***Wojskowa Akademia Techniczna***

***im. Jarosława Dąbrowskiego***

Sprawozdanie z projektu semestralnego z przedmiotu:

[Programowanie w](http://shaql.w.staszic.waw.pl/~shaql/wcy/viewforum.php?f=13)spółbieżne

Sprawozdanie z projektu nr 11:

**Statek**

Spis treści

[Treść zadania 2](#_Toc94014438)

[Opis problemu 2](#_Toc94014439)

[Wytyczne: 2](#_Toc94014440)

[Założenia: 2](#_Toc94014441)

[Współdzielone zasoby 3](#_Toc94014442)

[Punkty synchronizacji 3](#_Toc94014443)

[Ochrona dostępu do kolejki pasażerów 3](#_Toc94014444)

[Ochrona dostępu do mostka 4](#_Toc94014445)

[Ochrona dostępu do statku 4](#_Toc94014446)

[Obiekty synchronizacji 5](#_Toc94014447)

[Procesy sekwencyjne 5](#_Toc94014448)

[Zarządzanie listą pasażerów na statku 5](#_Toc94014449)

[Listing programu 5](#_Toc94014450)

[Kod inicjujący program (Main.java) 5](#_Toc94014451)

[Kod zarządzający interfejsem programu (MainController.java) 6](#_Toc94014452)

[Kod zarządzający wartościami początkowymi (Properties.java) 8](#_Toc94014453)

[Kod zarządzający ustawieniami programu (PropertiesController.java) 10](#_Toc94014454)

[Kod powołujący procesy programu (start.java) 12](#_Toc94014455)

[Kod wykonujący zadania kapitana 14](#_Toc94014456)

[Kod wykonujący zadania pasażera 16](#_Toc94014457)

# Treść zadania

Obraz zawierający tekst

Opis wygenerowany automatycznie

# Opis problemu

### Wytyczne:

1. Pojemność statku i mostka mają wartości nadawane na początku programu.
2. Pojemność statku jest większa od pojemności mostka.
3. Statek cyklicznie wyrusza w rejs, kapitan zarządza rejsami.
4. Aby statek mógł odpłynąć, na mostku nie może być pasażerów.
5. Kapitan oraz Pasażer to osobne procesy, które muszą być zsynchronizowane.

### Założenia:

1. Pojemność statku oraz pojemność mostka deklarowana jest za każdym razem w momencie uruchamiania programu.
2. Kapitan posiada 2 mechanizmy oczekiwania na rejs:
   1. Cykliczny – co określony czas.
   2. Aktywny – obserwuje zapełnienie statku, w momencie, gdy wszystkie miejsca są zajęte rozpoczyna rejs wcześniej.
3. Kapitan zarządza 3 stanami:
   1. Ładowanie statku.
   2. Rejs.
   3. Koniec działania programu.
4. Ładowanie statku opiera się na powoływaniu wątków pasażera, którzy będą ubiegać się o miejsce na statku.
5. Rejs opiera się na zablokowaniu dostępu do statku i odbyciu rejsu, który trwa określony czas.
6. Przed rejsem kapitan po kolei:
   1. Blokuje pasażerom dostęp do statku.
   2. wstrzymuje powoływanie kolejnych pasażerów (celem ochrony zasobów sprzętowych).
   3. usuwa wszystkich pasażerów z mostka.
   4. Zmienia status statku na „w trakcie rejsu”.
7. Po rejsie kapitan:
   1. Usuwa wszystkich pasażerów ze statku
   2. Ustawia wszystkie miejsca na statku jako wolne
   3. Wznawia proces powoływania pasażerów
8. Synchronizacja odbywa się poprzez kolejkę – nowi pasażerowie dodawani są do kolejki, na podstawie której starają się o miejsce na statku
9. Zarządzanie dostępem do sekcji krytycznej (statku i mostka) odbywa się poprzez konkurencję o dostęp do niej (przy użyciu semafora).

# Współdzielone zasoby

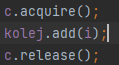
Zasobami współdzielonymi w programie są mostek oraz statek. Za zarządzanie mostkiem odpowiada tablica semaforów wielkości pojemności mostka, pozwalająca pasażerom na kolejne pokonywanie każdego miejsca na mostku, aż do dostania się do statku. Za zarządzanie statkiem odpowiada semafor o liczbie dostępnych miejsc równej pojemności statku.

# Punkty synchronizacji

### Ochrona dostępu do kolejki pasażerów

Odpowiada za niego semafor „c”

* Obsługa interrupta w przypadku zakończenia symulacji Obraz zawierający tekst

  Opis wygenerowany automatycznie
* Dodawanie indeksu pasażera do kolejki 
* Dodawanie pasażera do listy pasażerów na statkuObraz zawierający tekst

  Opis wygenerowany automatycznie

### Ochrona dostępu do mostka

Obraz zawierający tekst

Opis wygenerowany automatycznie

* Zajmowanie pierwszego miejsca na mostku
* Przechodzenie po kolei do kolejnych miejsc na mostku
* Kiedy następne miejsce mostka jest wolne, skończ przechodzenie
* Jeśli jest na ostatnim punkcie mostka, próbuje się dostać na statek
* Jeśli wejdzie na statek, zwalnia ostatnie miejsce na mostku

### Ochrona dostępu do statku

* Wejście na statekObraz zawierający tekst

  Opis wygenerowany automatycznie
* Zwalnianie miejsc na statku (kapitan) 

# Obiekty synchronizacji

1. Semafon c
   1. Semafor kontrolujący dostęp do listy pasażerów
   2. Listą pasażerów nie może być obsługiwana na raz przez kilka wątków
2. Semafor[] mostek
   1. Tablica semaforów zarządzająca dostępem do każdego miejsca mostka
   2. Na każdym punkcie mostka może być 1 pasażer
3. Semafor statek
   1. Semafor kontrolujący ilość dostępnych miejsc na statku
   2. Na statku może być maksymalnie konkretna ilość pasażerów
   3. Po rejsie wszystkie wcześniej zajęte miejsca na statku zostają zwolnione

# Procesy sekwencyjne

### Zarządzanie listą pasażerów na statku

* Dodawanie pasażera po zajęciu miejsca na statku

Obraz zawierający tekst

Opis wygenerowany automatycznie

* Usuwanie pasażerów ze statku po rejsie

Obraz zawierający tekst

Opis wygenerowany automatycznie

# Listing programu

### Kod inicjujący program (Main.java)

package com.example.statek;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
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("main-view.fxml")); //Nowy FXML  
 Scene scene = new Scene(fxmlLoader.load(), 1000, 500); //Nowa scena (okno) w utworzonym FXML  
 stage.setResizable(false); //Nie wolno zmieniać rozmiaru  
 stage.setTitle("Statek"); //Tytuł okna  
 stage.setScene(scene); //Ustaw scene jako scene dla okna  
 stage.show(); //Pokaz scene (otwrorz)  
 }  
  
 public static void main(String[] args) {  
 *launch*(); //lauch app  
 }  
}

### Kod zarządzający interfejsem programu (MainController.java)

package com.example.statek;  
  
import javafx.event.ActionEvent;  
import javafx.event.EventHandler;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.layout.Pane;  
import javafx.scene.paint.Color;  
import javafx.scene.shape.Line;  
import javafx.scene.shape.Rectangle;  
import javafx.scene.shape.StrokeType;  
import javafx.scene.text.Font;  
import javafx.scene.text.Text;  
import javafx.stage.Modality;  
import javafx.stage.Stage;  
import javafx.stage.WindowEvent;  
  
import java.io.IOException;  
import java.util.ArrayList;  
  
public class MainController {  
  
 static boolean *interupt*;  
 public Pane pane;  
 static Pane *paneS*;  
  
 public Button btnStart, btnStop, btnProperties; //Przyciski w oknie  
  
 static int *bokRectagle* = 40; //Pole na pasażerów  
 static ArrayList<Rectangle> *statekConteiners*; //Miejsca na statku  
 static ArrayList<Rectangle> *mostekConteiners*; //Miejsca na mostku  
 Rectangle rectangle;  
  
 static ArrayList<start> *starts*; //  
  
 Text text;  
 static Text *textStatus*; //Wyświetlany na dole tekstu  
  
 public void startSymulation(ActionEvent actionEvent) {  
 btnStart.setDisable(true);  
 btnStop.setDisable(false);  
 btnProperties.setDisable(true);  
  
 *interupt*=false; //Nie kończ programu  
  
 *starts*=new ArrayList<>(); //Lista pasażerów  
 *paneS*=pane;  
 start start = new start(); //Nowi pasażerowie  
 start.start(); //Zacznij pasażerów  
 *starts*.add(start); //Dodaj start  
 }  
  
 public void stopSymulation(ActionEvent actionEvent) {  
 btnStart.setDisable(true); //setDisable(true) - nie można wcisnąć przycisku  
 btnStop.setDisable(true);  
 btnProperties.setDisable(false);  
  
  
 *interupt*=true; //Zakończ działanie  
 start.*kapitan*.interrupt(); //Zakończ kapitana  
 for (int i=0; i<*starts*.size(); i++) *starts*.get(i).interrupt(); //Zakończ metody start  
 }  
  
 public void properties(ActionEvent actionEvent) throws IOException {  
 btnStart.setDisable(false); //setDisable(true) - nie można wcisnąć przycisku  
 btnStop.setDisable(true);  
 btnProperties.setDisable(true);  
  
 FXMLLoader fxmlLoader = new FXMLLoader(Main.class.getResource("properties-view.fxml")); //Nowe ustawienia  
 Scene scene = new Scene(fxmlLoader.load(), 400, 310); //Nowe okno  
 Stage stage = new Stage(); //Nowa scena w oknie  
 stage.initModality(Modality.*APPLICATION\_MODAL*); //Nie możesz działać w głównym oknie  
 stage.setResizable(false); //Nie możesz zmieniać rozmiaru  
 stage.setTitle("Ustawienia"); //Nazwa okna  
 stage.setScene(scene); //Nowa scena  
  
 stage.setOnHidden(windowEvent -> { //DEKLARACJA - Po wciśnięciu zastosuj lub zamknij - generuj obiekty w oknie  
 pane.getChildren().clear(); //Wszystkie podelementy pane'a (Childres - pojedyncze rysunki)  
 *statekConteiners*=new ArrayList<>(); //Kwadraty statku  
 *mostekConteiners*=new ArrayList<>(); //Kwadraty mostka  
 int X=10;  
 int odstep=20; //P(X,Y) - położenie kwadratu  
 double Y=pane.getHeight()/2-*bokRectagle*\*PropertiesController.*properties*.getN()/5/2-odstep\*(PropertiesController.*properties*.getN()-2)/5/2;  
 int j=0;  
 for (int i=0; i<PropertiesController.*properties*.getN(); i++){ //Tworzy kwadraty na współrzędnych  
 rectangle=new Rectangle(X+odstep\*j+j\**bokRectagle*,Y, *bokRectagle*, *bokRectagle*); //Tutaj jest zmiana współrzędnej kwadratu  
 rectangle.setFill(Color.*TRANSPARENT*);  
 rectangle.setStrokeType(StrokeType.*INSIDE*);  
 rectangle.setStrokeWidth(1);  
 rectangle.setStroke(Color.*BLACK*);  
 *statekConteiners*.add(rectangle);  
  
 pane.getChildren().add(rectangle);  
  
 j++;  
 if(j==5){  
 X=10;  
 j=0;  
 Y+=*bokRectagle*+odstep; //Następne położenie kwadratu  
 }  
 }  
 //PRZELICZANIE DLA MOSTKA  
 X=10+5\*odstep+5\**bokRectagle*;  
 Y=pane.getHeight()/2-*bokRectagle*/2; //P(X,Y) - połozenie kwadratu  
 for (int i=0; i<PropertiesController.*properties*.getK(); i++){  
 rectangle=new Rectangle(X+odstep\*i+i\**bokRectagle*,Y, *bokRectagle*, *bokRectagle*); //Tutaj jest zmiana współrzędnej kwadratu  
 rectangle.setFill(Color.*TRANSPARENT*);  
 rectangle.setStrokeType(StrokeType.*INSIDE*);  
 rectangle.setStrokeWidth(1);  
 rectangle.setStroke(Color.*BLACK*);  
 *mostekConteiners*.add(rectangle);  
  
 pane.getChildren().add(rectangle);  
  
 }  
 //Tworzy przestrzeń do wpisywania statusu statku  
 text=new Text("Status statku: ");  
 text.setY(pane.getHeight()-10);  
 text.setX(3); //P(setX, setY)  
 text.setFont(new Font(16));  
 text.setFill(Color.*BLACK*);  
 //Tutaj jest wstawiany status  
 *textStatus*=new Text();  
 *textStatus*.setY(pane.getHeight()-10);  
 *textStatus*.setX(3+text.getBoundsInLocal().getWidth()+3);  
 *textStatus*.setFont(new Font(16));  
 *textStatus*.setFill(Color.*BLACK*);  
  
 //Dodawanie tekstów na pane  
 pane.getChildren().add(text);  
 pane.getChildren().add(*textStatus*);  
  
 });  
 stage.show(); //pokaż okno ustawień - dopiero po jego zakończeniu zaczną generować się elementy  
 }  
}

### Kod zarządzający wartościami początkowymi (Properties.java)

package com.example.statek;  
//MAGAZYNUJE DANE POCZĄTKOWE, UDOSTĘPNIA JE DALEJ, PRZECHOWUJE METODY USTAWIANIA NOWYCH  
public class Properties {  
 private int N; // Pojemność statku  
 private int K; //Pojemność mostka  
 private int P; //Ilosc pasażerów  
 private int m\_delay; //Czas przejścia pzrzez mostek na statek  
 private int k\_delay; //Czas między rejsami (rzeczywisty: 1h)  
 private int r\_delay; //Czas rejsu  
  
 public Properties() {  
 this.N = 13;  
 this.K = 5;  
 this.P = 100;  
 this.m\_delay = 1000;  
 this.k\_delay = 10000;  
 this.r\_delay = 5000;  
 }  
  
 public int getN() {  
 return N;  
 }  
  
 public void setN(int n) {  
 N = n;  
 }  
  
 public int getK() {  
 return K;  
 }  
  
 public void setK(int k) {  
 K = k;  
 }  
  
 public int getP() {  
 return P;  
 }  
  
 public void setP(int p) {  
 P = p;  
 }  
  
 public int getM\_delay() {  
 return m\_delay;  
 }  
  
 public void setM\_delay(int m\_delay) {  
 this.m\_delay = m\_delay;  
 }  
  
 public int getK\_delay() {  
 return k\_delay;  
 }  
  
 public void setK\_delay(int k\_delay) {  
 this.k\_delay = k\_delay;  
 }  
  
 public int getR\_delay() {  
 return r\_delay;  
 }  
  
 public void setR\_delay(int r\_delay) {  
 this.r\_delay = r\_delay;  
 }  
}

### Kod zarządzający ustawieniami programu (PropertiesController.java)

package com.example.statek;  
  
import javafx.event.ActionEvent;  
import javafx.scene.Node;  
import javafx.scene.control.TextField;  
import javafx.stage.FileChooser;  
import javafx.stage.Stage;  
import org.w3c.dom.Document;  
import org.w3c.dom.Element;  
import org.xml.sax.SAXException;  
  
import javax.xml.parsers.DocumentBuilder;  
import javax.xml.parsers.DocumentBuilderFactory;  
import javax.xml.parsers.ParserConfigurationException;  
import javax.xml.transform.\*;  
import javax.xml.transform.dom.DOMSource;  
import javax.xml.transform.stream.StreamResult;  
import java.io.File;  
import java.io.FileNotFoundException;  
import java.io.FileOutputStream;  
import java.io.IOException;  
import java.util.Arrays;  
  
public class PropertiesController {  
 public static Properties *properties* = new Properties();  
 public TextField tf\_N, tf\_K, tf\_P, tf\_m\_delay, tf\_k\_delay, tf\_r\_delay;  
  
 public void initialize(){  
 fillTextFields();  
 }  
  
 public void saveToFile(ActionEvent actionEvent) throws ParserConfigurationException, FileNotFoundException, TransformerException {  
 save();  
  
 Stage stage = (Stage) ((Node)actionEvent.getSource()).getScene().getWindow();  
  
 FileChooser fileChooser = new FileChooser();  
 fileChooser.setTitle("Zapisz");  
 File file = fileChooser.showSaveDialog(stage);  
  
 if(file==null) return;  
  
 DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory.*newInstance*();  
 DocumentBuilder documentBuilder = documentBuilderFactory.newDocumentBuilder();  
  
 Document document = documentBuilder.newDocument();  
 Element el = document.createElement("shipSymulation");  
 document.appendChild(el);  
 Element N = document.createElement("N");  
 N.setTextContent(String.*valueOf*(*properties*.getN()));  
 el.appendChild(N);  
 Element K = document.createElement("K");  
 K.setTextContent(String.*valueOf*(*properties*.getK()));  
 el.appendChild(K);  
 Element P = document.createElement("P");  
 P.setTextContent(String.*valueOf*(*properties*.getP()));  
 el.appendChild(P);  
 Element m\_delay = document.createElement("m\_delay");  
 m\_delay.setTextContent(String.*valueOf*(*properties*.getM\_delay()));  
 el.appendChild(m\_delay);  
 Element k\_delay = document.createElement("k\_delay");  
 k\_delay.setTextContent(String.*valueOf*(*properties*.getK\_delay()));  
 el.appendChild(k\_delay);  
 Element r\_delay = document.createElement("r\_delay");  
 r\_delay.setTextContent(String.*valueOf*(*properties*.getR\_delay()));  
 el.appendChild(r\_delay);  
  
 FileOutputStream fileOutputStream = new FileOutputStream(file);  
  
 TransformerFactory transformerFactory = TransformerFactory.*newInstance*();  
 Transformer transformer = transformerFactory.newTransformer();  
 transformer.setOutputProperty(OutputKeys.*INDENT*, "yes");  
 transformer.setOutputProperty(OutputKeys.*OMIT\_XML\_DECLARATION*, "yes");  
 DOMSource source = new DOMSource(document);  
 StreamResult result = new StreamResult(fileOutputStream);  
 transformer.transform(source, result);  
  
 stage.close();  
 }  
  
 public void loadFromFile(ActionEvent actionEvent) throws ParserConfigurationException, IOException, SAXException {  
 Stage stage = (Stage) ((Node)actionEvent.getSource()).getScene().getWindow();  
  
 FileChooser fileChooser = new FileChooser();  
 fileChooser.setTitle("Zapisz");  
 File file = fileChooser.showOpenDialog(stage);  
  
 if(file==null) return;  
  
 DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory.*newInstance*();  
 DocumentBuilder documentBuilder = documentBuilderFactory.newDocumentBuilder();  
 Document document = documentBuilder.parse(file);  
  
 *properties*.setN(Integer.*parseInt*(document.getElementsByTagName("N").item(0).getTextContent()));  
 *properties*.setK(Integer.*parseInt*(document.getElementsByTagName("K").item(0).getTextContent()));  
 *properties*.setP(Integer.*parseInt*(document.getElementsByTagName("P").item(0).getTextContent()));  
 *properties*.setM\_delay(Integer.*parseInt*(document.getElementsByTagName("m\_delay").item(0).getTextContent()));  
 *properties*.setK\_delay(Integer.*parseInt*(document.getElementsByTagName("k\_delay").item(0).getTextContent()));  
 *properties*.setR\_delay(Integer.*parseInt*(document.getElementsByTagName("r\_delay").item(0).getTextContent()));  
  
 fillTextFields();  
 }  
  
 private void fillTextFields() {  
 tf\_N.setText(String.*valueOf*(*properties*.getN()));  
 tf\_K.setText(String.*valueOf*(*properties*.getK()));  
 tf\_P.setText(String.*valueOf*(*properties*.getP()));  
 tf\_m\_delay.setText(String.*valueOf*(*properties*.getM\_delay()));  
 tf\_k\_delay.setText(String.*valueOf*(*properties*.getK\_delay()));  
 tf\_r\_delay.setText(String.*valueOf*(*properties*.getR\_delay()));  
 }  
  
 public void set(ActionEvent actionEvent) {  
 Node node = (Node) actionEvent.getSource();  
 Stage stage = (Stage) node.getScene().getWindow();  
 save();  
 stage.close();  
 }  
  
 private void save(){  
 *properties*.setN(Integer.*parseInt*(tf\_N.getText()));  
 *properties*.setK(Integer.*parseInt*(tf\_K.getText()));  
 *properties*.setP(Integer.*parseInt*(tf\_P.getText()));  
 *properties*.setM\_delay(Integer.*parseInt*(tf\_m\_delay.getText()));  
 *properties*.setK\_delay(Integer.*parseInt*(tf\_k\_delay.getText()));  
 *properties*.setR\_delay(Integer.*parseInt*(tf\_r\_delay.getText()));  
 }  
}

### Kod powołujący procesy programu (start.java)

package com.example.statek;  
  
import javafx.scene.shape.Circle;  
import javafx.scene.text.Text;  
  
import java.util.ArrayList;  
import java.util.concurrent.Semaphore;  
  
public class start extends Thread{  
 Semaphore statek;  
 Semaphore[] mostek;  
 ArrayList<Integer> kolej;  
 ArrayList<Circle> pasazerList;  
 ArrayList<Text> pasazerTextList;  
  
 Semaphore c;  
  
 static ArrayList<Pasazer> *pasazers*;  
 int i;  
  
 static Kapitan *kapitan*;  
  
 //Kontruktor dla kapitana, do powoływania na nowo procedur powołania wątków pasażerów  
 public start(Semaphore statek, Semaphore[] mostek, ArrayList<Circle> pasazerList, ArrayList<Text> pasazerTextList, int i){  
 this.statek=statek;  
 this.mostek=mostek;  
 this.kolej=new ArrayList<>();  
 this.c=new Semaphore(1);  
 this.pasazerList=pasazerList;  
 this.pasazerTextList=pasazerTextList;  
 *pasazers*=new ArrayList<>();  
 this.i=i;  
 MainController.*interupt*=false;  
 }  
  
 public start (){ //Konstruktor tylko dla maincontrollera, żeby powował kapitana  
 pasazerList=new ArrayList<>();  
 pasazerTextList=new ArrayList<>();  
 c=new Semaphore(1);  
 statek=new Semaphore(PropertiesController.*properties*.getN());  
 mostek=new Semaphore[PropertiesController.*properties*.getK()];  
 for (int i=0; i<PropertiesController.*properties*.getK(); i++){  
 mostek[i]=new Semaphore(1);  
 }  
 kolej=new ArrayList<>();  
 *pasazers*=new ArrayList<>();  
 i=0;  
 *kapitan* = new Kapitan(statek, mostek, kolej, pasazerList, pasazerTextList);  
 *kapitan*.start();  
 MainController.*interupt*=false;  
 }  
  
 @Override  
 public void run() {  
 startAlg(); //Rozpocznij procesy współbierzne  
 }  
  
 public void startAlg() {  
 MainController.*textStatus*.setText("Przyjmuje pasażerów"); //Status statku na dole ekranu  
  
 Pasazer pasazer; //Nowy pasażer  
  
 while (i<PropertiesController.*properties*.getP()){  
 try {  
 boolean b=true;  
 while (b){  
 c.acquire();  
 if(kolej.size()<PropertiesController.*properties*.getK()+1) b=false; //Dla interrupta, żeby skończyć program  
 c.release();  
 }  
 c.acquire();  
 kolej.add(i); //Dodaj indeks do kolei, kolejność wszystkich pasażerów  
 c.release();  
  
 pasazer=new Pasazer(i, statek, mostek, kolej, c, pasazerList, pasazerTextList);  
 *pasazers*.add(pasazer); //Dodaj pasażera do listy pasażerów  
 pasazer.start(); //Startuj pasażera  
 } catch (InterruptedException e) {  
 if(MainController.*interupt*){ //Jeśli "stop symulacji"  
 for (int i=0; i<*pasazers*.size(); i++){  
 *pasazers*.get(i).interrupt(); //Zakończ działanie pasażerów  
 }  
 }  
 break;  
 }  
 i++;  
 }  
 MainController.*starts*.remove(this); //Kończy proces tego startu (każdy start odpowiada tylko za powoływanie pasażerów)  
 }  
}

### Kod wykonujący zadania kapitana

package com.example.statek;  
  
import javafx.application.Platform;  
import javafx.scene.shape.Circle;  
import javafx.scene.text.Text;  
  
import java.util.ArrayList;  
import java.util.concurrent.Semaphore;  
  
public class Kapitan extends Thread {  
 ArrayList<Circle> pasazerList; //Obiekty (kółka) pasażerów  
 ArrayList<Text> pasazerTextList; //Numery pasażerów  
  
 Semaphore statek;  
 Semaphore[] mostek;  
 ArrayList<Integer> kolej; //Kolejnosc na mostku  
  
 public Kapitan(Semaphore statek, Semaphore[] mostek, ArrayList<Integer> kolej, ArrayList<Circle> pasazerList, ArrayList<Text> pasazerTextList) {  
 this.statek = statek;  
 this.mostek = mostek;  
 this.kolej=kolej;  
 this.pasazerList=pasazerList;  
 this.pasazerTextList=pasazerTextList;  
 }  
  
 public void run() {  
 Runnable runnable = () -> {  
 try {  
 Thread.*sleep*(PropertiesController.*properties*.getK\_delay()-1000); //Oczekiwanie kapitana na rejs  
 } catch (InterruptedException e) {  
 return;  
 }  
 };  
 Thread thread = null;  
  
  
 while (true){  
 try{  
 thread=new Thread(runnable);  
 thread.start(); //Czeka  
 while (statek.availablePermits()!=0 && thread.isAlive() && !MainController.*interupt*); //Jak się zapełni statek to robi rejs  
 if(MainController.*interupt*) this.interrupt(); //Jeśli zakończ program, to przerwij działanie wątku  
 if(thread.isAlive()) Thread.*sleep*(1000); //Czas oczekiwania, chciałem, żeby było widać, że czekający schodzą z mostka  
  
 MainController.*textStatus*.setText("Odpływa"); //Status statku na dole okna  
  
 int newI;  
 if(!start.*pasazers*.isEmpty()) newI = start.*pasazers*.get(0).id; //Pobiera pasażera pierwszego na mostku, po rejsie jako pierwszy wejdzie na statek  
 else newI=-1; //Jeżeli żaden pasażer nie czeka na mostku (nie będzie następnego rejsu)  
  
 for (int i=0; i<kolej.size(); i++){  
 kolej.remove(0); //Czyści kolejkę, po rejsie będzie zapełniana na nowo  
 }  
 Platform.*runLater*(()->{  
 for (int i=0; i<start.*pasazers*.size(); i++){ //Usuwa pasażerów z mostka  
 int finalI = i;  
 MainController.*paneS*.getChildren().remove(start.*pasazers*.get(finalI).circle); //Usuwa koło z mostka  
 MainController.*paneS*.getChildren().remove(start.*pasazers*.get(finalI).text); //Usuwa numer z mostka  
 }  
 for(int i=0; i<start.*pasazers*.size(); i++) start.*pasazers*.get(i).interrupt(); //Zakończ proces pasażera z mostka z listy start  
 for (int i=0; i<start.*pasazers*.size(); i++) start.*pasazers*.remove(0); //Usuń pasażera z mostka z listy start  
 });  
  
  
 for (int i=0; i<MainController.*starts*.size(); i++){  
 MainController.*starts*.get(i).interrupt(); //usuń wszystkie aktualne starty  
 }  
 for (int i=0; i<MainController.*starts*.size(); i++){  
 MainController.*starts*.remove(0); //Czyści listę startów (bo tak to by były tylko interrupted)  
 }  
  
 Thread.*sleep*(PropertiesController.*properties*.getR\_delay()); //Trwa rejs  
 Platform.*runLater*(()->{ //Musi poczekać na usunięcie z paneS  
 for (int i=0; i<pasazerList.size(); i++){  
 MainController.*paneS*.getChildren().remove(pasazerList.get(i)); //Usuwanie kółek graficznie  
 MainController.*paneS*.getChildren().remove(pasazerTextList.get(i)); //Usuwanie numerów graficznie  
 }  
 pasazerList.clear(); //Usuwanie kółek pasażerów z listy  
 pasazerTextList.clear(); //Usuwanie numerów pasażerów z listy  
 });  
 statek.release(PropertiesController.*properties*.getN()-statek.availablePermits()); //Koniec rejsu, semafory statku puste, zwolnij (wszystkie-wolne) - dla sytuacji, kiedy nie cały statek jest zajęty  
 for (int i=0; i<PropertiesController.*properties*.getK(); i++){  
 if(mostek[i].availablePermits()==0) mostek[i].release(); //Zwalnia wszystkie semafory mostka  
 }  
  
 if(newI!=-1){ //Jeżeli statek wyruszył w rejs, ale zostali jeszcze pasażerowie  
 start start = new start(statek, mostek, pasazerList, pasazerTextList, newI); //Powołuj dalej  
 start.start();  
 MainController.*starts*.add(start);  
 }else{  
 break; //Zakończ działanie kapitana - idzie do 99 linii - koniec symulacji  
 }  
 } catch (InterruptedException e) {  
 thread.interrupt();  
 break;  
 }  
 }  
  
 MainController.*textStatus*.setText("Koniec symulacji");  
 }  
}

### Kod wykonujący zadania pasażera

package com.example.statek;  
  
import javafx.application.Platform;  
import javafx.scene.paint.Color;  
import javafx.scene.shape.Circle;  
import javafx.scene.text.Font;  
import javafx.scene.text.Text;  
  
import java.util.ArrayList;  
import java.util.concurrent.Semaphore;  
  
public class Pasazer extends Thread {  
 Circle circle;  
 Text text;  
 ArrayList<Circle> pasazerList;  
 ArrayList<Text> pasazerTextList;  
  
 Semaphore statek;  
 Semaphore[] mostek;  
 ArrayList<Integer> kolej;  
 int id;  
  
 Semaphore c; //Semafor chron, kontroluje dostęp do kolei (żeby 2 procesy nie miały dostępu na raz)  
  
 public Pasazer(int id, Semaphore statek, Semaphore[] mostek, ArrayList<Integer> kolej, Semaphore c, ArrayList<Circle> pasazerList, ArrayList<Text> pasazerTextList) {  
 super(String.*valueOf*(id));  
 this.id=id;  
 this.statek = statek;  
 this.mostek = mostek;  
 this.kolej=kolej;  
 this.c=c;  
 this.pasazerList=pasazerList;  
 this.pasazerTextList=pasazerTextList;  
 }  
  
 public void run(){  
 boolean b=true;  
  
 try{  
 while(b){  
 c.acquire();  
 if(kolej.indexOf(id)==0) b=false; //Dla interrupta, żeby skończyć program  
 c.release();  
 }  
  
 mostek[PropertiesController.*properties*.getK()-1].acquire(); //Zajmij semafor na mostku dla tego pasażera  
 circle=new Circle(  
 MainController.*mostekConteiners*.get(PropertiesController.*properties*.getK()-1).getX()+MainController.*bokRectagle*/2,  
 MainController.*mostekConteiners*.get(PropertiesController.*properties*.getK()-1).getY()+MainController.*bokRectagle*/2,  
 MainController.*bokRectagle*/2-5); //Kółko dla pasażera  
 circle.setFill(Color.*BLACK*);  
 circle.setAccessibleText(String.*valueOf*(id));  
 //Numer pasażera:  
 text=new Text(String.*valueOf*(id));  
 text.setX(circle.getCenterX()-3);  
 text.setY(circle.getCenterY()+3);  
 text.setFont(new Font(15));  
 text.setFill(Color.*WHITE*);  
  
 Platform.*runLater*(()->{ //Dodaj na pane, pasażer zajmuje pierwsze miejsce mostka  
 MainController.*paneS*.getChildren().add(circle);  
 MainController.*paneS*.getChildren().add(text);  
 });  
 Thread.*sleep*(PropertiesController.*properties*.getM\_delay()/PropertiesController.*properties*.getK()); //Czas przejścia przez punkt mostka (nr1)  
 for (int i=1; i<PropertiesController.*properties*.getK(); i++){  
 mostek[PropertiesController.*properties*.getK()-1-i].acquire(); //Zajmij następny punkt mostka  
 circle.setCenterX(MainController.*mostekConteiners*.get(PropertiesController.*properties*.getK()-1-i).getX()+MainController.*bokRectagle*/2); //Zmień położenie kółka na następny punkt mostka  
 if(i==1){ //Jeśli wszedł na drugie miejsce mostka  
 c.acquire();  
 kolej.remove((Integer) id); //Usuń pasażera z kolejki  
 c.release();  
 }  
 text.setX(circle.getCenterX()-3);  
 mostek[PropertiesController.*properties*.getK()-1-(i-1)].release(); //Zwolnij poprzedni punkt mostka  
 Thread.*sleep*(PropertiesController.*properties*.getM\_delay()/PropertiesController.*properties*.getK()); //Czas przejścia przez punkt mostka  
 }  
  
 statek.acquire(); //Zajmuje miejsce na statku  
 c.acquire();  
 pasazerList.add(circle); //Dodaj obiekt do listy  
 pasazerTextList.add(text);  
 c.release(); //Poniżej pasażer na odpowiednie miejsce statku  
 circle.setCenterX(MainController.*statekConteiners*.get(pasazerList.indexOf(circle)).getX()+MainController.*bokRectagle*/2);  
 circle.setCenterY(MainController.*statekConteiners*.get(pasazerList.indexOf(circle)).getY()+MainController.*bokRectagle*/2);  
 text.setX(MainController.*statekConteiners*.get(pasazerList.indexOf(circle)).getX()+MainController.*bokRectagle*/2-3);  
 text.setY(MainController.*statekConteiners*.get(pasazerList.indexOf(circle)).getY()+MainController.*bokRectagle*/2+3);  
 mostek[0].release(); //Zszedł z mostka, wolne miejsce  
 start.*pasazers*.remove(Pasazer.this); //Proces pasażera się kończy po wejściu na mostek  
 } catch (InterruptedException e) {  
 return;  
 }  
 }  
}