +
truck.container);
                Main.bricksOnBeltLabel.setText("Bricks on belt: " +
bricksOnBelt.getSize());
                Main.truckCountLabel.setText("Truck count: " + (truck.id));
            }
        });
    }
}
```