**Programowanie Współbieżne**

**SPRAWOZDANIE**



**Temat:** Załadunek Koparkami

**Prowadzący :** płk dr inż. Jarosław Rulka

**Autor :**

**Data Oddania:**

**Grupa:**

1. **Treść zadania**

**Załadunek Koparkami**

**Założenia:**

Napisać program synchronizację podstawiania samochodów pod załadunek koparką. Należy założyć istnienie N koparek i K samochodów o małej ładowności i L samochodów o dużej ładowalności. Jedna koparka obsługuje równocześnie dwa małe pojazdy lub jeden duży. Dodatkowo należy przyjąć, że operatorzy koparek mają przerwę obiadową między godziną 1300 i 1400, a niecierpliwi kierowcy rezygnują z załadunku jeśli nie podjadą na stanowisko w ciągu założonego maksymalnego czasu oczekiwania np. 0.5 godz.

1. **Syntetyczny opis problem – przyjęte założenia**

Implementacji powyższego zadania dokonuję w języku Java.

Aby dokonać pełnej kompilacji kodu należy mieć skonfigurowaną bibliotekę Javafx w środowisku InteliJ.

|  |  |
| --- | --- |
| **Nazwa Klasy** | **Reprezentacja założenia** |
| **Application** | Powołuje wątki samochodów i ładowarek, tworzy wszystkie obiekty i łączy je z graficznym interfejsem |
| **Car** | Klasa samochodu jest klasą wątkową, która symuluje działanie samochodu. Dołącza do kolejki w losowym przedziale czasu i czeka na załadunek. Kończy działanie po załadowaniu lub po upłynięciu czasu rezygnacji. |
| **Config** | Klasa przechowująca na początku wszystkie ustawienia domyślne pobrane z pliku. Następnie nadpisuje je przez wprowadzone przez użytkownika. |
| **Loader** | Klasa ładowarki jest klasą wątku, która symuluje działanie ładowarki załadowując auta na statek. Praca wątku mocno uzależniona jest od czasomierza. Ładowarka zaczyna i konczy pracę poprzez sygnał od zegara, tak samo odchodzi i wraca z przerwy. Załadunek odbywa się w losowym przedziale czasu. |
| **Queue** | Klasa sterująca kolejką samochodów. Przedstawia ją za pomoca listy. |
| **Stopper** | Klasa pełniąca funkcję czasomierza. Każdy samochód w momencie dołączenia do kolejki uruchamia czasomierz. Po upłynięciu czasu rezygnacji Stopper informuje samochód. Kończy działanie w momencie gdy samochód zrezygnuje z załadunku lub zostanie załadowany |
| **WorkClock** | Klasa wątku symulujaca zegar. Odmierza dobę w przyśpieszony sposób. Steruje pracą ładowarek. |
| **Main** | Klasa rozpoczynająca wątek interfejsu graficzngo javaFX |
| **SceneController** | Klasa sterująca interfejsem graficznym i powołująca wątek aplikacji. |

Symulacja działa w następujący sposób:

* + - Użytkownik na wejściu podaje:
      * Liczbę samochodów o małej i dużej objętości.
      * Przedział czasowy możliwego czasu wykonywania załadunku
      * Przedział czasowy możliwego dołączania pojazdów
      * Godzinę początku i końca pracy
      * Godzinę początku i końca przerwy
      * Minuty po których następuje rezygnacja auta.
    - Samochody w maksymalnym przedziale czasowym dołaczają i czekają na załadunek.
    - Jednocześnie jedna ładowarka może ładować 2 auta o małej objętości lub jedno auto o dużej objętości. Kolejność ładowania odbywa się za pomocą pierwszeństwa w kolejce. Przyjęte założenia problematycznych sytuacji:
      * Gdy pierwsze w kolejce jest auto o małej objętości, jednak w kolejce nie ma już żadnych aut o małej objętości. Auto jest ładowane pojedyńczo.
      * Gdy pierwsze w kolejce jest auto o małej objętości, jednak drugie w kolejce jest auto o dużej objętości. Kolejka przeszukiwana jest pod względem małego auta i do załadunku wybierane jest następne auto o małej pojemności.
    - Ładowarka rozpoczyna i kończy pracę według zegara. Podczas przerwy ładowarka nie pracuje.
    - Czas załadunku jest losowy i trwa mieści się w przedziale wprowadzonym przez użytkownika.
    - Po załadunku auta wędrują do listy załadunkowe, odpowiadającej statkowi.
    - Gdy wątek auta dołącza do kolejki włączany jest stopper mirzący ile czasu dane auto spędziło w kolejce. Jeśli stopper odliczy czas rezygnacji auto usuwane jest z kolejki i dodawane do listy aut, które zrezygnowały a następnie wątek auta i stoppera kończy pracę.

1. **Wykaz współdzielonych zasobów**

|  |  |
| --- | --- |
| **Współdzielony Zaasób** | **Opis** |
| **Queue** | Do kolejki dołączają auta, oraz są z niej usuwane w różnych momentach. Ładowarki pobierają auta o różnych pojemnościach z kolejki. |

1. **Wykaz wyróżnionych punktów synchronizacji**

- Pobranie auta z kolejki

- Usunięcie auta z kolejki

1. **Wykaz obiektów synchronizacji**

- Lock access ( Zamek dostęp do moniora)

- Condition empty ( Warunek kolejkaPusta )

- Condition breakTime ( Warunek czasPrzerwy )

- Condition workTime ( Warunek czasPracy )

1. **Wykaz procesów sekwencyjnych**

- Car ( Samochód )

- Stopper ( Czasomierz )

- Loader ( Ładowarka )

- WorkClock ( Zegar Pracy )

Synchronizacja Danych procesów polega na zastosowania mechanizmu monitora i zamków.

Proces Kolejki dopuszcza tylko jeden proces sekwencyjny do manipulowania kolejki.

Gdy kolejka jest pusta i Ładowarka chce pobrać zasób z kolejki wchodzi do monitora i zatrzymuje się na warunku KolejkaPusta. Gdy nowe auto zostanie dodane do kolejki ładowarka zostanie odblokowana.

Gdy następuje czas przerwy i ładowarka chce pobrać kolejne auto zatrzymuje się na warunku CzasPrzerwy. Zostanie odblokowane gdy zegar wymierzy czas pracy.

Gdy skończy się czas pracy i ładowarka chce auto z kolejki zatrzymuje się na warunku CzasPracy. Zostanie odblokowane gdy zegar wymierzy czas pracy.

Gdy auto zrezygnuje i chce być usunięte z kolejki musi otrzymac dostęp do monitora.

1. **Listing Programu**

**Application. java**

package app;  
import gui.SceneController;  
import java.util.LinkedList;  
  
public class Application extends Thread {  
  
 SceneController sceneController;  
 public Config config;  
 public Queue queue;  
 public LinkedList<Car> loadedCars;  
 public WorkClock workClock;  
 Car[] lowCapacityCars;  
 Car[] highCapacityCars;  
 public Loader[] loaders;

public Application(SceneController sceneController, Config config){  
  
 this.sceneController =sceneController;  
 this.config = config;  
 this.queue = new Queue("Queue",sceneController);  
 this.loadedCars = new LinkedList<Car>();  
 this.workClock = new WorkClock(9, 0, 0, config, queue,sceneController);  
 this.lowCapacityCars = new Car[config.lowCapacityCars];  
 this.highCapacityCars= new Car[config.highCapacityCars];  
 this.loaders = new Loader[config.numberOfLoaders];  
 }  
  
 public void run() {  
  
 workClock.start();  
 for (int i = 0; i < config.lowCapacityCars; i++) {  
 lowCapacityCars[i] = new Car(i, "Low", config, queue, workClock);}  
 for (int i = 0; i < config.highCapacityCars; i++) {  
 highCapacityCars[i] = new Car(i, "High", config, queue, workClock); }  
 for (int i = 0; i < config.numberOfLoaders; i++) {  
 loaders[i] = new Loader(i, queue, workClock, config, loadedCars);}  
 for (int i = 0; i < config.highCapacityCars; i++) {  
 highCapacityCars[i].start();}  
 for (int i = 0; i < config.lowCapacityCars; i++) {  
 lowCapacityCars[i].start();}  
 for (int i = 0; i < loaders.length; i++) { loaders[i].start(); }  
 for (int i = 0; i < config.lowCapacityCars; i++) {  
 try { lowCapacityCars[i].join();  
 } catch (InterruptedException e) { e.printStackTrace();}  
 }  
 for (int i = 0; i < config.highCapacityCars; i++) {  
 try {  
 highCapacityCars[i].join();  
 } catch (InterruptedException e) { e.printStackTrace();}  
 }  
 for (int i = 0; i < loaders.length; i++) {  
 try {  
 loaders[i].join();  
 } catch (InterruptedException e) {e.printStackTrace();}  
 }  
 try {  
 workClock.join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
}

**Car. java**

package app;  
import javafx.animation.FadeTransition;  
import javafx.animation.TranslateTransition;  
import javafx.scene.paint.Color;  
import javafx.scene.shape.Rectangle;  
import javafx.util.Duration;  
import org.w3c.dom.css.Rect;  
import java.util.Random;  
import java.util.concurrent.Semaphore;  
  
public class Car extends Thread{  
  
 int id;  
 public String capacity;  
 Config config;  
 WorkClock workClock;  
 volatile boolean loaded;  
 Stopper stopper;  
 public Color color;  
 public int x = 0;  
 public int y = 0;  
 public Rectangle rectangle;  
 public TranslateTransition transition;  
 Semaphore carSem;  
 Queue queue;  
 Random rand = new Random();  
  
 Car(int id ,String capacity, Config config, Queue queue, WorkClock workClock){  
 this.id = id;  
 this.capacity = capacity;  
 this.config = config;  
 this.queue = queue;  
 this.workClock=workClock;  
 this.transition =new TranslateTransition();  
 this.ft = new FadeTransition(Duration.*millis*(5000), rectangle);  
 int r = rand.nextInt(255);  
 int g = rand.nextInt(255);  
 int b = rand.nextInt(255);  
 this.color =Color.*rgb*(r,g,b);  
 this.loaded = false;  
 this.carSem = new Semaphore(0);  
 }  
  
 private void delay(int delayMin, int delayMax){  
 if(delayMin<=delayMax && delayMin>0 ) {  
 try {  
 Thread.*sleep*(rand.nextInt(delayMax - delayMin) + delayMin);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 else{  
 try {  
 Thread.*sleep*(rand.nextInt(8000) + 2000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 public void joinQueue(){  
 delay(config.joiningQueueMinDelay,config.joiningQueueMaxDelay);  
 if(workClock.workTime==true && workClock.on==true) {  
 queue.addCar(this);  
 stopper = new Stopper(config.minutesAfterDriversResign, carSem, config);  
 stopper.start();  
 queue.displayQueue();  
 System.*out*.println("I am a car " + id + " I joined the queue");  
 }  
 }  
  
 public void run(){  
 joinQueue();  
 try {  
 carSem.acquire();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 if(stopper.resigned==true){  
 System.*out*.println("I am a car " + id + " I resigned");  
 queue.carResigned(this);  
 }  
 else if(loaded){  
 System.*out*.println("I am a car " + id + " I've been removed from queue");  
 }  
 try {  
 stopper.join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
}

public void joinQueue(){  
 delay(config.joiningQueueMinDelay,config.joiningQueueMaxDelay);  
 if(workClock.workTime==true && workClock.on==true) {  
 queue.addCar(this);  
 stopper = new Stopper(config.minutesAfterDriversResign, carSem, config);  
 stopper.start();  
 queue.displayQueue();  
 System.*out*.println("I am a car " + id + " I joined the queue");  
 }  
 }  
  
 public void run(){  
 joinQueue();  
 try {  
 carSem.acquire();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 if(stopper.resigned==true){  
 System.*out*.println("I am a car " + id + " I resigned");  
 queue.carResigned(this);  
 }  
 else if(loaded){  
 System.*out*.println("I am a car " + id + " I've been removed from queue");  
 }  
 try {  
 stopper.join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
}

**Config.java**

package app;  
import java.io.FileInputStream;  
import java.io.InputStream;  
import java.util.Properties;  
  
public class Config {  
  
 static int *hoursLimit* = 24;  
 static int *minutesLimit* = 60;  
 static int *secondsLimit* = 60 ;  
 public int lowCapacityCars ;  
 public int highCapacityCars ;  
 public int numberOfLoaders ;  
 public int startWorkHour ;  
 public int endWorkHour ;  
 public int startBreakHour;  
 public int endBreakHour ;  
 public int minutesAfterDriversResign;  
 public int pasteOfTime;  
 public int joiningQueueMinDelay;  
 public int joiningQueueMaxDelay;  
 public int loadingCarMinDelay;  
 public int loadingCarMaxDelay;  
  
 public Config(String inputStream){  
 try(InputStream input = new FileInputStream(inputStream)){  
 Properties prop = new Properties();  
 prop.load(input);  
 this.lowCapacityCars = Integer.*parseInt*(prop.getProperty("lowCapacityCars"));  
 this.highCapacityCars=Integer.*parseInt*(prop.getProperty("highCapacityCars"));  
 this.numberOfLoaders=Integer.*parseInt*(prop.getProperty("numberOfLoaders"));  
 this.startWorkHour=Integer.*parseInt*(prop.getProperty("startWorkHour"));  
 this.endWorkHour=Integer.*parseInt*(prop.getProperty("endWorkHour"));  
 this.startBreakHour=Integer.*parseInt*(prop.getProperty("startBreakHour"));  
 this.endBreakHour=Integer.*parseInt*(prop.getProperty("endBreakHour"));  
 this.minutesAfterDriversResign=Integer.*parseInt*(prop.getProperty("minutesAfterDriversResign"));  
 this.pasteOfTime=Integer.*parseInt*(prop.getProperty("pasteOfTime"));  
 this.joiningQueueMinDelay=Integer.*parseInt*(prop.getProperty("joiningQueueMinDelay"));  
 this.joiningQueueMaxDelay=Integer.*parseInt*(prop.getProperty("joiningQueueMaxDelay"));  
 this.loadingCarMinDelay=Integer.*parseInt*(prop.getProperty("loadingCarMinDelay"));  
 this.loadingCarMaxDelay=Integer.*parseInt*(prop.getProperty("loadingCarMaxDelay"));  
  
 }catch (Exception e){  
 e.printStackTrace();  
 }  
 }  
}

**Loader.java**

package app;  
import java.util.LinkedList;  
import java.util.Random;  
  
public class Loader extends Thread{  
 public int id;  
 Config config;  
 Queue queue;  
 WorkClock workClock;  
 Random rand = new Random();  
 public LinkedList<Car> retCars;  
 public LinkedList<Car> loadedCars;  
  
 Loader(int id, Queue queue, WorkClock workClock, Config config, LinkedList<Car> loadedCars ){  
 this.config = config;  
 this.queue = queue;  
 this.workClock = workClock;  
 this.id = id;  
 this.retCars= new LinkedList<Car>();  
 this.loadedCars = loadedCars;  
 }  
 private void delay(int delayMin, int delayMax){  
 if(delayMin<=delayMax && delayMin>0 && delayMax >0 ) {  
 try {  
 Thread.*sleep*(rand.nextInt(delayMax - delayMin) + delayMin);  
 } catch (InterruptedException e) {e.printStackTrace();}  
 }  
 else{  
 try {  
 Thread.*sleep*(rand.nextInt(8000) + 2000);  
 } catch (InterruptedException e) { e.printStackTrace();}  
 }  
 }  
 public void loading(LinkedList<Car> retCars){  
 System.*out*.println("app.Loader : "+id+" - I am loading...");  
 delay(config.loadingCarMinDelay,config.loadingCarMaxDelay);  
 System.*out*.println("app.Loader : "+id+" - I finished loading ");  
 for(int i=0; i<retCars.size();i++){  
 loadedCars.add(retCars.get(i));  
 }  
 retCars.clear();  
 }  
 public void run(){  
 working();  
 }  
  
 public void working(){  
 while (workClock.on) {  
 //System.out.println("app.Loader : "+id+" - I am free");  
 retCars = queue.getLoad(this);  
 loading(retCars);  
 }  
 }  
}

**Queue. java**

package app;  
import gui.SceneController;  
import javafx.application.Platform;  
import java.util.LinkedList;  
import java.util.concurrent.locks.Condition;  
import java.util.concurrent.locks.Lock;  
import java.util.concurrent.locks.ReentrantLock;  
  
public class Queue{  
  
 String name;  
 public LinkedList<Car> cars = new LinkedList<Car>();  
 public LinkedList<Car> resignedCars= new LinkedList<Car>();  
 Lock access;  
 Condition empty;  
 Condition breakTime;  
 Condition workTime;  
 SceneController sceneController;  
  
 Queue(String name, SceneController sceneController){  
 this.name = name;  
 this.access = new ReentrantLock();  
 this.empty = access.newCondition();  
 this.breakTime = access.newCondition();  
 this.workTime = access.newCondition();  
 this.sceneController = sceneController;  
 }  
  
 public void displayQueue(){  
 for (int i = 0; i < cars.size(); i++) {  
 System.*out*.print(cars.get(i).id + " ");}  
 System.*out*.println();  
 }  
  
 public int searchForNextLowCapacityCar(){  
 for (int i = 1; i < cars.size(); i++) {  
 if(cars.get(i).capacity=="Low"){  
 return i;  
 }  
 }  
 return -1;  
 }  
  
 public LinkedList<Car> loadCar(LinkedList<Car> retCars, Car car){  
 retCars.add(car);  
 car.loaded = true;  
 car.carSem.release();  
 removeCar(car);  
 return retCars;  
 }  
  
 public LinkedList<Car> getLoad(Loader loader){  
  
 LinkedList<Car> retCars= new LinkedList<Car>();  
 try{  
 access.lock();  
 //Lock stops loader if there is break Time  
 if(loader.workClock.breakTime==true){  
 try {  
 System.*out*.println("app.Loader : "+loader.id+" - Start of the break time");  
 breakTime.await();  
 System.*out*.println("app.Loader : "+loader.id+" - End of daily breaK time");  
  
 } catch (Exception e) { System.*out*.println(e.getMessage());}  
 }  
  
 //Lock stops loader if there is end of daily work time  
 //if(loader.workClock.workTime==false){  
 // try {  
 // System.out.println("app.Loader : "+loader.id+"- End of daily work time");  
 // workTime.await();  
 // System.out.println("app.Loader : "+loader.id+"- Start of daily work time");  
 // } catch (Exception e) { System.out.println(e.getMessage());}  
 //}  
 //System.out.println("app.Loader : "+loader.id+" - I am in the monitor");  
  
  
 //lock stops loader if the queue is empty  
 if (cars.size() == 0) {  
 try {  
 empty.await();  
 System.*out*.println("app.Loader : "+loader.id+" - The queue is empty");  
  
 } catch (Exception e) { System.*out*.println(e.getMessage());}  
 }  
  
 //load the first car  
 String capacity = cars.getFirst().capacity;  
 loadCar(retCars,cars.getFirst());  
  
 if(capacity.equals("Low")) {  
 int lcIndex= searchForNextLowCapacityCar();  
 if (lcIndex != -1) {  
 loadCar(retCars,cars.get(lcIndex));  
 }  
 }  
  
 System.*out*.print("app.Loader : "+loader.id+" - I took a load of: ");  
 for(int i=0;i<retCars.size();i++){  
 System.*out*.print(" car id: "+retCars.get(i).id+ " Capacity: "+ retCars.get(i).capacity);  
 }  
 System.*out*.println();  
 }finally {  
 System.*out*.println("app.Loader : "+loader.id+" - I stopped accessing queue");  
 access.unlock();  
 return retCars;  
 }  
 }  
  
 public void carResigned(Car car){  
 try {  
 access.lock();  
 cars.remove(car);  
 Platform.*runLater*(() -> {  
 sceneController.carResigning(car);  
 });  
 resignedCars.add(car);  
 empty.signal();  
 }finally {  
 access.unlock();  
 }  
 }  
  
 public void addCar(Car car) {  
 try {  
 access.lock();  
 cars.add(car);  
 empty.signal();  
 }finally {  
 access.unlock();  
 }  
 }  
  
 public void removeCar(Car car) {  
 try {  
 access.lock();  
 cars.remove(car);  
 empty.signal();  
 }finally {  
 access.unlock();  
 }  
 }  
}

} catch (Exception e) { System.*out*.println(e.getMessage());}  
 }  
 if (cars.size() == 0) {  
 try {  
 empty.await();  
 System.*out*.println("app.Loader : "+loader.id+" - The queue is empty");  
  
 } catch (Exception e) { System.*out*.println(e.getMessage());}  
 }  
  
 String capacity = cars.getFirst().capacity;  
 loadCar(retCars,cars.getFirst());  
  
 if(capacity.equals("Low")) {  
 int lcIndex= searchForNextLowCapacityCar();  
 if (lcIndex != -1) {  
 loadCar(retCars,cars.get(lcIndex));  
 }  
 }  
  
 System.*out*.print("app.Loader : "+loader.id+" - I took a load of: ");  
 for(int i=0;i<retCars.size();i++){  
 System.*out*.print(" car id: "+retCars.get(i).id+ " Capacity: "+ retCars.get(i).capacity);  
 }  
 System.*out*.println();  
 }finally {  
 System.*out*.println("app.Loader : "+loader.id+" - I stopped accessing queue");  
 access.unlock();  
 return retCars;  
 }  
 }  
 public void carResigned(Car car){  
 try {  
 access.lock();  
 cars.remove(car);  
 Platform.*runLater*(() -> {  
 sceneController.carResigning(car);  
 });  
 resignedCars.add(car);  
 empty.signal();  
 }finally {  
 access.unlock();  
 }  
 }  
 public void addCar(Car car) {  
 try {  
 access.lock();  
 cars.add(car);  
 empty.signal();  
 }finally {  
 access.unlock();  
 }  
 }  
  
 public void removeCar(Car car) {  
 try {  
 access.lock();  
 cars.remove(car);  
 empty.signal();  
 }finally {  
 access.unlock();  
 }  
 }  
}

**Stopper.java**

package app;  
import java.util.concurrent.Semaphore;  
  
public class Stopper extends Thread{  
  
 int minutes;  
 int minutesToWait;  
 int seconds;  
  
 boolean alarm;  
 Config config;  
  
 Semaphore carSem;  
 boolean resigned;  
  
 Stopper(int minutesToWait, Semaphore carSem, Config config){  
 this.minutes = 0;  
 this.config = config;  
 this.alarm = false;  
 this.minutesToWait = minutesToWait;  
 this.carSem = carSem;  
 this.resigned = false;  
 }  
  
 public void run(){  
 while(!alarm) {  
 try {  
 Thread.*sleep*(0,config.pasteOfTime);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 if(minutes == minutesToWait){  
 alarm=true;  
 this.resigned = true;  
 carSem.release();  
 }  
 if(seconds == config.*secondsLimit*){  
 minutes++;  
 seconds = 0;  
 }  
 seconds ++;  
 }  
 }  
}

**WorkClock.Java**

package app;  
import gui.SceneController;  
public class WorkClock extends Thread {  
  
 static  
 int *hour*;  
 int minutes;  
 int seconds;  
 Config config;  
 public boolean breakTime;  
 public boolean workTime;  
 public boolean on;  
 Queue queue;  
 SceneController sceneController;  
  
 WorkClock(int startHour, int startMinute, int startSecond, Config config, Queue queue, SceneController sceneController){  
 this.*hour* = startHour;  
 this.minutes = startMinute;  
 this.seconds = startSecond;  
 this.breakTime = false;  
 this.workTime = false;  
 this.config = config;  
 this.queue = queue;  
 this.sceneController = sceneController;  
 this.on = true;  
 }  
 public void run (){  
 while(on) {  
 try {  
 Thread.*sleep*(0,config.pasteOfTime);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 if(*hour* == config.startBreakHour){  
 breakTime= true;  
 }  
 if(*hour* == config.endBreakHour){  
 breakTime=false;  
 try {  
 queue.access.lock();  
 queue.breakTime.signal();  
 }finally {  
 queue.access.unlock();  
 }  
 }  
 if(*hour* == config.startWorkHour){  
  
 workTime = true;  
 try {  
 queue.access.lock();  
 queue.workTime.signal();  
 }finally {  
 queue.access.unlock();  
 }  
 }  
 if(*hour* == config.endWorkHour){  
 workTime = false;  
 }  
 if(*hour* == config.*hoursLimit*){  
 *hour* = 0;  
  
 }  
 if(minutes == config.*minutesLimit*){  
 *hour*++;  
 minutes=0;  
  
 }  
 if(seconds == config.*secondsLimit*){  
 minutes++;  
 sceneController.setTime(*hour*,minutes);  
 seconds = 0;  
 }  
 seconds ++;  
 }  
 }  
}

if(*hour* == config.endWorkHour){  
 workTime = false;  
 }  
 if(*hour* == config.*hoursLimit*){  
 *hour* = 0;  
  
 }  
 if(minutes == config.*minutesLimit*){  
 *hour*++;  
 minutes=0;  
  
 }  
 if(seconds == config.*secondsLimit*){  
 minutes++;  
 sceneController.setTime(*hour*,minutes);  
 seconds = 0;  
 }  
 seconds ++;  
 }  
 }  
}

**Main.java**

package gui;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.scene.shape.Circle;  
import javafx.stage.Stage;  
  
import java.io.IOException;  
  
public class Main extends Application {  
 @Override  
 public void start(Stage stage) throws IOException {  
 FXMLLoader fxmlLoader = new FXMLLoader(Main.class.getResource("Scene.fxml"));  
 Scene scene = new Scene(fxmlLoader.load(), 1160, 720);  
 stage.setTitle("Loading Simulation");  
 stage.setScene(scene);  
 stage.setResizable(false);  
  
 Circle circle = new Circle();  
 circle.setCenterX(100);  
 circle.setCenterY(100);  
 circle.setVisible(true);  
 stage.show();  
 }  
  
 public static void main(String[] args) {  
 *launch*();  
 }  
}

**SceneController.java**

package gui;  
import app.\*;  
import javafx.animation.FadeTransition;  
import javafx.animation.TranslateTransition;  
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.TextField;  
import javafx.scene.layout.Pane;  
import javafx.scene.paint.Color;  
import javafx.scene.shape.Rectangle;  
import javafx.scene.transform.Translate;  
import javafx.util.Duration;  
  
import java.net.URL;  
import java.util.ResourceBundle;  
public class SceneController implements Initializable {  
  
 @FXML private Pane simulationPane;  
 @FXML public Pane resignedCarsPane;  
 @FXML private Label time;  
 @FXML private TextField lowCapacityCarsTextField;  
 @FXML private TextField highCapacityCarsTextField;  
 @FXML private TextField startWorkHourTextField;  
 @FXML private TextField endWorkHourTextField;  
 @FXML private TextField startBreakHourTextField;  
 @FXML private TextField endBreakHourTextField;  
 @FXML private TextField joiningQueueMaxDelayTextField;  
 @FXML private TextField joiningQueueMinDelayTextField;  
 @FXML private TextField loadingMaxDelayTextField;  
 @FXML private TextField loadingMinDelayTextField;  
 @FXML private TextField minutesToResignTextField;  
 @FXML private Label loader1Status;  
 @FXML private Label loader2Status;  
 @FXML private Label loader3Status;  
 @FXML private Label loader4Status;  
  
 Queue queue;  
 Application application;  
 Config config;  
  
 int [] loadersX1cords=new int[4];  
 int [] loadersX2cords=new int[4];  
 int [] loadersYCords =new int[4];  
  
 public void carResigning(Car car){  
  
 Platform.*runLater*(() -> {  
 Rectangle rect = new Rectangle();  
 rect.setHeight(15);  
 rect.setWidth(53);  
 rect.setX(948);  
 rect.setY(300);  
 rect.setFill(Color.*rgb*(59,59,59));  
 simulationPane.getChildren().add(rect);  
 });  
  
 Rectangle rect = new Rectangle();  
 simulationPane.getChildren().add(rect);  
 rect.setHeight(15);  
 if(car.capacity.equals("Low")) {  
 rect.setWidth(22);  
 }else {rect.setWidth(53);}  
 rect.setX(car.x);  
 rect.setY(300);  
 rect.setFill(car.color);  
  
 car.transition.setNode(rect);  
 car.transition.setDuration(Duration.*millis*(4000));  
 car.transition.setByX(948-car.x);  
 car.transition.play();  
  
 }  
 @Override  
 public void initialize(URL url, ResourceBundle resourceBundle) {  
  
 this.config = new Config("E:/GotoweProjekty/Loading\_GUI/src/resources/config.properties");  
 initializeTextFields();  
 initializeLoaderLabels();  
 initializeCords();  
  
 }  
 private void initializeLoaderLabels(){  
 loader1Status.setText("LOADER 1");  
 loader2Status.setText("LOADER 2");  
 loader3Status.setText("LOADER 3");  
 loader4Status.setText("LOADER 4");  
 }  
 private void setLoaderLabels(String text){  
 loader1Status.setText(text);  
 loader2Status.setText(text);  
 loader3Status.setText(text);  
 loader4Status.setText(text);  
 }  
 private void initializeTextFields(){  
 this.lowCapacityCarsTextField.setText(Integer.*toString*( config.lowCapacityCars));  
 this.highCapacityCarsTextField.setText(Integer.*toString*( config.highCapacityCars));  
 this.startWorkHourTextField.setText(Integer.*toString*(config.startWorkHour));  
 this.endWorkHourTextField.setText(Integer.*toString*( config.endWorkHour));  
 this.startBreakHourTextField.setText(Integer.*toString*( config.startBreakHour));  
 this.endBreakHourTextField.setText(Integer.*toString*( config.endBreakHour));  
 this.joiningQueueMaxDelayTextField.setText(Integer.*toString*( config.joiningQueueMaxDelay));  
 this.joiningQueueMinDelayTextField.setText(Integer.*toString*( config.joiningQueueMinDelay));  
 this.loadingMaxDelayTextField.setText(Integer.*toString*( config.loadingCarMaxDelay));  
 this.loadingMinDelayTextField.setText(Integer.*toString*( config.loadingCarMinDelay));  
 this.minutesToResignTextField.setText(Integer.*toString*( config.minutesAfterDriversResign));  
 }  
 public void getValuesFromTextFields(){  
 this.config.lowCapacityCars =Integer.*parseInt*( lowCapacityCarsTextField.getText());  
 this.config.highCapacityCars =Integer.*parseInt*( highCapacityCarsTextField.getText());  
 this.config.startWorkHour =Integer.*parseInt*( startWorkHourTextField.getText());  
 this.config.endWorkHour =Integer.*parseInt*( endWorkHourTextField.getText());  
 this.config.startBreakHour =Integer.*parseInt*( startBreakHourTextField.getText());  
 this.config.endBreakHour =Integer.*parseInt*( endBreakHourTextField.getText());  
 this.config.joiningQueueMaxDelay =Integer.*parseInt*( joiningQueueMaxDelayTextField.getText());  
 this.config.joiningQueueMinDelay=Integer.*parseInt*( joiningQueueMinDelayTextField.getText());  
 this.config.loadingCarMaxDelay =Integer.*parseInt*( loadingMaxDelayTextField.getText());  
 this.config.loadingCarMinDelay =Integer.*parseInt*( loadingMinDelayTextField.getText());  
 this.config.minutesAfterDriversResign =Integer.*parseInt*( minutesToResignTextField.getText());  
 }  
 private void initializeCords(){  
 loadersX1cords[0]=36;  
 loadersX1cords[1]=290;  
 loadersX1cords[2]=540;  
 loadersX1cords[3]=770;  
  
 loadersX2cords[0]=67;  
 loadersX2cords[1]=321;  
 loadersX2cords[2]=570;  
 loadersX2cords[3]=801;  
  
 loadersYCords[0]=463;  
 loadersYCords[1]=458;  
 loadersYCords[2]=458;  
 loadersYCords[3]=454;  
 }  
 private Rectangle spawnCar(int x, int y, String capacity, Color color, Pane pane){  
 Rectangle rect = new Rectangle();  
 rect.setHeight(15);  
 if(capacity.equals("Low")) {  
 rect.setWidth(22);  
 }else {rect.setWidth(53);}  
 rect.setX(x);  
 rect.setY(y);  
 rect.setFill(color);  
 if(x<900) {  
 pane.getChildren().add(rect);  
 }  
 return rect;  
 }  
 public void stopSimulation(ActionEvent actionEvent) {  
 application.workClock.on = false;  
 for(int i=0;i<4;i++) {  
 application.loaders[i].retCars.clear();  
 }  
 application.loadedCars.clear();  
 application.queue.cars.clear();  
 application.queue.resignedCars.clear();  
  
 clearCargo();  
 clearQueue();  
 clearResignedCars();  
 updateLoaders();  
  
 setLoaderLabels("Stopped");  
 }  
 public void clearCargo(){  
 Platform.*runLater*(() -> {  
 Rectangle rect = new Rectangle();  
 rect.setHeight(100);  
 rect.setWidth(870);  
 rect.setX(0);  
 rect.setY(625);  
 rect.setFill(Color.*rgb*(149,117,80));  
 simulationPane.getChildren().add(rect);  
 });  
 }  
 public void startSimulation(ActionEvent actionEvent){  
  
 getValuesFromTextFields();  
 this.application=new Application(this,config);  
 application.start();  
 this.queue = application.queue;  
  
 }  
 private void updateLoaders(){  
  
 if(application!=null && application.workClock.workTime==true) {  
  
 for (int i = 0; i < application.loaders.length; i++) {  
  
 clearLoader(loadersX1cords[i], loadersYCords[i]);  
 if (application.loaders[i].retCars.size()!=0) {  
 spawnCar(loadersX1cords[i], loadersYCords[i], application.loaders[i].retCars.get(0).capacity, application.loaders[i].retCars.get(0).color, simulationPane);  
 if (application.loaders[i].retCars.size() == 2) {  
 spawnCar(loadersX2cords[i], loadersYCords[i], application.loaders[i].retCars.get(1).capacity, application.loaders[i].retCars.get(1).color,simulationPane);  
 }  
 }  
 }  
 }  
 }  
 private void updateCargo(){  
  
 String tempCapacity;  
 int cargoX = 10;  
 int cargoY = 630;  
  
 if(application.loadedCars.size()!=0){  
  
 for(int i=0; i<application.loadedCars.size();i++){  
 if(cargoX>830){  
 cargoX=10;  
 cargoY+=20;  
 }  
 tempCapacity = application.loadedCars.get(i).capacity;  
 spawnCar(cargoX,cargoY,tempCapacity,application.loadedCars.get(i).color,simulationPane);  
 if(tempCapacity=="High"){  
 cargoX +=53;  
 }  
 else{cargoX+=25;}  
 }  
 }  
 }  
 private void clearLoader(int x, int y){  
 spawnCar(x,y,"High",Color.*rgb*(104,88,108),simulationPane);  
 }  
 public void updateQueue(){  
  
 clearQueue();  
 int x = 50;  
 final int y = 260;  
  
 for(int i=0; i<queue.cars.size();i++){  
 final int fx = x;  
 final Car car = queue.cars.get(i);  
 car.x = x;  
 car.y = y;  
 Platform.*runLater*(() -> {  
 car.rectangle= spawnCar(fx, y, car.capacity, car.color, simulationPane);  
 });  
  
 if(car.capacity.equals("Low")){  
 x+=25;  
 }  
 else{  
 x+=50;  
 }  
  
 }  
 }  
 private void spawnResignedCar(int x, int y, String capacity, Color color, Pane pane){  
  
 Rectangle rect = new Rectangle();  
 rect.setWidth(15);  
 if(capacity.equals("Low")) {  
 rect.setHeight(22);  
 }else {rect.setHeight(44);}  
 rect.setX(x);  
 rect.setY(y);  
 rect.setFill(color);  
 pane.getChildren().add(rect);  
 }  
 public void updateResignedCars(){  
 int x = 5;  
 int y = 40;  
  
 for(int i=0; i<queue.resignedCars.size();i++){  
 final int fx = x;  
 final int fy = y;  
 final Car car = queue.resignedCars.get(i);  
  
 Platform.*runLater*(() -> {  
 spawnResignedCar(fx,fy,car.capacity,car.color,resignedCarsPane);  
 });  
 if(x<100) {  
 x += 18;  
 }  
 else {  
 x=5;  
 y+=50;  
 }  
 }  
 }  
 private void clearResignedCars(){  
 Platform.*runLater*(() -> {  
 Rectangle rect = new Rectangle();  
 rect.setHeight(800);  
 rect.setWidth(130);  
 rect.setX(0);  
 rect.setY(30);  
 rect.setFill(Color.*GRAY*);  
 resignedCarsPane.getChildren().add(rect);  
 });  
 }  
 private void clearQueue(){  
 Platform.*runLater*(() -> {  
 Rectangle rect = new Rectangle();  
 rect.setHeight(15);  
 rect.setWidth(950);  
 rect.setX(50);  
 rect.setY(260);  
 rect.setFill(Color.*rgb*(59,59,59));  
 simulationPane.getChildren().add(rect);  
 });  
 }  
 public void setTime(int hour, int minute) {  
  
 String textHour;  
 String textMinute;  
 if (hour < 10 || hour == 0) {  
 textHour = '0' + Integer.*toString*(hour);  
 } else {  
 textHour = Integer.*toString*(hour);  
 }  
 if (minute < 10 || minute == 0) {  
 textMinute = '0' + Integer.*toString*(minute);  
 } else {  
 textMinute = Integer.*toString*(minute);  
 }  
 Platform.*runLater*(() -> {  
 time.setText(textHour + ":" + textMinute);  
 updateQueue();  
 updateLoaders();  
 updateCargo();  
 updateLabels();  
 updateResignedCars();  
 });  
 }  
 private void updateLabels(){  
 if(application.workClock.workTime==true && application.workClock.breakTime==false){  
 setLoaderLabels("Working...");  
 }  
 if(application.workClock.breakTime==true && application.workClock.workTime==true){  
 setLoaderLabels("Break time");  
 }  
 if(application.workClock.breakTime==false && application.workClock.workTime==false){  
 initializeLoaderLabels();  
 }  
 }  
  
  
}

package application;  
  
import javafx.application.Platform;  
import javafx.concurrent.Task;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.control.\*;  
import javafx.scene.paint.Color;  
import javafx.scene.shape.Circle;  
import java.io.FileInputStream;  
import java.io.IOException;  
import java.io.InputStream;  
import java.util.Properties;  
  
public class SceneController{  
  
 @FXML public TextField shelfSize;  
 @FXML public TextField ordersNum;  
  
 @FXML public Label orderNumber;  
 @FXML public Label ordersOnShelf;  
 @FXML public Label ordersFixed;  
  
 @FXML public Circle circle1;  
 @FXML public Circle circle2;  
 @FXML public Circle circle3;  
  
 @FXML public Label fixer1;  
 @FXML public Label fixer2;  
 @FXML public Label fixer3;  
  
 @FXML public Label allOrders;  
 @FXML public Label clientData;  
 @FXML public Label clientData2;  
  
 Shelf shelf;  
 Receiver receiver;  
 Fixer[] fixers=new Fixer[3];  
 int addButton = 0;  
  
 public void addOrderButton(ActionEvent event) throws IOException {  
  
 receiver.addOrderButton();  
 Platform.*runLater*(()->{  
 allOrders.setText(Integer.*toString*(receiver.numberOfOrders));  
 });  
 fixers[addButton].addOrderButton();  
 if(addButton<2){  
 addButton++;  
 }  
 else {addButton=0;}  
 }

}else {rect.setWidth(53);}  
 rect.setX(car.x);  
 rect.setY(300);  
 rect.setFill(car.color);  
  
 car.transition.setNode(rect);  
 car.transition.setDuration(Duration.*millis*(4000));  
 car.transition.setByX(948-car.x);  
 car.transition.play();  
  
 }  
 @Override  
 public void initialize(URL url, ResourceBundle resourceBundle) {  
  
 this.config = new Config("E:/GotoweProjekty/Loading\_GUI/src/resources/config.properties");  
 initializeTextFields();  
 initializeLoaderLabels();  
 initializeCords();  
  
 }  
 private void initializeLoaderLabels(){  
 loader1Status.setText("LOADER 1");  
 loader2Status.setText("LOADER 2");  
 loader3Status.setText("LOADER 3");  
 loader4Status.setText("LOADER 4");  
 }  
 private void setLoaderLabels(String text){  
 loader1Status.setText(text);  
 loader2Status.setText(text);  
 loader3Status.setText(text);  
 loader4Status.setText(text);  
 }  
 private void initializeTextFields(){  
 this.lowCapacityCarsTextField.setText(Integer.*toString*( config.lowCapacityCars));  
 this.highCapacityCarsTextField.setText(Integer.*toString*( config.highCapacityCars));  
 this.startWorkHourTextField.setText(Integer.*toString*(config.startWorkHour));  
 this.endWorkHourTextField.setText(Integer.*toString*( config.endWorkHour));  
 this.startBreakHourTextField.setText(Integer.*toString*( config.startBreakHour));  
 this.endBreakHourTextField.setText(Integer.*toString*( config.endBreakHour));  
 this.joiningQueueMaxDelayTextField.setText(Integer.*toString*( config.joiningQueueMaxDelay));  
 this.joiningQueueMinDelayTextField.setText(Integer.*toString*( config.joiningQueueMinDelay));  
 this.loadingMaxDelayTextField.setText(Integer.*toString*( config.loadingCarMaxDelay));  
 this.loadingMinDelayTextField.setText(Integer.*toString*( config.loadingCarMinDelay));  
 this.minutesToResignTextField.setText(Integer.*toString*( config.minutesAfterDriversResign));  
 }  
 public void getValuesFromTextFields(){  
 this.config.lowCapacityCars =Integer.*parseInt*( lowCapacityCarsTextField.getText());  
 this.config.highCapacityCars =Integer.*parseInt*( highCapacityCarsTextField.getText());  
 this.config.startWorkHour =Integer.*parseInt*( startWorkHourTextField.getText());  
 this.config.endWorkHour =Integer.*parseInt*( endWorkHourTextField.getText());  
 this.config.startBreakHour =Integer.*parseInt*( startBreakHourTextField.getText());  
 this.config.endBreakHour =Integer.*parseInt*( endBreakHourTextField.getText());  
 this.config.joiningQueueMaxDelay =Integer.*parseInt*( joiningQueueMaxDelayTextField.getText());  
 this.config.joiningQueueMinDelay=Integer.*parseInt*( joiningQueueMinDelayTextField.getText());  
 this.config.loadingCarMaxDelay =Integer.*parseInt*( loadingMaxDelayTextField.getText());  
 this.config.loadingCarMinDelay =Integer.*parseInt*( loadingMinDelayTextField.getText());  
 this.config.minutesAfterDriversResign =Integer.*parseInt*( minutesToResignTextField.getText());  
 }  
 private void initializeCords(){  
 loadersX1cords[0]=36;  
 loadersX1cords[1]=290;  
 loadersX1cords[2]=540;  
 loadersX1cords[3]=770;  
  
 loadersX2cords[0]=67;  
 loadersX2cords[1]=321;  
 loadersX2cords[2]=570;  
 loadersX2cords[3]=801;  
  
 loadersYCords[0]=463;  
 loadersYCords[1]=458;  
 loadersYCords[2]=458;  
 loadersYCords[3]=454;  
 }  
 private Rectangle spawnCar(int x, int y, String capacity, Color color, Pane pane){  
 Rectangle rect = new Rectangle();  
 rect.setHeight(15);  
 if(capacity.equals("Low")) {  
 rect.setWidth(22);  
 }else {rect.setWidth(53);}  
 rect.setX(x);  
 rect.setY(y);  
 rect.setFill(color);  
 if(x<900) {  
 pane.getChildren().add(rect);  
 }  
 return rect;  
 }  
 public void stopSimulation(ActionEvent actionEvent) {  
 application.workClock.on = false;  
 for(int i=0;i<4;i++) {  
 application.loaders[i].retCars.clear();  
 }  
 application.loadedCars.clear();  
 application.queue.cars.clear();  
 application.queue.resignedCars.clear();  
  
 clearCargo();  
 clearQueue();  
 clearResignedCars();  
 updateLoaders();  
  
 setLoaderLabels("Stopped");  
 }  
 public void clearCargo(){  
 Platform.*runLater*(() -> {  
 Rectangle rect = new Rectangle();  
 rect.setHeight(100);  
 rect.setWidth(870);  
 rect.setX(0);  
 rect.setY(625);  
 rect.setFill(Color.*rgb*(149,117,80));  
 simulationPane.getChildren().add(rect);  
 });  
 }  
 public void startSimulation(ActionEvent actionEvent){  
  
 getValuesFromTextFields();  
 this.application=new Application(this,config);  
 application.start();  
 this.queue = application.queue;  
  
 }  
 private void updateLoaders(){  
  
 if(application!=null && application.workClock.workTime==true) {  
  
 for (int i = 0; i < application.loaders.length; i++) {  
  
 clearLoader(loadersX1cords[i], loadersYCords[i]);  
 if (application.loaders[i].retCars.size()!=0) {  
 spawnCar(loadersX1cords[i], loadersYCords[i], application.loaders[i].retCars.get(0).capacity, application.loaders[i].retCars.get(0).color, simulationPane);  
 if (application.loaders[i].retCars.size() == 2) {  
 spawnCar(loadersX2cords[i], loadersYCords[i], application.loaders[i].retCars.get(1).capacity, application.loaders[i].retCars.get(1).color,simulationPane);  
 }  
 }  
 }  
 }  
 }  
 private void updateCargo(){  
  
 String tempCapacity;  
 int cargoX = 10;  
 int cargoY = 630;  
  
 if(application.loadedCars.size()!=0){  
  
 for(int i=0; i<application.loadedCars.size();i++){  
 if(cargoX>830){  
 cargoX=10;  
 cargoY+=20;  
 }  
 tempCapacity = application.loadedCars.get(i).capacity;  
 spawnCar(cargoX,cargoY,tempCapacity,application.loadedCars.get(i).color,simulationPane);  
 if(tempCapacity=="High"){  
 cargoX +=53;  
 }  
 else{cargoX+=25;}  
 }  
 }  
 }  
 private void clearLoader(int x, int y){  
 spawnCar(x,y,"High",Color.*rgb*(104,88,108),simulationPane);  
 }  
 public void updateQueue(){  
  
 clearQueue();  
 int x = 50;  
 final int y = 260;  
  
 for(int i=0; i<queue.cars.size();i++){  
 final int fx = x;  
 final Car car = queue.cars.get(i);  
 car.x = x;  
 car.y = y;  
 Platform.*runLater*(() -> {  
 car.rectangle= spawnCar(fx, y, car.capacity, car.color, simulationPane);  
 });  
  
 if(car.capacity.equals("Low")){  
 x+=25;  
 }  
 else{  
 x+=50;  
 }  
  
 }  
 }  
 private void spawnResignedCar(int x, int y, String capacity, Color color, Pane pane){  
  
 Rectangle rect = new Rectangle();  
 rect.setWidth(15);  
 if(capacity.equals("Low")) {  
 rect.setHeight(22);  
 }else {rect.setHeight(44);}  
 rect.setX(x);  
 rect.setY(y);  
 rect.setFill(color);  
 pane.getChildren().add(rect);  
 }  
 public void updateResignedCars(){  
 int x = 5;  
 int y = 40;  
  
 for(int i=0; i<queue.resignedCars.size();i++){  
 final int fx = x;  
 final int fy = y;  
 final Car car = queue.resignedCars.get(i);  
  
 Platform.*runLater*(() -> {  
 spawnResignedCar(fx,fy,car.capacity,car.color,resignedCarsPane);  
 });  
 if(x<100) {  
 x += 18;  
 }  
 else {  
 x=5;  
 y+=50;  
 }  
 }  
 }  
 private void clearResignedCars(){  
 Platform.*runLater*(() -> {  
 Rectangle rect = new Rectangle();  
 rect.setHeight(800);  
 rect.setWidth(130);  
 rect.setX(0);  
 rect.setY(30);  
 rect.setFill(Color.*GRAY*);  
 resignedCarsPane.getChildren().add(rect);  
 });  
 }  
 private void clearQueue(){  
 Platform.*runLater*(() -> {  
 Rectangle rect = new Rectangle();  
 rect.setHeight(15);  
 rect.setWidth(950);  
 rect.setX(50);  
 rect.setY(260);  
 rect.setFill(Color.*rgb*(59,59,59));  
 simulationPane.getChildren().add(rect);  
 });  
 }  
 public void setTime(int hour, int minute) {  
  
 String textHour;  
 String textMinute;  
 if (hour < 10 || hour == 0) {  
 textHour = '0' + Integer.*toString*(hour);  
 } else {  
 textHour = Integer.*toString*(hour);  
 }  
 if (minute < 10 || minute == 0) {  
 textMinute = '0' + Integer.*toString*(minute);  
 } else {  
 textMinute = Integer.*toString*(minute);  
 }  
 Platform.*runLater*(() -> {  
 time.setText(textHour + ":" + textMinute);  
 updateQueue();  
 updateLoaders();  
 updateCargo();  
 updateLabels();  
 updateResignedCars();  
 });  
 }  
 private void updateLabels(){  
 if(application.workClock.workTime==true && application.workClock.breakTime==false){  
 setLoaderLabels("Working...");  
 }  
 if(application.workClock.breakTime==true && application.workClock.workTime==true){  
 setLoaderLabels("Break time");  
 }  
 if(application.workClock.breakTime==false && application.workClock.workTime==false){  
 initializeLoaderLabels();  
 }  
 }  
  
  
}

joiningQueueMaxDelayTextField.getText());  
 this.config.joiningQueueMinDelay=Integer.*parseInt*( joiningQueueMinDelayTextField.getText());  
 this.config.loadingCarMaxDelay =Integer.*parseInt*( loadingMaxDelayTextField.getText());  
 this.config.loadingCarMinDelay =Integer.*parseInt*( loadingMinDelayTextField.getText());  
 this.config.minutesAfterDriversResign =Integer.*parseInt*( minutesToResignTextField.getText());  
 }  
 private void initializeCords(){  
 loadersX1cords[0]=36;  
 loadersX1cords[1]=290;  
 loadersX1cords[2]=540;  
 loadersX1cords[3]=770;  
  
 loadersX2cords[0]=67;  
 loadersX2cords[1]=321;  
 loadersX2cords[2]=570;  
 loadersX2cords[3]=801;  
  
 loadersYCords[0]=463;  
 loadersYCords[1]=458;  
 loadersYCords[2]=458;  
 loadersYCords[3]=454;  
 }  
 private Rectangle spawnCar(int x, int y, String capacity, Color color, Pane pane){  
 Rectangle rect = new Rectangle();  
 rect.setHeight(15);  
 if(capacity.equals("Low")) {  
 rect.setWidth(22);  
 }else {rect.setWidth(53);}  
 rect.setX(x);  
 rect.setY(y);  
 rect.setFill(color);  
 if(x<900) {  
 pane.getChildren().add(rect);  
 }  
 return rect;  
 }  
 public void stopSimulation(ActionEvent actionEvent) {  
 application.workClock.on = false;  
 for(int i=0;i<4;i++) {  
 application.loaders[i].retCars.clear();  
 }  
 application.loadedCars.clear();  
 application.queue.cars.clear();  
 application.queue.resignedCars.clear();  
  
 clearCargo();  
 clearQueue();  
 clearResignedCars();  
 updateLoaders();  
  
 setLoaderLabels("Stopped");  
 }  
 public void clearCargo(){  
 Platform.*runLater*(() -> {  
 Rectangle rect = new Rectangle();  
 rect.setHeight(100);  
 rect.setWidth(870);  
 rect.setX(0);  
 rect.setY(625);  
 rect.setFill(Color.*rgb*(149,117,80));  
 simulationPane.getChildren().add(rect);  
 });  
 }  
 public void startSimulation(ActionEvent actionEvent){  
  
 getValuesFromTextFields();  
 this.application=new Application(this,config);  
 application.start();  
 this.queue = application.queue;  
  
 }  
 private void updateLoaders(){  
  
 if(application!=null && application.workClock.workTime==true) {  
  
 for (int i = 0; i < application.loaders.length; i++) {  
  
 clearLoader(loadersX1cords[i], loadersYCords[i]);  
 if (application.loaders[i].retCars.size()!=0) {  
 spawnCar(loadersX1cords[i], loadersYCords[i], application.loaders[i].retCars.get(0).capacity, application.loaders[i].retCars.get(0).color, simulationPane);  
 if (application.loaders[i].retCars.size() == 2) {  
 spawnCar(loadersX2cords[i], loadersYCords[i], application.loaders[i].retCars.get(1).capacity, application.loaders[i].retCars.get(1).color,simulationPane);  
 }  
 }  
 }  
 }  
 }  
 private void updateCargo(){  
  
 String tempCapacity;  
 int cargoX = 10;  
 int cargoY = 630;  
  
 if(application.loadedCars.size()!=0){  
  
 for(int i=0; i<application.loadedCars.size();i++){  
 if(cargoX>830){  
 cargoX=10;  
 cargoY+=20;  
 }  
 tempCapacity = application.loadedCars.get(i).capacity;  
 spawnCar(cargoX,cargoY,tempCapacity,application.loadedCars.get(i).color,simulationPane);  
 if(tempCapacity=="High"){  
 cargoX +=53;  
 }  
 else{cargoX+=25;}  
 }  
 }  
 }  
 private void clearLoader(int x, int y){  
 spawnCar(x,y,"High",Color.*rgb*(104,88,108),simulationPane);  
 }  
 public void updateQueue(){  
  
 clearQueue();  
 int x = 50;  
 final int y = 260;  
  
 for(int i=0; i<queue.cars.size();i++){  
 final int fx = x;  
 final Car car = queue.cars.get(i);  
 car.x = x;  
 car.y = y;  
 Platform.*runLater*(() -> {  
 car.rectangle= spawnCar(fx, y, car.capacity, car.color, simulationPane);  
 });  
  
 if(car.capacity.equals("Low")){  
 x+=25;  
 }  
 else{  
 x+=50;  
 }  
  
 }  
 }  
 private void spawnResignedCar(int x, int y, String capacity, Color color, Pane pane){  
  
 Rectangle rect = new Rectangle();  
 rect.setWidth(15);  
 if(capacity.equals("Low")) {  
 rect.setHeight(22);  
 }else {rect.setHeight(44);}  
 rect.setX(x);  
 rect.setY(y);  
 rect.setFill(color);  
 pane.getChildren().add(rect);  
 }  
 public void updateResignedCars(){  
 int x = 5;  
 int y = 40;  
  
 for(int i=0; i<queue.resignedCars.size();i++){  
 final int fx = x;  
 final int fy = y;  
 final Car car = queue.resignedCars.get(i);  
  
 Platform.*runLater*(() -> {  
 spawnResignedCar(fx,fy,car.capacity,car.color,resignedCarsPane);  
 });  
 if(x<100) {  
 x += 18;  
 }  
 else {  
 x=5;  
 y+=50;  
 }  
 }  
 }  
 private void clearResignedCars(){  
 Platform.*runLater*(() -> {  
 Rectangle rect = new Rectangle();  
 rect.setHeight(800);  
 rect.setWidth(130);  
 rect.setX(0);  
 rect.setY(30);  
 rect.setFill(Color.*GRAY*);  
 resignedCarsPane.getChildren().add(rect);  
 });  
 }  
 private void clearQueue(){  
 Platform.*runLater*(() -> {  
 Rectangle rect = new Rectangle();  
 rect.setHeight(15);  
 rect.setWidth(950);  
 rect.setX(50);  
 rect.setY(260);  
 rect.setFill(Color.*rgb*(59,59,59));  
 simulationPane.getChildren().add(rect);  
 });  
 }  
 public void setTime(int hour, int minute) {  
  
 String textHour;  
 String textMinute;  
 if (hour < 10 || hour == 0) {  
 textHour = '0' + Integer.*toString*(hour);  
 } else {  
 textHour = Integer.*toString*(hour);  
 }  
 if (minute < 10 || minute == 0) {  
 textMinute = '0' + Integer.*toString*(minute);  
 } else {  
 textMinute = Integer.*toString*(minute);  
 }  
 Platform.*runLater*(() -> {  
 time.setText(textHour + ":" + textMinute);  
 updateQueue();  
 updateLoaders();  
 updateCargo();  
 updateLabels();  
 updateResignedCars();  
 });  
 }  
 private void updateLabels(){  
 if(application.workClock.workTime==true && application.workClock.breakTime==false){  
 setLoaderLabels("Working...");  
 }  
 if(application.workClock.breakTime==true && application.workClock.workTime==true){  
 setLoaderLabels("Break time");  
 }  
 if(application.workClock.breakTime==false && application.workClock.workTime==false){  
 initializeLoaderLabels();  
 }  
 }  
  
  
}

public void startSimulation(ActionEvent actionEvent){  
  
 getValuesFromTextFields();  
 this.application=new Application(this,config);  
 application.start();  
 this.queue = application.queue;  
  
 }  
 private void updateLoaders(){  
  
 if(application!=null && application.workClock.workTime==true) {  
  
 for (int i = 0; i < application.loaders.length; i++) {  
  
 clearLoader(loadersX1cords[i], loadersYCords[i]);  
 if (application.loaders[i].retCars.size()!=0) {  
 spawnCar(loadersX1cords[i], loadersYCords[i], application.loaders[i].retCars.get(0).capacity, application.loaders[i].retCars.get(0).color, simulationPane);  
 if (application.loaders[i].retCars.size() == 2) {  
 spawnCar(loadersX2cords[i], loadersYCords[i], application.loaders[i].retCars.get(1).capacity, application.loaders[i].retCars.get(1).color,simulationPane);  
 }  
 }  
 }  
 }  
 }  
 private void updateCargo(){  
  
 String tempCapacity;  
 int cargoX = 10;  
 int cargoY = 630;  
  
 if(application.loadedCars.size()!=0){  
  
 for(int i=0; i<application.loadedCars.size();i++){  
 if(cargoX>830){  
 cargoX=10;  
 cargoY+=20;  
 }  
 tempCapacity = application.loadedCars.get(i).capacity;  
 spawnCar(cargoX,cargoY,tempCapacity,application.loadedCars.get(i).color,simulationPane);  
 if(tempCapacity=="High"){  
 cargoX +=53;  
 }  
 else{cargoX+=25;}  
 }  
 }  
 }  
 private void clearLoader(int x, int y){  
 spawnCar(x,y,"High",Color.*rgb*(104,88,108),simulationPane);  
 }  
 public void updateQueue(){  
  
 clearQueue();  
 int x = 50;  
 final int y = 260;  
  
 for(int i=0; i<queue.cars.size();i++){  
 final int fx = x;  
 final Car car = queue.cars.get(i);  
 car.x = x;  
 car.y = y;  
 Platform.*runLater*(() -> {  
 car.rectangle= spawnCar(fx, y, car.capacity, car.color, simulationPane);  
 });  
  
 if(car.capacity.equals("Low")){  
 x+=25;  
 }  
 else{  
 x+=50;  
 }  
  
 }  
 }  
 private void spawnResignedCar(int x, int y, String capacity, Color color, Pane pane){  
  
 Rectangle rect = new Rectangle();  
 rect.setWidth(15);  
 if(capacity.equals("Low")) {  
 rect.setHeight(22);  
 }else {rect.setHeight(44);}  
 rect.setX(x);  
 rect.setY(y);  
 rect.setFill(color);  
 pane.getChildren().add(rect);  
 }  
 public void updateResignedCars(){  
 int x = 5;  
 int y = 40;  
  
 for(int i=0; i<queue.resignedCars.size();i++){  
 final int fx = x;  
 final int fy = y;  
 final Car car = queue.resignedCars.get(i);  
  
 Platform.*runLater*(() -> {  
 spawnResignedCar(fx,fy,car.capacity,car.color,resignedCarsPane);  
 });  
 if(x<100) {  
 x += 18;  
 }  
 else {  
 x=5;  
 y+=50;  
 }  
 }  
 }  
 private void clearResignedCars(){  
 Platform.*runLater*(() -> {  
 Rectangle rect = new Rectangle();  
 rect.setHeight(800);  
 rect.setWidth(130);  
 rect.setX(0);  
 rect.setY(30);  
 rect.setFill(Color.*GRAY*);  
 resignedCarsPane.getChildren().add(rect);  
 });  
 }  
 private void clearQueue(){  
 Platform.*runLater*(() -> {  
 Rectangle rect = new Rectangle();  
 rect.setHeight(15);  
 rect.setWidth(950);  
 rect.setX(50);  
 rect.setY(260);  
 rect.setFill(Color.*rgb*(59,59,59));  
 simulationPane.getChildren().add(rect);  
 });  
 }  
 public void setTime(int hour, int minute) {  
  
 String textHour;  
 String textMinute;  
 if (hour < 10 || hour == 0) {  
 textHour = '0' + Integer.*toString*(hour);  
 } else {  
 textHour = Integer.*toString*(hour);  
 }  
 if (minute < 10 || minute == 0) {  
 textMinute = '0' + Integer.*toString*(minute);  
 } else {  
 textMinute = Integer.*toString*(minute);  
 }  
 Platform.*runLater*(() -> {  
 time.setText(textHour + ":" + textMinute);  
 updateQueue();  
 updateLoaders();  
 updateCargo();  
 updateLabels();  
 updateResignedCars();  
 });  
 }  
 private void updateLabels(){  
 if(application.workClock.workTime==true && application.workClock.breakTime==false){  
 setLoaderLabels("Working...");  
 }  
 if(application.workClock.breakTime==true && application.workClock.workTime==true){  
 setLoaderLabels("Break time");  
 }  
 if(application.workClock.breakTime==false && application.workClock.workTime==false){  
 initializeLoaderLabels();  
 }  
 }  
  
  
}

car.rectangle= spawnCar(fx, y, car.capacity, car.color, simulationPane);  
 });  
  
 if(car.capacity.equals("Low")){  
 x+=25;  
 }  
 else{  
 x+=50;  
 }  
  
 }  
 }  
 private void spawnResignedCar(int x, int y, String capacity, Color color, Pane pane){  
  
 Rectangle rect = new Rectangle();  
 rect.setWidth(15);  
 if(capacity.equals("Low")) {  
 rect.setHeight(22);  
 }else {rect.setHeight(44);}  
 rect.setX(x);  
 rect.setY(y);  
 rect.setFill(color);  
 pane.getChildren().add(rect);  
 }  
 public void updateResignedCars(){  
 int x = 5;  
 int y = 40;  
  
 for(int i=0; i<queue.resignedCars.size();i++){  
 final int fx = x;  
 final int fy = y;  
 final Car car = queue.resignedCars.get(i);  
  
 Platform.*runLater*(() -> {  
 spawnResignedCar(fx,fy,car.capacity,car.color,resignedCarsPane);  
 });  
 if(x<100) {  
 x += 18;  
 }  
 else {  
 x=5;  
 y+=50;  
 }  
 }  
 }  
 private void clearResignedCars(){  
 Platform.*runLater*(() -> {  
 Rectangle rect = new Rectangle();  
 rect.setHeight(800);  
 rect.setWidth(130);  
 rect.setX(0);  
 rect.setY(30);  
 rect.setFill(Color.*GRAY*);  
 resignedCarsPane.getChildren().add(rect);  
 });  
 }  
 private void clearQueue(){  
 Platform.*runLater*(() -> {  
 Rectangle rect = new Rectangle();  
 rect.setHeight(15);  
 rect.setWidth(950);  
 rect.setX(50);  
 rect.setY(260);  
 rect.setFill(Color.*rgb*(59,59,59));  
 simulationPane.getChildren().add(rect);  
 });  
 }  
 public void setTime(int hour, int minute) {  
  
 String textHour;  
 String textMinute;  
 if (hour < 10 || hour == 0) {  
 textHour = '0' + Integer.*toString*(hour);  
 } else {  
 textHour = Integer.*toString*(hour);  
 }  
 if (minute < 10 || minute == 0) {  
 textMinute = '0' + Integer.*toString*(minute);  
 } else {  
 textMinute = Integer.*toString*(minute);  
 }  
 Platform.*runLater*(() -> {  
 time.setText(textHour + ":" + textMinute);  
 updateQueue();  
 updateLoaders();  
 updateCargo();  
 updateLabels();  
 updateResignedCars();  
 });  
 }  
 private void updateLabels(){  
 if(application.workClock.workTime==true && application.workClock.breakTime==false){  
 setLoaderLabels("Working...");  
 }  
 if(application.workClock.breakTime==true && application.workClock.workTime==true){  
 setLoaderLabels("Break time");  
 }  
 if(application.workClock.breakTime==false && application.workClock.workTime==false){  
 initializeLoaderLabels();  
 }  
 }  
  
  
}

});  
 }  
 public void setTime(int hour, int minute) {  
  
 String textHour;  
 String textMinute;  
 if (hour < 10 || hour == 0) {  
 textHour = '0' + Integer.*toString*(hour);  
 } else {  
 textHour = Integer.*toString*(hour);  
 }  
 if (minute < 10 || minute == 0) {  
 textMinute = '0' + Integer.*toString*(minute);  
 } else {  
 textMinute = Integer.*toString*(minute);  
 }  
 Platform.*runLater*(() -> {  
 time.setText(textHour + ":" + textMinute);  
 updateQueue();  
 updateLoaders();  
 updateCargo();  
 updateLabels();  
 updateResignedCars();  
 });  
 }  
 private void updateLabels(){  
 if(application.workClock.workTime==true && application.workClock.breakTime==false){  
 setLoaderLabels("Working...");  
 }  
 if(application.workClock.breakTime==true && application.workClock.workTime==true){  
 setLoaderLabels("Break time");  
 }  
 if(application.workClock.breakTime==false && application.workClock.workTime==false){  
 initializeLoaderLabels();  
 }  
 }  
  
  
}

1. **Interfejs Graficzny**

Wykonany za pomocą biblioteki javaFX.

Ustawienia domyślne pobierane są z pliku data.properties.

Jeśli użytkownik nie wpisze danych do lub wpisze nieprawidłowe, zostanie wywołana symulacja z ustawieniami domyślnymi z pliku data.properties.





