Картина, която съдържа текст, графична колекция

Описанието е генерирано автоматичноТЕХНИЧЕСКИ УНИВЕРСИТЕТ – ВАРНА

Факултет по изчислителна техника и автоматизация

Катедра „КНТ“

**КУРСОВ ПРОЕКТ**

по дисциплината „ООП – 2ра част”

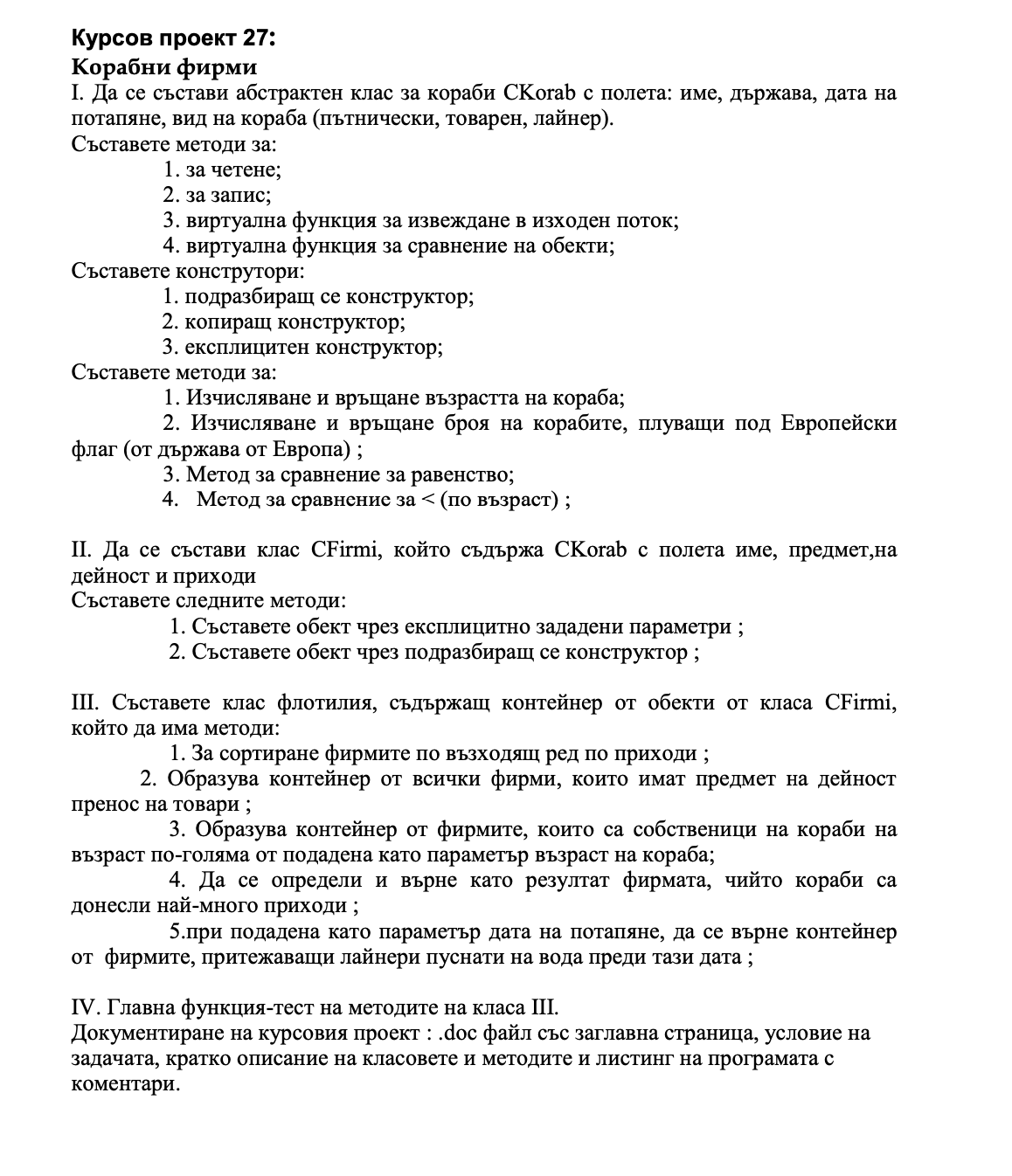
на тема: „Програма за обработкa кораби,фирми и пристанище(флот) ”

Вариант 27

|  |  |
| --- | --- |
| Изготвил: Ивайло Илиев | Проверил: |
| Специалност: КСТ |  |
| Група: 3б |  |
| Факултетен номер: 22621518  Курс: 3ти |  |

2024

**I. Задание**



**II. Кратко описание на програмата**

**Предназначение: Програмата се използва за:**

* **Управление на кораби и компании в рамките на дадено пристанище.**
* **Извършване на справки като сортиране по приходи, филтриране по дейност, извеждане на компании с най-стари кораби и други.**

**Данни:**

* **Ship: Име, държава, дата на пускане, тип (пътнически, товарен и др.).**
* **Company: Име, дейност, приходи и списък от кораби.**
* **Port: Съдържа множество компании.**

**Вход и изход на данни:**

* **Зареждане на данни от файлове ships.txt, Companies.txt.**
* **Запис на справки в изходни файлове.**

**III. Обща структура на програмата**

**1. Клас Ship**

**Описание:** Съхранява информация за кораби. **Член-променливи:**

* String name - Име на кораба.
* String country - Държава на кораба.
* String dateOfLaunch - Дата на пускане.
* String type - Тип на кораба.

**Методи:**

* Конструктори (по подразбиране, с параметри, копиращ).
* int calculateAge() - Изчислява възрастта на кораба.
* Статичен метод loadFromFile(String filename) - Зарежда кораби от файл.

**2. Клас Company**

**Описание:** Съхранява информация за компании и списък от кораби. **Член-променливи:**

* String name - Име на компанията.
* String subjectOfActivity - Дейност на компанията.
* double income - Приходи.
* ArrayList<Ship> ships - Списък от кораби.

**Методи:**

* void addShip(Ship ship) - Добавя кораб.
* int countShipsByAge(int age) - Брой кораби по възраст.
* Методи за запис и зареждане във/от файл.

**3. Клас Port**

**Описание:** Управлява списък от компании. **Член-променливи:**

* ArrayList<Company> companies - Списък от компании.

**Методи:**

* void addCompany(Company company) - Добавя компания.
* List<Company> sortCompaniesByIncome() - Сортира компаниите по приходи.
* List<Company> getCompaniesByActivity(String activity) - Филтрира компаниите по дейност.
* Company getCompanyWithHighestIncome() - Връща компанията с най-висок доход.
* List<Company> getCompaniesWithOldShips(int age) - Компании със стари кораби.
* void saveToFile(String filename) - Запис на данни във файл.

**4. Клас ConsoleApp**

**Описание:** Управлява работата с потребителя чрез конзола. **Основни опции:**

1. Извеждане на всички компании.
2. Сортиране на компаниите по приходи.
3. Филтриране по дейност.
4. Компании с кораби, по-стари от дадена възраст.
5. Компания с най-висок доход.
6. Компании с кораби, пуснати преди определена дата.
7. Запис на данни във файл.
8. Зареждане на данни от файл.

**5. Клас SwingApp**

**Описание:** Графичен интерфейс за управление на пристанището.

**Функционалности:**

* Списък на компаниите и корабите в таблица.
* Сортиране и филтриране с бутони.
* Генериране на справки и извеждане в текстови полета.

**IV. Тестови резултати**

**Нужните файлове за работа с програмата:**

**A screenshot of a computer

Description automatically generated**A screenshot of a computer

Description automatically generated

Примерна работа на програмата:

1. SWING

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated**

1. **ConsoleApp**

**A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated**

**IV.Source код**

import java.io.\*;  
import java.util.ArrayList;  
import java.util.Calendar;  
import java.util.List;  
import java.util.Set;  
  
public class Ship implements FileWork {  
 private String name;  
 private String country;  
 private String dateOfLaunch;  
 private String type; // Passenger, Cargo, or Liner  
 public Ship(){  
 this.name="";  
 this.country="";  
 this.dateOfLaunch="";  
 this.type="";  
 }  
 public Ship(String name, String country, String dateOfLaunch, String type) {  
 this.name = name;  
 this.country = country;  
 this.dateOfLaunch = dateOfLaunch;  
 this.type = type;  
 }  
 public Ship(Ship other) {  
 this.name = other.name;  
 this.country = other.country;  
 this.dateOfLaunch = other.dateOfLaunch;  
 this.type = other.type;  
 }  
 public String getType() {  
 return type;  
 }  
 public void setType(String type) {  
 this.type = type;  
 }  
 public String getDateOfLaunch() {  
 return dateOfLaunch;  
 }  
 public void setDateOfLaunch(String dateOfLaunch) {  
 this.dateOfLaunch = dateOfLaunch;  
 }  
 public String getCountry() {  
 return country;  
 }  
 public void setCountry(String country) {  
 this.country = country;  
 }  
 public String getName() {  
 return name;  
 }  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 @Override  
 public void saveToFile(String filename) throws IOException {  
 try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename, true))) {  
 writer.write("\tName: " + name + " Country: " + country + "\n" +  
 "\tDate Of Launch: " + dateOfLaunch + " Type: " + type + "\n");  
 writer.newLine();  
 }  
 }  
  
 static public ArrayList<Ship> loadFromFile(String filename) throws IOException {  
 ArrayList<Ship> input=new ArrayList<>();  
 try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {  
 String line="";  
 while ((line = reader.readLine())!=null){  
 if (line != null) {  
 String[] parts = line.split("\\s\*\\|\\s\*");  
 if (parts.length == 4) {  
 input.add(new Ship(parts[0],parts[1],parts[2],parts[3]));  
 }  
 }  
 }  
 }  
 return input;  
 }  
  
 @Override  
 public String toString() {  
 return "\tName: " + name + " Country: " + country + "\n" +  
 "\tDate Of Launch: " + dateOfLaunch + " Type: " + type + "\n";  
 }  
  
 public int calculateAge() {  
 int currentYear = Calendar.*getInstance*().get(Calendar.*YEAR*);  
 int launchYear = Integer.*parseInt*(dateOfLaunch.split("-")[2]);  
 return currentYear - launchYear;  
 }  
 // Static method to count ships under European flags  
 public static int countEuropeanShips(List<Ship> ships, Set<String> europeanCountries) {  
 int count = 0;  
 for (Ship ship : ships) {  
 if (europeanCountries.contains(ship.getCountry().toLowerCase())) {  
 count++;  
 }  
 }  
 return count;  
 }  
  
 public boolean equals(Ship other) {  
 return this.name.equalsIgnoreCase(other.name) &&  
 this.type.equalsIgnoreCase(other.type);  
 }  
 public boolean isOlderThan(Ship other) {  
 return this.calculateAge() < other.calculateAge();  
 }  
}

import java.io.\*;  
import java.util.ArrayList;  
  
public class Company implements FileWork {  
 private String name;  
 private String subjectOfActivity;  
 private double income;  
 private ArrayList<Ship> ships;  
  
 public Company() {  
 this.name = "";  
 this.subjectOfActivity = "";  
 this.income = 0;  
 }  
 public Company(String name, String subjectOfActivity, double income) {  
 this.name = name;  
 this.subjectOfActivity = subjectOfActivity;  
 this.income = income;  
 this.ships = new ArrayList<>();  
 }  
  
 public void addShip(Ship ship) {  
 ships.add(ship);  
 }  
  
 public ArrayList<Ship> getShips() {  
 return ships;  
 }  
 public void setShips(ArrayList<Ship> sh) {  
 this.ships=sh;  
 }  
 public String getName() {  
 return name;  
 }  
 public void setName(String name) {  
 this.name = name;  
 }  
 public String getSubjectOfActivity() {  
 return subjectOfActivity;  
 }  
 public void setSubjectOfActivity(String subjectOfActivity) {  
 this.subjectOfActivity = subjectOfActivity;  
 }  
 public double getIncome() {  
 return income;  
 }  
 public void setIncome(double income) {  
 this.income = income;  
 }  
  
 public int countShipsByAge(int age) {  
 return (int) ships.stream().filter(ship -> ship.calculateAge() > age).count();  
 }  
  
 @Override  
 public void saveToFile(String filename) throws IOException {  
 try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {  
 String help="";  
 int count=0;  
 for(Ship i : ships)  
 {  
 count++;  
 help+=" Ship "+count+"\n";  
 help+=i.toString();  
  
 };  
 writer.write("Name: " + name + " Subject Of Activity: " + subjectOfActivity + " Income: " + income  
 +"\n"+help+"\n");  
 writer.newLine();  
 }  
 }  
  
 static public ArrayList<Company> loadFromFile(String filename) throws IOException {  
 ArrayList<Company> out=new ArrayList<>();  
 try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {  
 String line="";  
 while ((line = reader.readLine())!=null) {  
 if (line != null) {  
 String[] parts = line.split(", ");  
 out.add(new Company(parts[0], parts[1], Double.*parseDouble*(parts[2])));  
 }  
 }  
 }  
 return out;  
 }  
 @Override  
 public String toString() {  
 String help="";  
 int count=0;  
 for(Ship i : ships)  
 {  
 count++;  
 help+=" Ship "+count+"\n";  
 help+=i.toString();  
  
 };  
 return "Name: " + name + " Subject Of Activity: " + subjectOfActivity + " Income: " + income  
 +"\n"+help+"\n";  
  
 }  
}

import java.util.\*;  
import java.util.stream.Collectors;  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.io.\*;  
  
public class Port implements FileWork {  
 private ArrayList<Company> companies;  
  
 public Port() {  
 this.companies = new ArrayList<>();  
 }  
  
 public void addCompany(Company company) {  
 companies.add(company);  
 }  
  
 // 1. Sort Companies by Income  
 public List<Company> sortCompaniesByIncome() {  
 return companies.stream()  
 .sorted(Comparator.*comparingDouble*(Company::getIncome))  
 .collect(Collectors.*toList*());  
 }  
  
 // 2. Get Companies with Subject of Activity as Cargo Transfer  
 public List<Company> getCompaniesByActivity(String activity) {  
 return companies.stream()  
 .filter(company -> company.getSubjectOfActivity().equalsIgnoreCase(activity))  
 .collect(Collectors.*toList*());  
 }  
  
 // 3. Get Companies with Ships Older than a Given Age  
 public List<Company> getCompaniesWithOldShips(int shipAge) {  
 return companies.stream()  
 .filter(company -> company.getShips().stream()  
 .anyMatch(ship -> ship.calculateAge() > shipAge))  
 .collect(Collectors.*toList*());  
 }  
  
 // 4. Get the Company with the Highest Income  
 public Company getCompanyWithHighestIncome() {  
 return companies.stream()  
 .max(Comparator.*comparingDouble*(Company::getIncome))  
 .orElse(null);  
 }  
  
 // 5. Get Companies with Liners Launched Before a Given Date  
 public List<Company> getCompaniesWithLinersBeforeDate(String date) {  
 SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");  
 Date targetDate;  
 try {  
 targetDate = sdf.parse(date);  
 } catch (ParseException e) {  
 System.*out*.println("Invalid date format. Please use dd-MM-yyyy.");  
 return Collections.*emptyList*();  
 }  
  
 return companies.stream()  
 .filter(company -> company.getShips().stream()  
 .anyMatch(ship -> {  
 try {  
 return ship.getType().equalsIgnoreCase("liner") &&  
 sdf.parse(ship.getDateOfLaunch()).before(targetDate);  
 } catch (ParseException e) {  
 return false;  
 }  
 }))  
 .collect(Collectors.*toList*());  
 }  
  
 @Override  
 public void saveToFile(String filename) {  
 try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {  
 writer.write(this.toString());  
 writer.newLine();  
 } catch (IOException e) {  
 System.*out*.println("Error saving to file: " + e.getMessage());  
 }  
 }  
  
  
 @Override  
 public String toString() {  
 String out="";int count=0;  
 for(Company y: companies)  
 {  
 count++;  
 out+="Company "+count+"\n";  
 out+=y.toString();  
 }  
 return out;  
 }  
  
 public ArrayList<Company> getCompanies() {  
 return companies;  
 }  
 public Map<String, Integer> getShipCountByAge(int age) {  
 Map<String, Integer> result = new HashMap<>();  
 for (Company company : companies) {  
 result.put(company.getName(), company.countShipsByAge(age));  
 }  
 return result;  
 }  
}

import java.io.IOException;

import java.util.ArrayList;

public interface FileWork {

void saveToFile(String filename) throws IOException;

}

SWING

t javax.swing.\*;

import javax.swing.table.DefaultTableModel;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import java.util.Map;

import java.util.Random;

public class SwingApp {

private Port port;

private JFrame frame;

private JTable table;

private JTextArea textArea;

public SwingApp() {

port = new Port();

fillWithData();

initUI();

}

private void fillWithData() {

try {

Random random = new Random();

ArrayList<Ship> ships = Ship.*loadFromFile*("ships.txt");

ArrayList<Company> companies = Company.*loadFromFile*("Companies.txt");

for (Company i : companies) {

for (int j = 0; j < 25; j++) { // Each company gets 25 random ships

i.addShip(ships.get(random.nextInt(ships.size())));

}

port.addCompany(i);

}

} catch (IOException e) {

throw new RuntimeException(e);

}

}

private void populatePortWithRandomData() {

Random random = new Random();

String[] companyNames = {"Alpha Shipping", "Beta Marine", "Gamma Transport", "Delta Cargo", "Epsilon Lines",

"Zeta Vessels", "Eta Seaways", "Theta Freight", "Iota Carriers", "Kappa Ships",

"Lambda Cargo", "Mu Logistics", "Nu Transit", "Xi Maritime", "Omicron Transport",

"Pi Shippers", "Rho Vessels", "Sigma Lines", "Tau Maritime", "Upsilon Freight"};

String[] subjectsOfActivity = {"Cargo Transfer", "Passenger Transport", "Freight Logistics", "Maritime Services", "Oil Transport"};

String[] countries = {"Bulgaria", "Germany", "France", "Italy", "Spain", "Netherlands", "Greece", "Sweden", "Norway", "Denmark"};

String[] shipTypes = {"Passenger", "Cargo", "Liner"};

for (String companyName : companyNames) {

String activity = subjectsOfActivity[random.nextInt(subjectsOfActivity.length)];

double income = 100000 + random.nextDouble() \* 900000; // Random income between 100,000 and 1,000,000

Company company = new Company(companyName, activity, income);

for (int i = 0; i < 25; i++) { // Each company gets 25 random ships

String shipName = "Ship-" + random.nextInt(1000);

String country = countries[random.nextInt(countries.length)];

String dateOfLaunch = (random.nextInt(((2024 - 1960) + 1) + 1960)) + "-01-01"; // Random year from 1960 to 2024

String type = shipTypes[random.nextInt(shipTypes.length)];

company.addShip(new Ship(shipName, country, dateOfLaunch, type));

}

port.addCompany(company);

}

}

private void initUI() {

frame = new JFrame("Port Management System");

frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

frame.setSize(800, 600);

// Main Panel Layout

JPanel panel = new JPanel(new BorderLayout());

panel.setBackground(new Color(0, 128, 0)); // Set panel background to green

// Buttons

JPanel buttonPanel = new JPanel(new GridLayout(4, 3));

buttonPanel.setBackground(new Color(0, 128, 0)); // Set button panel background to green

JButton listCompaniesBtn = new JButton("List All Companies");

JButton sortCompaniesBtn = new JButton("Sort Companies by Income");

JButton filterCompaniesBtn = new JButton("Filter Companies by Activity");

JButton oldShipsBtn = new JButton("Companies with Old Ships");

JButton highestIncomeBtn = new JButton("Company with Highest Income");

JButton linersBeforeDateBtn = new JButton("Liners Before Date");

JButton saveToFileBtn = new JButton("Save to File");

JButton loadFromFileBtn = new JButton("Load from File");

JButton shipCountBtn = new JButton("Ship Count by Age");

JButton portToStringBtn = new JButton("Port Overview");

buttonPanel.add(listCompaniesBtn);

buttonPanel.add(sortCompaniesBtn);

buttonPanel.add(filterCompaniesBtn);

buttonPanel.add(oldShipsBtn);

buttonPanel.add(highestIncomeBtn);

buttonPanel.add(linersBeforeDateBtn);

buttonPanel.add(saveToFileBtn);

buttonPanel.add(loadFromFileBtn);

buttonPanel.add(shipCountBtn);

buttonPanel.add(portToStringBtn);

// Table and Text Area

table = new JTable();

JScrollPane tableScrollPane = new JScrollPane(table);

textArea = new JTextArea(5, 40);

JScrollPane textScrollPane = new JScrollPane(textArea);

textScrollPane.setPreferredSize(new Dimension(800, 200)); // Задайте желаната широчина и височина

// Add to Panel

panel.add(buttonPanel, BorderLayout.*NORTH*);

panel.add(tableScrollPane, BorderLayout.*CENTER*);

panel.add(textScrollPane, BorderLayout.*SOUTH*);

frame.add(panel);

// Add Action Listeners

listCompaniesBtn.addActionListener(e -> listAllCompanies());

sortCompaniesBtn.addActionListener(e -> sortCompaniesByIncome());

filterCompaniesBtn.addActionListener(e -> filterCompaniesByActivity());

oldShipsBtn.addActionListener(e -> companiesWithOldShips());

highestIncomeBtn.addActionListener(e -> companyWithHighestIncome());

linersBeforeDateBtn.addActionListener(e -> companiesWithLinersBeforeDate());

saveToFileBtn.addActionListener(e -> saveToFile());

loadFromFileBtn.addActionListener(e -> loadFromFile());

shipCountBtn.addActionListener(e -> shipCountByAge());

portToStringBtn.addActionListener(e -> showPortOverview());

frame.setVisible(true);

}

private void listAllCompanies() {

updateTable(port.getCompanies(), "List of All Companies");

}

private void sortCompaniesByIncome() {

updateTable(port.sortCompaniesByIncome(), "Sorted Companies by Income");

}

private void filterCompaniesByActivity() {

String activity = JOptionPane.*showInputDialog*(frame, "Enter Activity to Filter:");

updateTable(port.getCompaniesByActivity(activity), "Filtered Companies by Activity: " + activity);

}

private void companiesWithOldShips() {

String ageInput = JOptionPane.*showInputDialog*(frame, "Enter Age to Filter Old Ships:");

int age = Integer.*parseInt*(ageInput);

updateTable(port.getCompaniesWithOldShips(age), "Companies with Ships Older than " + age);

}

private void companyWithHighestIncome() {

Company company = port.getCompanyWithHighestIncome();

textArea.setText("Company with the Highest Income:\n" + company);

}

private void companiesWithLinersBeforeDate() {

String date = JOptionPane.*showInputDialog*(frame, "Enter Date (yyyy-MM-dd):");

updateTable(port.getCompaniesWithLinersBeforeDate(date), "Companies with Liners Before " + date);

}

private void shipCountByAge() {

String ageInput = JOptionPane.*showInputDialog*(frame, "Enter Age of Ships:");

int age = Integer.*parseInt*(ageInput);

Map<String, Integer> shipCounts = port.getShipCountByAge(age);

StringBuilder result = new StringBuilder("Ship Count by Age:\n");

shipCounts.forEach((company, count) -> result.append("Company: ").append(company).append(", Ships: ").append(count).append("\n"));

textArea.setText(result.toString());

}

private void saveToFile() {

String filename = JOptionPane.*showInputDialog*(frame, "Enter filename to save:");

port.saveToFile(filename);

JOptionPane.*showMessageDialog*(frame, "Data saved to file: " + filename);

}

private void loadFromFile() {

populatePortWithRandomData();

JOptionPane.*showMessageDialog*(frame, "Random Data Loaded Successfully!");

}

private void showPortOverview() {

textArea.setText(port.toString());

}

private void updateTable(List<Company> companies, String title) {

String[] columnNames = {"Name", "Activity", "Income"};

DefaultTableModel model = new DefaultTableModel(columnNames, 0);

for (Company company : companies) {

model.addRow(new Object[]{company.getName(), company.getSubjectOfActivity(), company.getIncome()});

}

table.setModel(model);

textArea.setText(title);

}

public static void main(String[] args) {

SwingUtilities.*invokeLater*(SwingApp::new);

}

}

Console

import java.util.\*;

import java.io.\*;

public class ConsoleApp {

private Port port;

private Scanner scanner;

public ConsoleApp() {

port = new Port();

scanner = new Scanner(System.*in*);

fillWithData();

//populatePortWithRandomData();

}

private void fillWithData()

{

try {

Random random = new Random();

ArrayList<Ship> ships=Ship.*loadFromFile*("ships.txt");

ArrayList<Company> companies=Company.*loadFromFile*("Companies.txt");

for (Company i : companies) {

for (int j = 0; j < 25; j++) { // Each company gets 25 random ships

i.addShip(ships.get(random.nextInt(500)));

}

port.addCompany(i);

}

} catch (IOException e) {

throw new RuntimeException(e);

}

}

private void populatePortWithRandomData() {

Random random = new Random();

String[] companyNames = {"Alpha Shipping", "Beta Marine", "Gamma Transport", "Delta Cargo", "Epsilon Lines",

"Zeta Vessels", "Eta Seaways", "Theta Freight", "Iota Carriers", "Kappa Ships",

"Lambda Cargo", "Mu Logistics", "Nu Transit", "Xi Maritime", "Omicron Transport",

"Pi Shippers", "Rho Vessels", "Sigma Lines", "Tau Maritime", "Upsilon Freight"};

String[] subjectsOfActivity = {"Cargo Transfer", "Passenger Transport", "Freight Logistics", "Maritime Services", "Oil Transport"};

String[] countries = {"Bulgaria", "Germany", "France", "Italy", "Spain", "Netherlands", "Greece", "Sweden", "Norway", "Denmark"};

String[] shipTypes = {"Passenger", "Cargo", "Liner"};

for (String companyName : companyNames) {

String activity = subjectsOfActivity[random.nextInt(subjectsOfActivity.length)];

double income = 100000 + random.nextDouble() \* 900000; // Random income between 100,000 and 1,000,000

Company company = new Company(companyName, activity, income);

for (int i = 0; i < 25; i++) { // Each company gets 25 random ships

String shipName = "Ship-" + random.nextInt(1000);

String country = countries[random.nextInt(countries.length)];

String dateOfLaunch = (random.nextInt(((2024 - 1960) + 1) + 1960)) + "-01-01"; // Random year from 1960 to 2024

String type = shipTypes[random.nextInt(shipTypes.length)];

company.addShip(new Ship(shipName, country, dateOfLaunch, type));

}

port.addCompany(company);

}

}

public void start() {

while (true) {

System.*out*.println("\nChoose an option:");

System.*out*.println("1. List All Companies");

System.*out*.println("2. Sort Companies by Income");

System.*out*.println("3. Filter Companies by Activity");

System.*out*.println("4. Companies with Old Ships");

System.*out*.println("5. Company with Highest Income");

System.*out*.println("6. Companies with Liners Before a Given Date");

System.*out*.println("7. Save to File");

System.*out*.println("8. Load from File");

System.*out*.println("9. Exit");

System.*out*.println("10. Ship Count by Age for Each Company");

System.*out*.println("Choose ONE: ");

int choice = Integer.*parseInt*(scanner.nextLine());

switch (choice) {

case 1 -> listAllCompanies();

case 2 -> sortCompaniesByIncome();

case 3 -> filterCompaniesByActivity();

case 4 -> companiesWithOldShips();

case 5 -> companyWithHighestIncome();

case 6 -> companiesWithLinersBeforeDate();

case 7 -> saveToFile();

case 8 -> loadFromFile();

case 9 -> {

System.*out*.println("Exiting...");

return;

}

case 10 -> shipCountByAge();

default -> System.*out*.println("Invalid option. Please try again.");

}

}

}

private void listAllCompanies() {

System.*out*.println("\nAll Companies:");

System.*out*.println(port.toString());

//port.getCompaniesByActivity("").forEach(System.out::println);

}

private void sortCompaniesByIncome() {

System.*out*.println("\nSorted Companies by Income:");

port.sortCompaniesByIncome().forEach(System.*out*::println);

}

private void filterCompaniesByActivity() {

System.*out*.print("Enter Activity to Filter: ");

String activity = scanner.nextLine();

System.*out*.println("\nFiltered Companies:");

port.getCompaniesByActivity(activity).forEach(System.*out*::println);

}

private void companiesWithOldShips() {

System.*out*.print("Enter Age to Filter Companies by Old Ships: ");

int age = Integer.*parseInt*(scanner.nextLine());

System.*out*.println("\nCompanies with Ships Older than " + age + ":");

port.getCompaniesWithOldShips(age).forEach(System.*out*::println);

}

private void companyWithHighestIncome() {

Company company = port.getCompanyWithHighestIncome();

if (company != null) {

System.*out*.println("\nCompany with the Highest Income:");

System.*out*.println(company);

} else {

System.*out*.println("No companies available.");

}

}

private void companiesWithLinersBeforeDate() {

System.*out*.print("Enter Date (dd-MM-yyyy): ");

String date = scanner.nextLine();

List<Company> companies = port.getCompaniesWithLinersBeforeDate(date);

if (!companies.isEmpty()) {

System.*out*.println("\nCompanies with Liners Before " + date + ":");

companies.forEach(System.*out*::println);

} else {

System.*out*.println("No companies found with liners before the specified date.");

}

}

private void shipCountByAge() {

System.*out*.print("Enter Age of Ships to Filter: ");

int age = Integer.*parseInt*(scanner.nextLine());

Map<String, Integer> result = port.getShipCountByAge(age);

System.*out*.println("\nShip Count by Age for Each Company (Older than " + age + " years):");

result.forEach((company, count) ->

System.*out*.println("Company: " + company + ", Ships: " + count));

}

private void saveToFile() {

System.*out*.print("Enter filename to save: ");

String filename = scanner.nextLine();

port.saveToFile(filename);

System.*out*.println("Data saved to file: " + filename);

}

private void loadFromFile() {

populatePortWithRandomData();

}

public static void main(String[] args) {

ConsoleApp app = new ConsoleApp();

app.start();

}

}