Project Title: Boats Company Management System

Course code: CPS2232

Course Name: DATA STRUCTURE

Student Group List:

Student Name:Zhao qinjian Student Number: 1235624

Student Name: Jiao Luyao Student Number: 1235723

Student Name: Zhang Lei Student Number:1235752

Professor: Dr Ken Ehimwenma, Ph.D.

Date: December 17th, 2023

Abstract

This project focuses on the development of a boat management system for the fictional "Wenzhou-Kean University Fisher Boat Management Company" in the city of Wenzhou. The aim is to address the decentralized operations of local fishermen and facilitate efficient collaboration among them. Additionally, the system aims to boost the tourism industry by leasing idle boats to tourists. The boat management system offers comprehensive functions for both company employees and customers. Employees can import boat information from a CSV file and query it for clients. Customers can register accounts, browse available boats based on various attributes, and make purchases or lease agreements. The system records transaction details and provides waiting lists for boats in use. Technical implementation involves the use of data structures such as ArrayList and TreeMap to optimize boat storage and retrieval. Filtering methods are created using the computeIfAbsent function, improving efficiency. Challenges related to boat storage, lookup optimization, and user recommendation system design are addressed. The anticipated results include efficient collaboration among fishermen, improved resource utilization, and a boost to the tourism industry in Wenzhou. The boat management system aims to enhance the user experience by providing a user-friendly interface and convenient features. Further refinement may be required, but the successful implementation of this project will demonstrate the importance of leveraging technology for optimizing resource utilization and fostering collaboration within industries.

I. Introduction

1. Introduction

In the city of Wenzhou, a pivotal industry is fisheries; however, the operations of local fishermen are highly decentralized. Establishing a comprehensive boat management system would facilitate efficient collaboration among fishermen, allowing them to collectively utilize boats. Additionally, leasing idle boats to tourists has the potential to boost the tourism industry. Therefore, we have developed a project for the fictional "Wenzhou Kean University Fisher Boat Management Company," aimed at effectively recording and managing the company's specialized assets - boats.

2. Overview of Functions

For company employees, the system allows the import of information for all boats from a CSV file, which can be queried by clients. Users can view customer registration information and transaction records based on exported logs. For customers, various operations are available through the login menu. Users can register an account with a unique username, password, and email. User registration information is stored in a CSV file for program retrieval and employee queries. Upon logging into their registered accounts, users can choose to purchase, lease, or return boats.

Boats possess multiple attributes such as make, variant, length, region, price, year, owner, and user. When purchasing or leasing, users can filter boats based on these attributes. For each attribute filtered, the program will display all boats meeting the criteria along with their attributes. Users can continue to filter based on other attributes. After selecting a suitable boat, users confirm the purchase or lease and make the respective payment or deposit.

If a purchase is made, the boat's owner becomes the customer, removing it from consideration for other clients. If leased, the user is recorded as the boat's temporary user, and it is temporarily unavailable for rent. The system records the rental and return times, and users pay the rent upon return, calculated based on the boat's price and duration of lease. After returning, the user's association with the boat is cleared. User transaction records are logged for employee review.

Moreover, if a boat is currently in use, users wishing to lease it will be added to its waiting list. Upon the boat's return, the program sends email notifications to users on the waiting list.。

3. Innovation and Contribution

During the programming process, we encountered three main challenges. Here are our approaches to addressing them for the benefit of others:

1. Efficient Boat Storage:

- Challenge: Handling a large number of boats posed a significant challenge in designing an efficient data structure. Balancing storage efficiency with fast retrieval speed was crucial.

- Solution: Initially, we considered using a double map structure with HashMaps of HashMaps to store boat information. The outer map's key was the boat name, and its value was an inner map. However, this structure faced challenges in data transfer between classes and conversion into ordered data for user filtering. After a team discussion, we opted for an `ArrayList` with objects representing each boat. We successfully transferred data between classes by exporting information from the memory stack to a file.

2. Boat Lookup Optimization:

- Challenge: Switching to an `ArrayList` introduced concerns about the reduced speed of

traversal and lookup operations.

- Solution: In the `Database` file, we pre-created several sorters commonly used by clients and stored their results in a new `TreeMap`. During filtering, these pre-sorted data were automatically accessed, significantly speeding up the lookup process. Additionally, we used the `computeIfAbsent` function to create various filtering methods, extracting objects meeting specific conditions from the input `ArrayList` and storing them in a new `ArrayList`. This function accelerated the filtering process. Notably, we transformed boolean filtering using extensive research to leverage the `computeIfAbsent` function.

3. User Recommendation System Design:

- Challenge: The extensive options available to users resulted in a complex menu system, leading to code clutter.

- Solution: We stacked multiple `while` loops, `switch` functions, and other control structures for boat filtering and the recommendation system. However, this resulted in confusion regarding the use of the `Scanner`. Then we find use `Scanner`as parameter can settle. Further research is needed to refine the design and resolve this issue.

4. Details for lambda expression:

- When we use foreach, we find that lambda is not suitable for implementing variable traversal.

- Solution: After consulting the information, we found that when using Lambda expressions and forEach loops, if you try to modify externally defined variables inside the Lambda expression, a compilation error will occur. This is because the variables referenced inside the lambda expression must be final or effectively final. After searching, we know that we can use a java custom method AtomicInteger to solve this problem.

II. Related Works

In his work, Zheng M discusses the low risk of removal associated with the computeIfAbsent method. He explores the impact of different APIs on the removal of lambda expressions and finds that lambdas passed to APIs such as Map#computeIfAbsent and Optional#ifPresent are more likely to be removed compared to those passed to other APIs. On the other hand, lambdas used in Stream#map and Stream#filter methods are less likely to be removed.

Long et al. emphasizes in his work, "Java Concurrency Guidelines," that using atomic types like AtomicBoolean ensures visibility of write operations to other threads. This guarantees thread safety and prevents race conditions or inconsistencies when accessing or modifying the flag in a concurrent environment.

Bajracharya's review on Java HashMap and TreeMap explains that HashMap, based on hashing, enables fast retrieval of elements by keys. TreeMap, based on a Red-Black tree, provides efficient sorting and ordered traversal. HashMap is recommended for fast key-based retrieval, while TreeMap is suitable for scenarios requiring sorting or maintaining specific order. By leveraging the appropriate collection based on specific requirements, developers can enhance performance and create efficient and robust solutions.

Gil, J also mentions a fused hashing encoding method for HashMap and HashSet, which significantly reduces memory overhead by 20% to 45% on a 32-bit environment and 45% to 65% on a 64-bit environment. Regardless of key distribution within hash buckets, these figures serve as lower bounds. It is important to address potential performance bottlenecks in Java caused by containers like HashMap and HashSet.

According to Jacobson et al., with the release of Java 5.0, the ArrayList structure has become superior to arrays in representing contiguous lists both conceptually and in implementation. This shift can lead to more efficient and effective programming practices in

the long run.

III. Methodology

The entire program comprises four major systems: 1. Boat Reading System, 2. Filtering System, 3. Menu System, and 4. Waiting List. Around these systems, various auxiliary functions are implemented as well. We will present all the details in the order of their implementation:

	A	B	C	D	E	F	G
1	Make	Variant	Length	Region	Country	Price	Year
2	Alubat	Ovni 395	41	Europe	France	$267,233	2005
3	Bavaria	38 Cruiser	38	Europe	Croatia	$75,178	2005
4	Bavaria	38 Cruiser	38	Europe	Croatia	$66,825	2005
5	Bavaria	38 Cruiser	38	Europe	Croatia	$54,661	2005
6	Bavaria	38 Cruiser	38	Europe	Croatia	$53,447	2005
7	Bavaria	38 Cruiser	38	Europe	Greece	$91,101	2005
8	Bavaria	39 Cruiser	39	Europe	Greece	$66,748	2005
9	Bavaria	42 Match	41	Europe	Croatia	$78,945	2005
10	Bavaria	42 Match	41	Europe	Croatia	$58,297	2005
11	Bavaria	42 Cruiser	42	Europe	Croatia	$112,906	2005

Table 1 Sample Data

1. Boat Data Source: Boat data is stored in a CSV table, with each row representing a boat and each column representing a specific attribute. Above is a snippet example of a dataset sourced from the 2022 MCM competition problem, containing over 3000 boats, making it a substantial dataset. In real-life scenarios, employees of the company would collect and input boat information into this table.
2. Boat Class: A new class, `Boat`, is abstracted based on common attributes shared by boats. Data fields include characteristic properties of boats, along with owner, user, and waiting list.
3. Reading and Recording Boat Information: An `ArrayList<Boat>` is established to store all boat data. Using a `FileReader`, data is read line by line from the CSV file. For each line read, a new `Boat` object is created and added to the ArrayList. After reading all

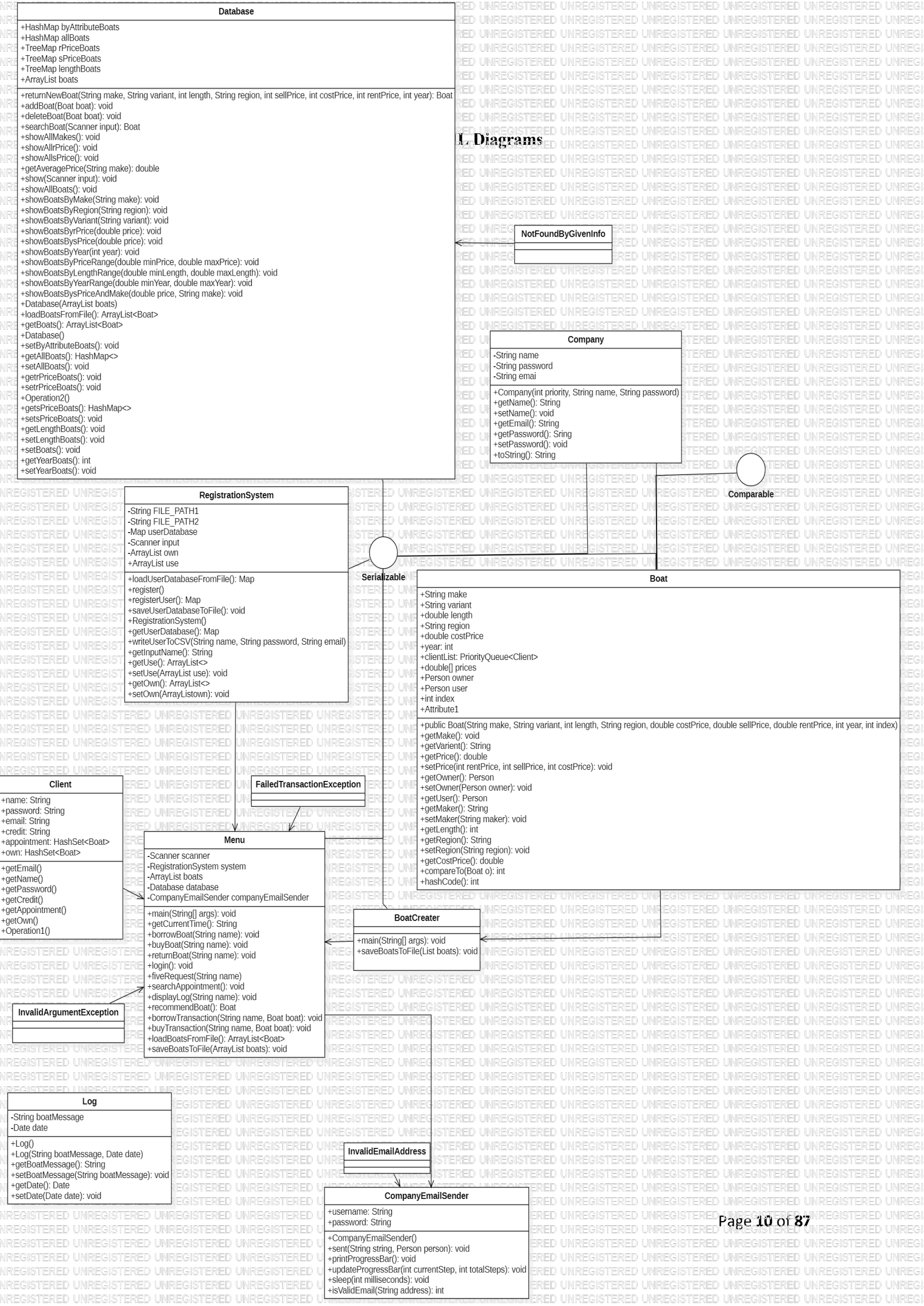
information, the details of each object in the ArrayList are retrieved from the stack using the `saveBoatsToFile` method. This information is then stored in a "boats" file, allowing data circulation between different classes.

4. Data Sorting: For numerical attributes (length, sellPrice, rentPrice, year), multiple `TreeMap` sorters are created, and their results are stored in a new `TreeMap`.
5. Data Filtering: For character attributes (make, variant, etc.), multiple filters are created using Java's built-in `computeIfAbsent` function. Results are stored in a new `ArrayList`. For numerical attributes, the TreeMap sorting results are used to limit values within specified upper and lower bounds for filtering.
6. Client Class: Data fields include username, password, email, using boat and owning boat.
7. User Registration System: This system accepts user input for username, password, and email, storing the user's information in a `HashMap` where the key is the username, and the value is a `Client` object. If a duplicate username is detected, the user is prompted to input a new username.
8. User Menu and Exception Handling: The `Menu.java` file is designed for user interaction. It facilitates user registration and login, rejecting access if the username does not exist or the password is incorrect. In the logged-in state, it employs multiple `while` loops and a `switch` function to call filtering methods for corresponding attributes in the `Database` class based on case numbers. Invalid inputs trigger error messages, prompting users to re-enter information. It then calls relevant functions to assist users with payment, boat return, and other operations.
9. Operating system of company's terminal: add, delete, modify and check the ships in the warehouse, and company terminal can also manage and view user information.
10. Email notification: When a user logs into the system, the company's mailbox can receive emails. When the log reaches 20 entries, the company mailbox will receive logs from all users. This can display the company's current transaction records with all users without

generating too many emails. When users log in to the system and the transaction is successful, they can also receive email reminders. Users can also receive email reminders when log reach 20.

11. User transaction and log: User can search boats the company have by searching main characteristic such as make, variant, year and length. After searching a boat, user can determine buy, return or borrow boat. After buy transaction, boat can be stored in client's field ArrayList<Boat>, and the boat will change the user from company to client. One thing should be notice is that the data structure storage boat will note remove the choosing boat, we just set user and owner to get to know whether client can choose this boat. This step can let company manage boats efficiently.

L. Diagrams


Database
+HashMap byAttributeBoats
+HashMap allBoats
+TreeMap rPriceBoats
+TreeMap sPriceBoats
+TreeMap lengthBoats
+ArrayList boats
+returnNewBoat(String make, String variant, int length, String region, int sellPrice, int costPrice, int rentPrice, int year): Boat
+addBoat(Boat boat): void
+deleteBoat(Boat boat): void
+searchBoat(Scanner input): Boat
+showAllMakes(): void
+showAllrPrice(): void
+showAllsPrice(): void
+getAveragePrice(String make): double
+show(Scanner input): void
+showAllBoats(): void
+showBoatsByMake(String make): void
+showBoatsByRegion(String region): void
+showBoatsByVariant(String variant): void
+showBoatsByrPrice(double price): void
+showBoatsBysPrice(double price): void
+showBoatsByYear(int year): void
+showBoatsByPriceRange(double minPrice, double maxPrice): void
+showBoatsByLengthRange(double minLength, double maxLength): void
+showBoatsByYearRange(double minYear, double maxYear): void
+showBoatsBysPriceAndMake(double price, String make): void
+Database(ArrayList boats)
+loadBoatsFromFile(): ArrayList<Boat>
+getBoats(): ArrayList<Boat>
+Database()
+setByAttributeBoats(): void
+getAllBoats(): HashMap<>
+setAllBoats(): void
+getrPriceBoats(): void
+setrPriceBoats(): void
+Operation2()
+getsPriceBoats(): HashMap<>
+setsPriceBoats(): void
+getLengthBoats(): void
+setLengthBoats(): void
+setBoats(): void
+getYearBoats(): int
+setYearBoats(): void
NotFoundByGivenInfo
Company
-String name
-String password
-String emai
+Company(int priority, String name, String password)
+getName(): String
+setName(): void
+getEmail(): String
+getPassword(): Sring
+setPassword(): void
+toString(): String
Comparable
RegistrationSystem
-String FILE_PATH1
-String FILE_PATH2
-Map userDatabase
-Scanner input
-ArrayList own
+ArrayList use
+loadUserDatabaseFromFile(): Map
+register()
+registerUser(): Map
+saveUserDatabaseToFile(): void
+RegistrationSystem()
+getUserDatabase(): Map
+writeUserToCSV(String name, String password, String email)
+getInputName(): String
+getUse(): ArrayList<>
+setUse(ArrayList use): void
+getOwn(): ArrayList<>
+setOwn(ArrayListown): void
Serializable
Boat
+String make
+String variant
+double length
+String region
+double costPrice
+year: int
+clientList: PriorityQueue<Client>
+double[] prices
+Person owner
+Person user
+int index
+Attribute1
+public Boat(String make, String variant, int length, String region, double costPrice, double sellPrice, double rentPrice, int year, int index)
+getMake(): void
+getVarient(): String
+getPrice(): double
+setPrice(int rentPrice, int sellPrice, int costPrice): void
+getOwner(): Person
+setOwner(Person owner): void
+getUser(): Person
+getMaker(): String
+setMaker(String maker): void
+getLength(): int
+getRegion(): String
+setRegion(String region): void
+getCostPrice(): double
+compareTo(Boat o): int
+hashCode(): int
Client
+name: String
+password: String
+email: String
+credit: String
+appointment: HashSet<Boat>
+own: HashSet<Boat>
+getEmail()
+getName()
+getPassword()
+getCredit()
+getAppointment()
+getOwn()
+Operation1()
FailedTransactionException
Menu
-Scanner scanner
-RegistrationSystem system
-ArrayList boats
-Database database
-CompanyEmailSender companyEmailSender
+main(String[] args): void
+getCurrentTime(): String
+borrowBoat(String name): void
+buyBoat(String name): void
+returnBoat(String name): void
+login(): void
+fiveRequest(String name)
+searchAppointment(): void
+displayLog(String name): void
+recommendBoat(): Boat
+borrowTransaction(String name, Boat boat): void
+buyTransaction(String name, Boat boat): void
+loadBoatsFromFile(): ArrayList<Boat>
+saveBoatsToFile(ArrayList boats): void
BoatCreater
+main(String[] args): void
+saveBoatsToFile(List boats): void
InvalidArgumentException
Log
-String boatMessage
-Date date
+Log()
+Log(String boatMessage, Date date)
+getBoatMessage(): String
+setBoatMessage(String boatMessage): void
+getDate(): Date
+setDate(Date date): void
InvalidEmailAddress
CompanyEmailSender
+username: String
+password: String
+CompanyEmailSender()
+sent(String string, Person person): void
+printProgressBar(): void
+updateProgressBar(int currentStep, int totalSteps): void
+sleep(int milliseconds): void
+isValidEmail(String address): int

b)

c) UML Code Generation

Asset.java 1 | Boat.java 9+ | Settings

D: > 课本 > CPS 2232 > project > 114514 > Model > Boat.java > ...

```java
import java.util.*;

/**
 *
 */
public class Boat implements Serializable, Asset, Comparable {

    /**
     * Default constructor
     */
    public Boat() {
    }

    /**
     *
     */
    public void String make;

    /**
     *
     */
    public void String variant;

    /**
     *
     */
    public void double length;

    /**
     *
     */
    public void String region;

    /**
     *
     */
    public void double costPrice;

    /**
     *
     */
    public int year;

    /**
     *
     */
    public PriorityQueue<Client> clientList;

    /**
     *
     */
    public void double[] prices;

    /**
```

```
58      public void Person owner;
59
60      /**
61       *
62       */
63      public void Person user;
64
65      /**
66       *
67       */
68      public void int index;
69
70      /**
71       *
72       */
73      public void Attribute1;
74
75      /**
76       * @param String make
77       * @param String variant
78       * @param int length
79       * @param String region
80       * @param double costPrice
81       * @param double sellPrice
82       * @param double rentPrice
83       * @param int year
84       * @param int index
85       */
86      public void public Boat(void String make, void String variant,
87      void int length, void String region, void double costPrice,
88       void double sellPrice, void double rentPrice, void int year, void int index) {
89          // TODO implement here
90      }
91
92      /**
93       * @return
94       */
95      public void getMake() {
96          // TODO implement here
97          return null;
98      }
99
100     /**
101      * @return
102      */
103     public String getVarient() {
104         // TODO implement here
105         return "";
106     }
107
108     /**
109      * @return
110      */
111     public double getPrice() {
112         // TODO implement here
```

```
114     /**
115      * @param int rentPrice
116      * @param int sellPrice
117      * @param int costPrice
118      * @return
119      */
120     public void setPrice(void int rentPrice, void int
121         // TODO implement here
122         return null;
123     }
124
125     /**
126      * @return
127      */
128     public Person getOwner() {
129         // TODO implement here
130         return null;
131     }
132
133     /**
134      * @param Person owner
135      * @return
136      */
137     public void setOwner(void Person owner) {
138         // TODO implement here
139         return null;
140     }
141
142     /**
143      * @return
144      */
145     public Person getUser() {
146         // TODO implement here
147         return null;
148     }
149
150     /**
151      * @return
152      */
153     public String getMaker() {
154         // TODO implement here
155         return "";
156     }
157
158     /**
159      * @param String maker
160      * @return
161      */
162     public void setMaker(void String maker) {
163         // TODO implement here
164         return null;
165     }
166
167     /**
168      * @return
169      */
```

```
165         }
166
167         /**
168          * @return
169          */
170         public int getLength() {
171             // TODO implement here
172             return 0;
173         }
174
175         /**
176          * @return
177          */
178         public String getRegion() {
179             // TODO implement here
180             return "";
181         }
182
183         /**
184          * @param String region
185          * @return
186          */
187         public void setRegion(void String region) {
188             // TODO implement here
189             return null;
190         }
191
192         /**
193          * @return
194          */
195         public double getCostPrice() {
196             // TODO implement here
197             return 0.0d;
198         }
199
200         /**
201          * @param Boat o
202          * @return
203          */
204         public int compareTo(void Boat o) {
205             // TODO implement here
206             return 0;
207         }
208
209         /**
210          * @return
211          */
212         public int hashCode() {
213             // TODO implement here
214             return 0;
215         }
216
217     }
```

Asset.java 1 | Boat.java 9+ | BoatCreater.java 9+ | Settings

D: > 课本 > CPS 2232 > project > 114514 > Model > BoatCreater.java > ...

```java

import java.util.*;

/**
 *
 */
public class BoatCreater implements Serializable {

    /**
     * Default constructor
     */
    public BoatCreater() {
    }

    /**
     * @param String[] args
     * @return
     */
    public void main(void String[] args) {
        // TODO implement here
        return null;
    }

    /**
     * @param List boats
     * @return
     */
    public void saveBoatsToFile(void List boats) {
        // TODO implement here
        return null;
    }

}
```

Asset.java 1 | Boat.java 9+ | BoatCreater.java 9+ | BoatReader.java 1

D: > 课本 > CPS 2232 > project > 114514 > Model > BoatReader.java > ...

```java

import java.util.*;

/**
 *
 */
public class BoatReader {

    /**
     * Default constructor
     */
    public BoatReader() {
    }

}
```

Asset.java 1 | Boat.java 9+ | BoatCreater.jav

D: > 课本 > CPS 2232 > project > 114514 > Model > Client

```java
import java.util.*;

/**
 *
 */
public class Client {

    /**
     * Default constructor
     */
    public Client() {
    }

    /**
     *
     */
    public String name;

    /**
     *
     */
    public String password;

    /**
     *
     */
    public String email;

    /**
     *
     */
    public String credit;

    /**
     *
     */
    public HashSet<Boat> appointment;

    /**
     *
     */
    public HashSet<Boat> own;

    /**
     *
     */
    public void getEmail() {
        // TODO implement here
    }

    /**
     *
     */
    public void getName() {
        // TODO implement here
```

Asset.java 1 | Boat.java 9+ | BoatCreater.java 9+ | BoatReader.java 1 | Client.java 1 | Settings

D: > 课本 > CPS 2232 > project > 114514 > Model > Client.java > ...

```
    }

    /**
     *
     */
    public void getAppointment() {
        // TODO implement here
    }

    /**
     *
     */
    public void getOwn() {
        // TODO implement here
    }

    /**
     *
     */
    public void Operation1() {
        // TODO implement here
    }

}
```

Company.java 9+ | Settings

D: > 课本 > CPS 2232 > project > 114514 > Model > Company.java > ...

```
import java.util.*;

/**
 *
 */
public class Company implements Serializable {

    /**
     * Default constructor
     */
    public Company() {
    }

    /**
     *
     */
    private void String name;

    /**
     *
     */
    private void String password;

    /**
     *
     */
    private void String emai;

    /**
     * @param int priority
     * @param String name
     * @param String password
     */
```

Company.java 9+ Settings

D: > 课本 > CPS 2232 > project > 114514 > Model > Company.java > ...

```
34          */
35          public void Company(void int priority, void String name, void String password) {
36              // TODO implement here
37          }
38
39          /**
40           * @return
41           */
42          public String getName() {
43              // TODO implement here
44              return "";
45          }
46
47          /**
48           * @return
49           */
50          public void setName() {
51              // TODO implement here
52              return null;
53          }
54
55          /**
56           * @return
57           */
58          public String getEmail() {
59              // TODO implement here
60              return "";
61          }
62
63          /**
64           * @return
65           */
66          public Sring getPassword() {
67              // TODO implement here
68              return null;
69          }
70
71          /**
72           * @return
73           */
74          public void setPassword() {
75              // TODO implement here
76              return null;
77          }
78
79          /**
80           * @return
81           */
82          public String toString() {
83              // TODO implement here
84              return "";
85          }
86
87      }
```

Company.java 9+ | CompanyEmailSender.java 9+ | Settings

D: > 课本 > CPS 2232 > project > 114514 > Model > CompanyEmailSender.java > ...

```java

import java.util.*;

/**
 *
 */
public class CompanyEmailSender {

    /**
     * Default constructor
     */
    public CompanyEmailSender() {
    }

    /**
     *
     */
    public String username;

    /**
     *
     */
    public String password;

    /**
     *
     */
    public void CompanyEmailSender() {
        // TODO implement here
    }

    /**
     * @param String string
     * @param Person person
     * @return
     */
    public void sent(void String string, void Person person) {
        // TODO implement here
        return null;
    }

    /**
     * @return
     */
    public void printProgressBar() {
        // TODO implement here
        return null;
    }

    /**
     * @param int currentStep
     * @param int totalSteps
     * @return
     */
    public void updateProgressBar(void int currentStep, void int totalSteps) {
        // TODO implement here
```

Company.java 9+ | CompanyEmailSender.java 9+ | Settings

D: > 课本 > CPS 2232 > project > 114514 > Model > CompanyEmailSender.java > ...

```java
     * @return
     */
    public void updateProgressBar(void int currentStep, void int totalSteps) {
        // TODO implement here
        return null;
    }

    /**
     * @param int milliseconds
     * @return
     */
    public void sleep(void int milliseconds) {
        // TODO implement here
        return null;
    }

    /**
     * @param String address
     * @return
     */
    public int isValidEmail(void String address) {
        // TODO implement here
        return 0;
    }

}
```

Comparable.java 1 | Settings

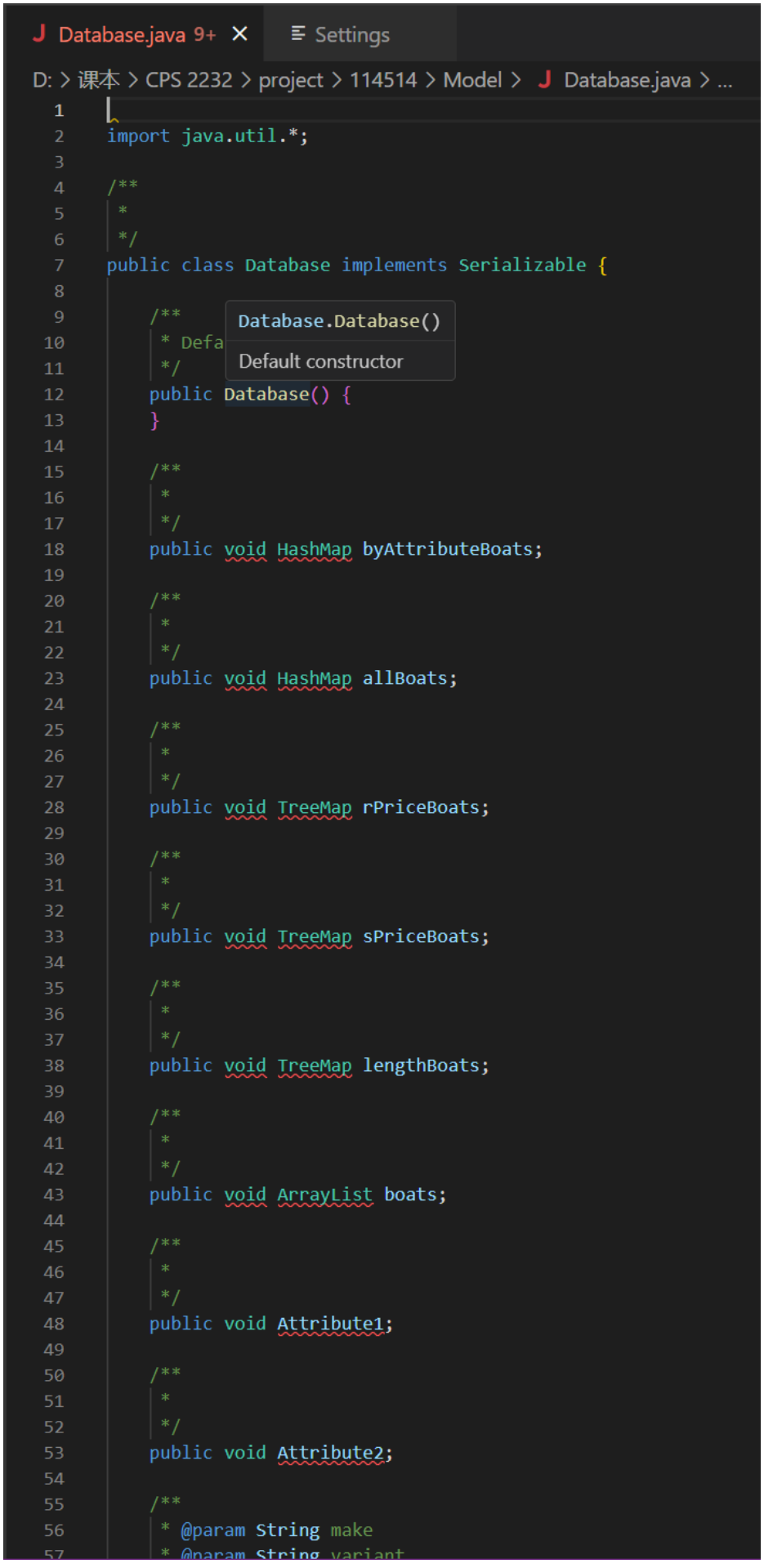
D: > 课本 > CPS 2232 > project > 114514 > N

```java

import java.util.*;

/**
 *
 */
public interface Comparable {

}
```

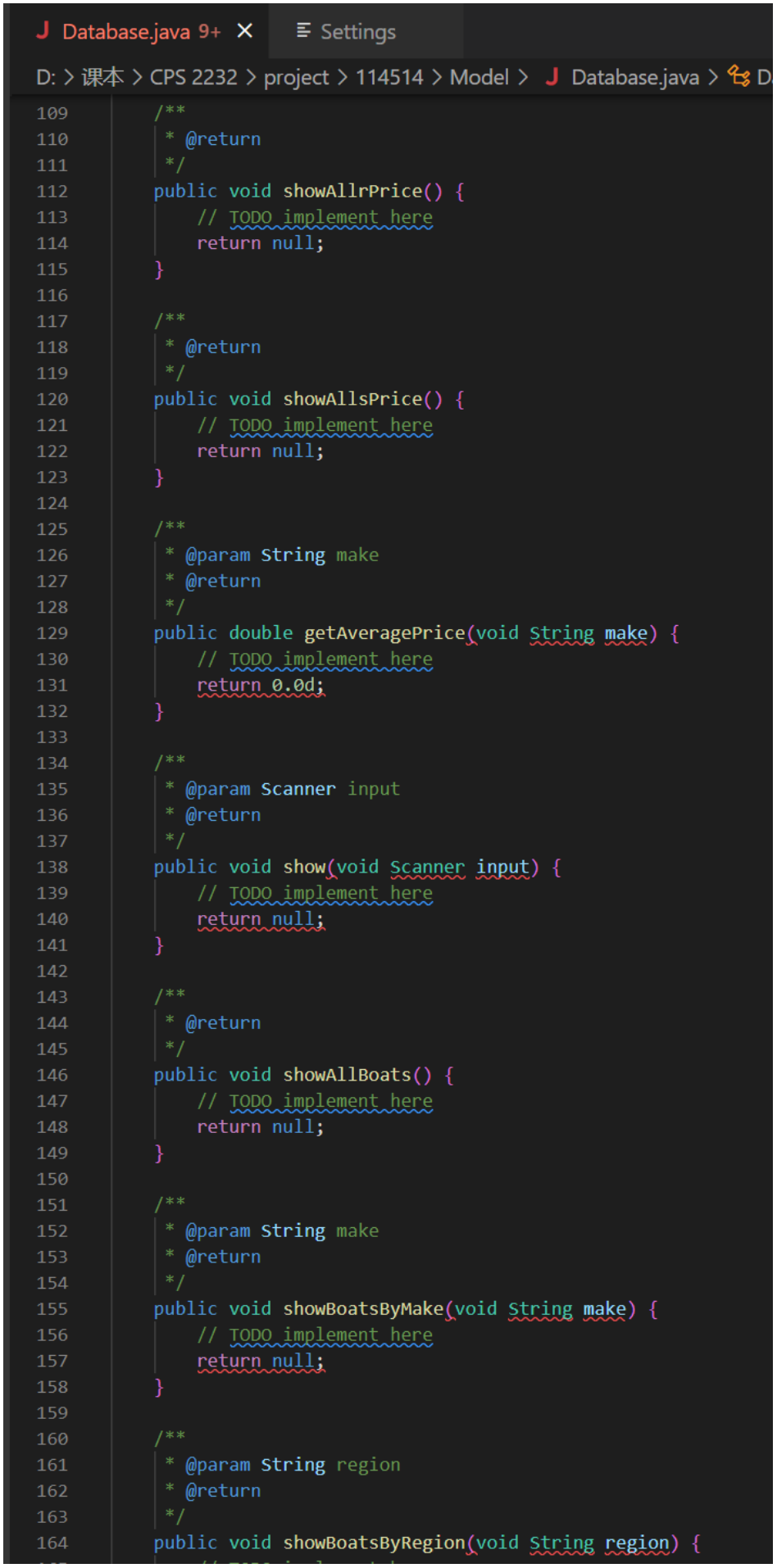
Asset.java 1

D: > 课本 > CPS 2232 > project > 114514 > Model > Asset.java > ...

```java

import java.util.*;

/**
 *
 */
public interface Asset {

}
```


Database.java 9+
Settings
D: > 课本 > CPS 2232 > project > 114514 > Model > Database.java > ...
Database.Database()
Default constructor
import java.util.*;
/**
*
*/
public class Database implements Serializable {
/**
* Defa
*/
public Database() {
}
/**
*
*/
public void HashMap byAttributeBoats;
/**
*
*/
public void HashMap allBoats;
/**
*
*/
public void TreeMap rPriceBoats;
/**
*
*/
public void TreeMap sPriceBoats;
/**
*
*/
public void TreeMap lengthBoats;
/**
*
*/
public void ArrayList boats;
/**
*
*/
public void Attribute1;
/**
*
*/
public void Attribute2;
/**
* @param String make

```
53      public void Attribute2;
54
55      /**
56       * @param String make
57       * @param String variant
58       * @param int length
59       * @param String region
60       * @param int sellPrice
61       * @param int costPrice
62       * @param int rentPrice
63       * @param int year
64       * @return
65       */
66      public Boat returnNewBoat(void String make,
67      void String variant, void int length,
68      void String region, void int sellPrice,
69      void int costPrice, void int rentPrice, void int year) {
70          // TODO implement here
71          return null;
72      }
73
74      /**
75       * @param Boat boat
76       * @return
77       */
78      public void addBoat(void Boat boat) {
79          // TODO implement here
80          return null;
81      }
82
83      /**
84       * @param Boat boat
85       * @return
86       */
87      public void deleteBoat(void Boat boat) {
88          // TODO implement here
89          return null;
90      }
91
92      /**
93       * @param Scanner input
94       * @return
95       */
96      public Boat searchBoat(void Scanner input) {
97          // TODO implement here
98          return null;
99      }
100
101     /**
102      * @return
103      */
104     public void showAllMakes() {
105         // TODO implement here
106         return null;
107     }
108
```


Database.java 9+
Settings
D: > 课本 > CPS 2232 > project > 114514 > Model > Database.java > D
109 /**
110 * @return
111 */
112 public void showAllrPrice() {
113 // TODO implement here
114 return null;
115 }
116
117 /**
118 * @return
119 */
120 public void showAllsPrice() {
121 // TODO implement here
122 return null;
123 }
124
125 /**
126 * @param String make
127 * @return
128 */
129 public double getAveragePrice(void String make) {
130 // TODO implement here
131 return 0.0d;
132 }
133
134 /**
135 * @param Scanner input
136 * @return
137 */
138 public void show(void Scanner input) {
139 // TODO implement here
140 return null;
141 }
142
143 /**
144 * @return
145 */
146 public void showAllBoats() {
147 // TODO implement here
148 return null;
149 }
150
151 /**
152 * @param String make
153 * @return
154 */
155 public void showBoatsByMake(void String make) {
156 // TODO implement here
157 return null;
158 }
159
160 /**
161 * @param String region
162 * @return
163 */
164 public void showBoatsByRegion(void String region) {

Database.java 9+ | Settings

D: > 课本 > CPS 2232 > project > 114514 > Model > Database.java > Database > returnNewBoat()

```
160     /**
161      * @param String region
162      * @return
163      */
164     public void showBoatsByRegion(void String region) {
165         // TODO implement here
166         return null;
167     }
168
169     /**
170      * @param String variant
171      * @return
172      */
173     public void showBoatsByVariant(void String variant) {
174         // TODO implement here
175         return null;
176     }
177
178     /**
179      * @param double price
180      * @return
181      */
182     public void showBoatsByrPrice(void double price) {
183         // TODO implement here
184         return null;
185     }
186
187     /**
188      * @param double price
189      * @return
190      */
191     public void showBoatsBysPrice(void double price) {
192         // TODO implement here
193         return null;
194     }
195
196     /**
197      * @param int year
198      * @return
199      */
200     public void showBoatsByYear(void int year) {
201         // TODO implement here
202         return null;
203     }
204
205     /**
206      * @param double minPrice
207      * @param double maxPrice
208      * @return
209      */
210     public void showBoatsByPriceRange(void double minPrice, void double maxPrice) {
211         // TODO implement here
212         return null;
213     }
214
215     /**
216      * @param double minLength
```

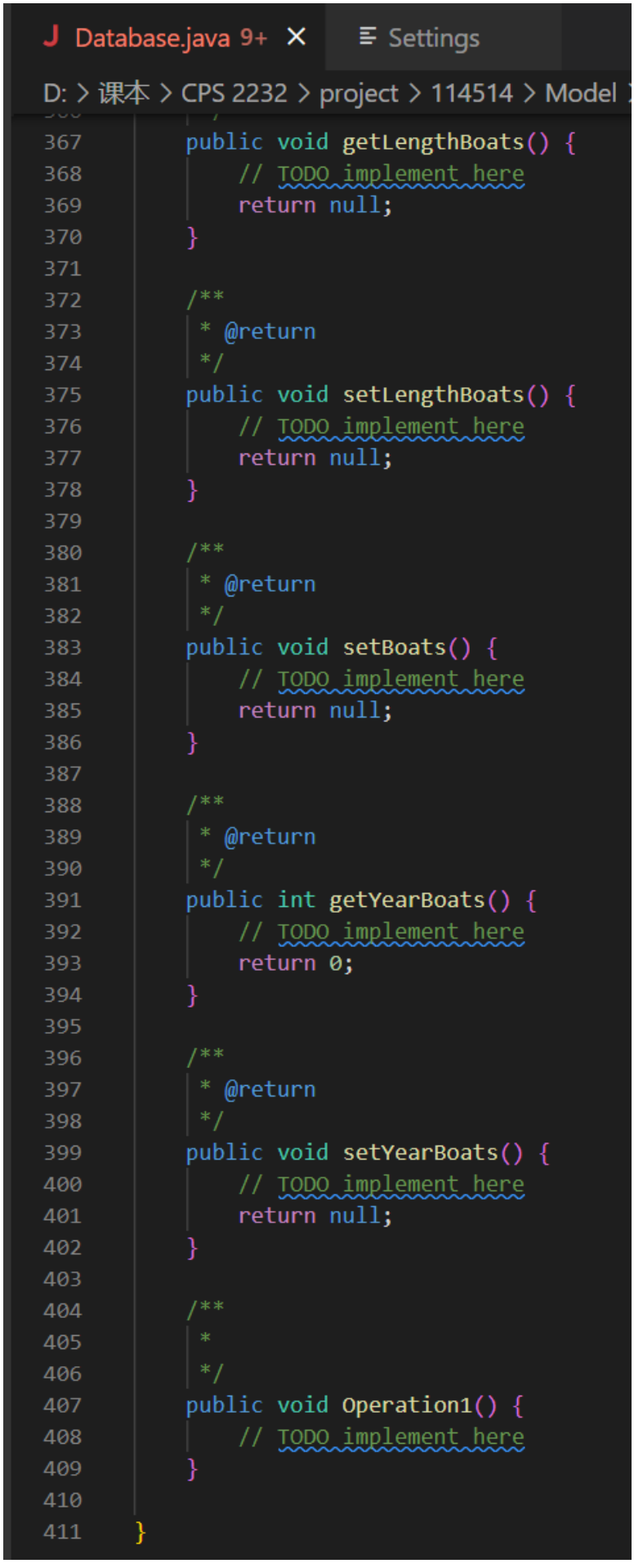
Database.java 9+ ● Settings

D: > 课本 > CPS 2232 > project > 114514 > Model > Database.java > Database > String

```
212         return null;
213     }
214
215     /**
216      * @param double minLength
217      * @param double maxLength
218      * @return
219      */
220     public void showBoatsByLengthRange(void double
221      minLength, void double maxLength) {
222         // TODO implement here
223         return null;
224     }
225
226     /**
227      * @param double minYear
228      * @param double maxYear
229      * @return
230      */
231     public void showBoatsByYearRange(void double minYear,
232     void double maxYear) {
233         // TODO implement here
234         return null;
235     }
236
237     /**
238      * @param double price
239      * @param String make
240      * @return
241      */
242     public void showBoatsBysPriceAndMake(void double price,
243     void String make) {
244         // TODO implement here
245         return null;
246     }
247
248     /**
249      * @param ArrayList boats
250      */
251     public void Database(void ArrayList boats) {
252         // TODO implement here
253     }
254
255     /**
256      * @return
257      */
258     public ArrayList<Boat> loadBoatsFromFile() {
259         // TODO implement here
260         return null;
261     }
262
263     /**
264      * @return
265      */
266     public ArrayList<Boat> getBoats() {
267         // TODO implement here
```


Database.java 9+
Settings
D: > 课本 > CPS 2232 > project > 114514 > Model > Database.java
264 * @return
265 */
266 public ArrayList<Boat> getBoats() {
267 // TODO implement here
268 return null;
269 }
270
271 /**
272 *
273 */
274 public void Database() {
275 // TODO implement here
276 }
277
278 /**
279 * @return
280 */
281 public void setByAttributeBoats() {
282 // TODO implement here
283 return null;
284 }
285
286 /**
287 * @return
288 */
289 public HashMap<> getAllBoats() {
290 // TODO implement here
291 return null;
292 }
293
294 /**
295 * @return
296 */
297 public void setByAttributeBoats() {
298 // TODO implement here
299 return null;
300 }
301
302 /**
303 * @return
304 */
305 public void setAllBoats() {
306 // TODO implement here
307 return null;
308 }
309
310 /**
311 * @return
312 */
313 public HashMap<> getAllBoats() {
314 // TODO implement here
315 return null;
316 }
317
318 /**
319 * @return


Database.java 9+
Settings
D: > 课本 > CPS 2232 > project > 114514 > Model >
*/
public void getrPriceBoats() {
// TODO implement here
return null;
}
/**
*
*/
public void setAllBoats() {
// TODO implement here
}
/**
* @return
*/
public void setrPriceBoats() {
// TODO implement here
return null;
}
/**
*
*/
public void Operation2() {
// TODO implement here
}
/**
* @return
*/
public HashMap<> getsPriceBoats() {
// TODO implement here
return null;
}
/**
* @return
*/
public void setsPriceBoats() {
// TODO implement here
return null;
}
/**
* @return
*/
public void getLengthBoats() {
// TODO implement here
return null;
}
/**
* @return
*/
public void setLengthBoats() {


Database.java 9+
Settings
D: > 课本 > CPS 2232 > project > 114514 > Model
public void getLengthBoats() {
// TODO implement here
return null;
}
/**
* @return
*/
public void setLengthBoats() {
// TODO implement here
return null;
}
/**
* @return
*/
public void setBoats() {
// TODO implement here
return null;
}
/**
* @return
*/
public int getYearBoats() {
// TODO implement here
return 0;
}
/**
* @return
*/
public void setYearBoats() {
// TODO implement here
return null;
}
/**
*
*/
public void Operation1() {
// TODO implement here
}
}

FailedTransactionException.java 1 Settings

D: > 课本 > CPS 2232 > project > 114514 > Model > Fa

```java

import java.util.*;

/**
 *
 */
public class FailedTransactionException {

    /**
     * Default constructor
     */
    public FailedTransactionException() {
    }

}
```

ImportClient.java 1 Settings

D: > 课本 > CPS 2232 > project > 114514

```java

import java.util.*;

/**
 *
 */
public class ImportClient {

    /**
     * Default constructor
     */
    public ImportClient() {
    }

}
```

InvalidArgumentException.java 1 Settings

D: > 课本 > CPS 2232 > project > 114514 > Model >

```java

import java.util.*;

/**
 *
 */
public class InvalidArgumentException {

    /**
     * Default constructor
     */
    public InvalidArgumentException() {
    }

}
```

InvalidEmailAddress.java 1 Settings

D: > 课本 > CPS 2232 > project > 114514 > Model

```java

import java.util.*;

/**
 *
 */
public class InvalidEmailAddress {

    /**
     * Default constructor
     */
    public InvalidEmailAddress() {
    }

}
```

Log.java 9+ Settings

D: > 课本 > CPS 2232 > project > 114514 > Model > Log.java > ...

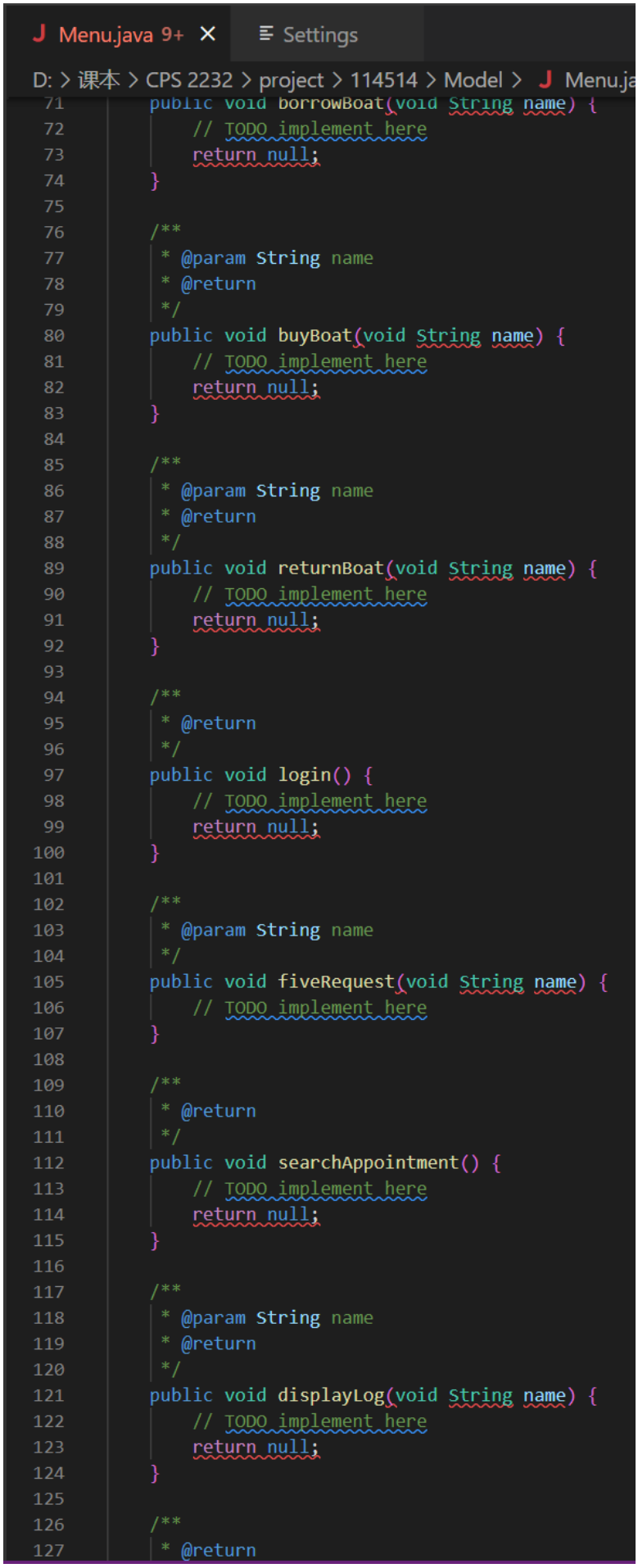
```java

import java.util.*;

/**
 *
 */
public class Log {

    /**
     * Default constructor
     */
    public Log() {
    }

    /**
     *
     */
    private void String boatMessage;

    /**
     *
     */
    private void Date date;

    /**
     *
     */
    public void Attribute1;

    /**
     *
     */
    public void Log() {
        // TODO implement here
    }

```

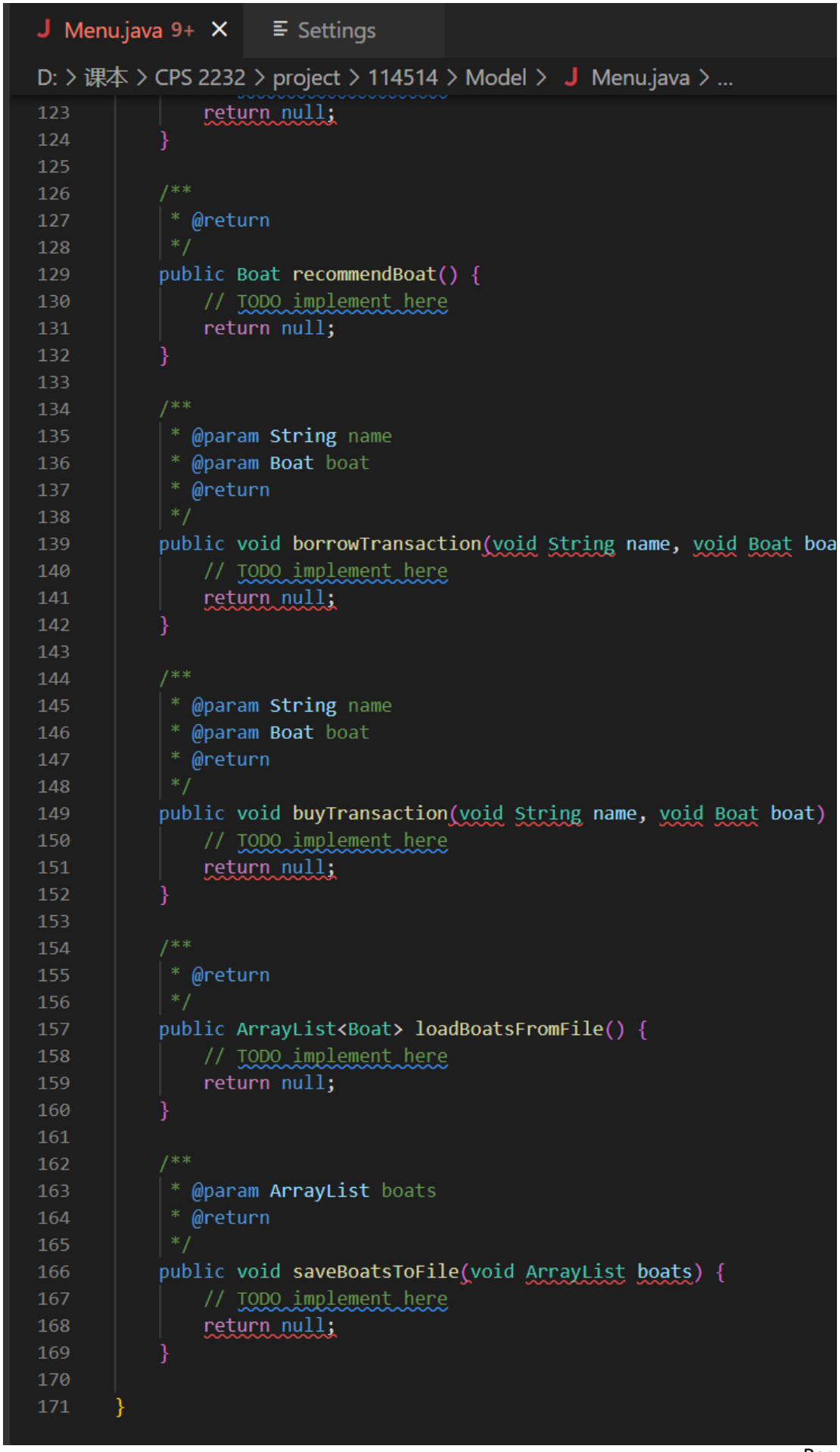

Log.java 9+
Settings
D: > 课本 > CPS 2232 > project > 114514 > Model > Log.java > ...
29
30 /**
31 *
32 */
33 public void Log() {
34 // TODO implement here
35 }
36
37 /**
38 * @param String boatMessage
39 * @param Date date
40 */
41 public void Log(void String boatMessage, void Date date) {
42 // TODO implement here
43 }
44
45 /**
46 * @return
47 */
48 public String getBoatMessage() {
49 // TODO implement here
50 return "";
51 }
52
53 /**
54 * @param String boatMessage
55 * @return
56 */
57 public void setBoatMessage(void String boatMessage) {
58 // TODO implement here
59 return null;
60 }
61
62 /**
63 * @return
64 */
65 public Date getDate() {
66 // TODO implement here
67 return null;
68 }
69
70 /**
71 * @param Date date
72 * @return
73 */
74 public void setDate(void Date date) {
75 // TODO implement here
76 return null;
77 }
78
79 }

Menu.java 9+ ✕ Settings

D: > 课本 > CPS 2232 > project > 114514 > Model > Menu.java > ...

```java
import java.util.*;

/**
 *
 */
public class Menu implements Serializable {

    /**
     * Default constructor
     */
    public Menu() {
    }

    /**
     *
     */
    private void Scanner scanner;

    /**
     *
     */
    private void RegistrationSystem system;

    /**
     *
     */
    private void ArrayList boats;

    /**
     *
     */
    private void Database database;

    /**
     *
     */
    public void ArrayList boats;

    /**
     *
     */
    private void CompanyEmailSender companyEmailSender;

    /**
     *
     */
    public void Attribute2;

    /**
     * @param String[] args
     * @return
     */
    public void main(void String[] args) {
        // TODO implement here
        return null;
    }
```


Menu.java 9+
Settings
D: > 课本 > CPS 2232 > project > 114514 > Model > Menu.ja
public void borrowBoat(void String name) {
// TODO implement here
return null;
}
/**
* @param String name
* @return
*/
public void buyBoat(void String name) {
// TODO implement here
return null;
}
/**
* @param String name
* @return
*/
public void returnBoat(void String name) {
// TODO implement here
return null;
}
/**
* @return
*/
public void login() {
// TODO implement here
return null;
}
/**
* @param String name
*/
public void fiveRequest(void String name) {
// TODO implement here
}
/**
* @return
*/
public void searchAppointment() {
// TODO implement here
return null;
}
/**
* @param String name
* @return
*/
public void displayLog(void String name) {
// TODO implement here
return null;
}
/**
* @return


Menu.java 9+
Settings
D: > 课本 > CPS 2232 > project > 114514 > Model > Menu.java > ...
123 return null;
124 }
125
126 /**
127 * @return
128 */
129 public Boat recommendBoat() {
130 // TODO implement here
131 return null;
132 }
133
134 /**
135 * @param String name
136 * @param Boat boat
137 * @return
138 */
139 public void borrowTransaction(void String name, void Boat boa
140 // TODO implement here
141 return null;
142 }
143
144 /**
145 * @param String name
146 * @param Boat boat
147 * @return
148 */
149 public void buyTransaction(void String name, void Boat boat)
150 // TODO implement here
151 return null;
152 }
153
154 /**
155 * @return
156 */
157 public ArrayList<Boat> loadBoatsFromFile() {
158 // TODO implement here
159 return null;
160 }
161
162 /**
163 * @param ArrayList boats
164 * @return
165 */
166 public void saveBoatsToFile(void ArrayList boats) {
167 // TODO implement here
168 return null;
169 }
170
171 }

NotFoundByGivenInfo.java 1 Settings

D: > 课本 > CPS 2232 > project > 114514 > Model >

```java

import java.util.*;

/**
 *
 */
public class NotFoundByGivenInfo {

    /**
     * Default constructor
     */
    public NotFoundByGivenInfo() {
    }

}
```

Operations.java 1 Settings

D: > 课本 > CPS 2232 > project > 114514 >

```java

import java.util.*;

/**
 *
 */
public class Operations {

    /**
     * Default constructor
     */
    public Operations() {
    }

}
```

RegistrationSystem.java 9+ Settings

D: > 课本 > CPS 2232 > project > 114514 > Model > RegistrationSyster

```java

import java.util.*;

/**
 *
 */
public class RegistrationSystem implements Serializable {

    /**
     * Default constructor
     */
    public RegistrationSystem() {
    }

    /**
     *
     */
    private void String FILE_PATH1;

    /**
     *
```


RegistrationSystem.java 9+
Settings
D: > 课本 > CPS 2232 > project > 114514 > Model > Regi
20 /**
21 *
22 */
23 private void String FILE_PATH2;
24
25 /**
26 *
27 */
28 private void Map userDatabase;
29
30 /**
31 *
32 */
33 private void Scanner input;
34
35 /**
36 *
37 */
38 private void ArrayList own;
39
40 /**
41 *
42 */
43 public void ArrayList use;
44
45 /**
46 * @return
47 */
48 public Map loadUserDatabaseFromFile() {
49 // TODO implement here
50 return null;
51 }
52
53 /**
54 *
55 */
56 public void register() {
57 // TODO implement here
58 }
59
60 /**
61 * @return
62 */
63 public Map registerUser() {
64 // TODO implement here
65 return null;
66 }
67
68 /**
69 * @return
70 */
71 public void saveUserDatabaseToFile() {
72 // TODO implement here
73 return null;
74 }
75

RegistrationSystem.java 9+ ● Settings

D: > 课本 > CPS 2232 > project > 114514 > Model > RegistrationS

```java
    /**
     *
     */
    public void RegistrationSystem() {
        // TODO implement here
    }

    /**
     * @return
     */
    public Map getUserDatabase() {
        // TODO implement here
        return null;
    }

    /**
     * @param String name
     * @param String password
     * @param String email
     */
    public void writeUserToCSV(void String name,
    void String password, void String email) {
        // TODO implement here
    }

    /**
     * @return
     */
    public String getInputName() {
        // TODO implement here
        return "";
    }

    /**
     * @return
     */
    public ArrayList<> getUse() {
        // TODO implement here
        return null;
    }

    /**
     * @param ArrayList use
     * @return
     */
    public void setUse(void ArrayList use) {
        // TODO implement here
        return null;
    }

    /**
     * @return
     */
    public ArrayList<> getOwn() {
        // TODO implement here
        return null;
    }
```

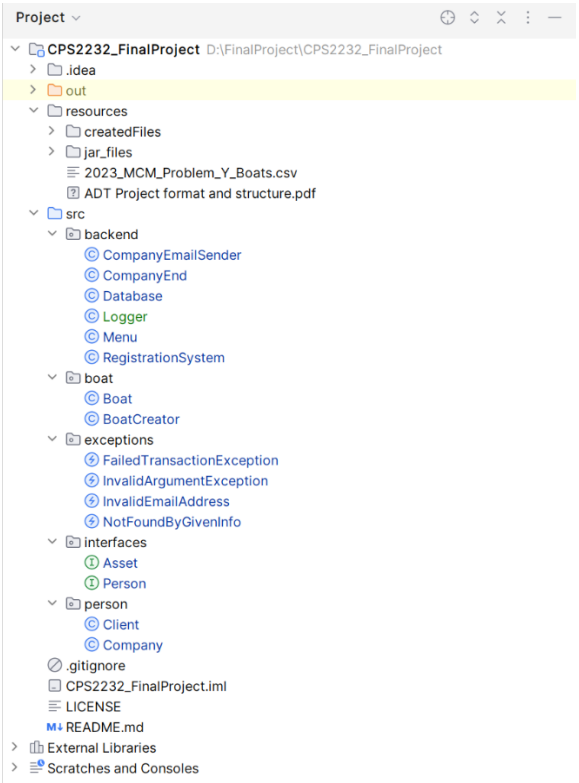
```
126     /**
127      * @return
128      */
129     public ArrayList<> getOwn() {
130         // TODO implement here
131         return null;
132     }
133
134     /**
135      * @param ArrayListown
136      * @return
137      */
138     public void setOwn(void ArrayListown) {
139         // TODO implement here
140         return null;
141     }
142
143 }
```

Serializable.java 1 ✕ Settings

D: > 课本 > CPS 2232 > project > 114514 > N

```
1
2   import java.util.*;
3
4   /**
5    *
6    */
7   public interface Serializable {
8
9   }
```

d) Full Code Implantation (& Screenshots)




```java
package backend;

import exceptions.InvalidEmailAddress;
import interfaces.Person;
import person.Client;
import javax.mail.*;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import java.io.Serializable;
import java.util.Properties;

/**
 * Purpose: This class is used to send email to the client or company administrator
 * @version 1.0
 * @since 2023-12-17
 */

4 usages  Albert Zhao *
public class CompanyEmailSender implements Serializable {
    2 usages
    final String username = "seacraft2018@outlook.com";
    1 usage
    final String password ="cps2232finalproject";


    2 usages  Albert Zhao
    public CompanyEmailSender() throws MessagingException {
    }

    /**
     * This method is used to send email to the client or company administrator
     * @param string the content of the email
     * @param person the person who will receive the email
     * @throws InvalidEmailAddress if the email address is invalid
     */
    8 usages  Albert Zhao
```

```java
public void sent(String string,Person person) throws InvalidEmailAddress {
    if (isValidEmail(person.getEmail())==-1){
        throw new InvalidEmailAddress("Invalid email address");
    }
    printProgressBar();
    // Get the default Session object.
    Properties props = new Properties();
    props.put("mail.smtp.host", "smtp.office365.com");
    props.put("mail.smtp.port", "587");
    props.put("mail.smtp.auth", "true");
    props.put("mail.smtp.starttls.enable", "true");
    Session session = Session.getInstance(props, getPasswordAuthentication() -> {
            return new PasswordAuthentication(username, password);
    });

    try {
        // Create a default MimeMessage object.
        Message message = new MimeMessage(session);

        // Set From: header field of the header.
        message.setFrom(new InternetAddress(username));

        // Set To: header field of the header.
        message.setRecipients(Message.RecipientType.TO, InternetAddress.parse(person.getEmail()));
        if (person instanceof Client){
            // Set Subject: header field
            message.setSubject("Notification from SeaCraft Company");

            // Now set the actual message
            message.setText(string);

            // Send message
            Transport.send(message);
        }
        else {
            message.setSubject("SeaCraft Company management logs");

            // Now set the actual message
            message.setText(string);

            // Send message
            Transport.send(message);
        }
        System.out.println(".......100% Task completed!");
        System.out.println("Email sent successfully!");

    }
    catch (MessagingException e) {
        throw new RuntimeException(e);
    }
}

//fake progress bar to show the process of the program, stop at one point when finish print 100%
1 usage  Albert Zhao
public static void printProgressBar() {
    for (int i = 0; i <= 45; i++) {
        updateProgressBar(i, totalSteps: 45);
        sleep( milliseconds: 10);
    }
}
//progress bar in the console, it is a fake progress bar, just for relax the user
1 usage  Albert Zhao
public static void updateProgressBar(int currentStep, int totalSteps) {
    int barLength = 66;
    int progress = (int) ((double) currentStep / totalSteps * barLength);

    StringBuilder progressBar = new StringBuilder("[");
    for (int i = 0; i < barLength; i++) {
        if (i < progress) {
            progressBar.append("=");
        } else {
            progressBar.append(" ");
```

```
            }
        }
        progressBar.append("] ");

        int percent = (int) (((double) currentStep / totalSteps) * 100);
        if (percent > 95) {
            return;
        }
        progressBar.append(percent).append("%");

        System.out.print("\r" + progressBar);
        System.out.flush();
    }

    //used for progress bar
    1 usage   Albert Zhao
    private static void sleep(int milliseconds) {
        try {
            Thread.sleep(milliseconds);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }


    /**
     * This method is used to check whether the email address is valid
     * @param address the email address
     * @return -1 if the email address is invalid, 1 if the email address is valid
     */
    2 usages   Albert Zhao
    public static int isValidEmail(String address) {
        int returnValue = 1;//this is for end output
        //idea: if invalid then give return value -1 else valid.

        //first:find out the place of "@"
        int placeOfAt = address.indexOf('@');
        if (placeOfAt == -1)
            return -1;//return -1 if invalid
        int lengthOfAddress = address.length();


        //substring 2 part
        String part1 = address.substring(0, placeOfAt);
        String part2 = address.substring( beginIndex: placeOfAt + 1);

        //judgement one: @ and '.' can't be begin or end
        //pickout 1st and end character check whether @or.
        char beginning = address.charAt(0);
        char ending = address.charAt(lengthOfAddress - 1);
        if (beginning == '.' || ending == '.' || beginning == '@' || ending == '@')
            return -1;


        //judgment two:all characters in part1/2 are digit,letter or '.'.
        //using for loop to check if the each are matched
        int lengthOfPart1 = part1.length();
        for (int begNum = 0; begNum < lengthOfPart1; begNum++) {
            char eachCharacter = part1.charAt(begNum);
            boolean digitJudge = Character.isDigit(eachCharacter);
            boolean letterJudge = Character.isLetter(eachCharacter);
            boolean dotJudge = (int) eachCharacter == 46;
            if (digitJudge && letterJudge && dotJudge == false) {
                begNum = lengthOfPart1 + 1;
                returnValue = -1;
            }
        }

        int lengthOfPart2 = part2.length();
        for (int begNum = 0; begNum < lengthOfPart2; begNum++) {
            char eachCharacter = part2.charAt(begNum);
            boolean digitJudge = Character.isDigit(eachCharacter);
```

```
        }

        int lengthOfPart2 = part2.length();
        for (int begNum = 0; begNum < lengthOfPart2; begNum++) {
            char eachCharacter = part2.charAt(begNum);
            boolean digitJudge = Character.isDigit(eachCharacter);
            boolean letterJudge = Character.isLetter(eachCharacter);
            boolean dotJudge = (int) eachCharacter == 46;
            if (digitJudge && letterJudge && dotJudge == false) {
                begNum = lengthOfPart1 + 1;
                returnValue = -1;
            }
        }

        //judgement three: . cannot beside @
        //end of part1/begin of part2 not '.'
        char endOfPart1 = part1.charAt(lengthOfPart1 - 1);
        char beginOfPart2 = part2.charAt(0);

        if (endOfPart1 == '.' || beginOfPart2 == '.')
            returnValue = -1;


        if (returnValue == -1)
            return returnValue;//return -1 if invalid

        return returnValue;
    }


}
```

```
1  package backend;
2
3
4  import ...
14
15 /**
16  * Purpose: This class is used to provide the functions for the company end
17  * @version 1.0
18  * @since 2023-12-17
19  */
20
   2 usages  Albert Zhao *
21 public class CompanyEnd {
       6 usages
22     Database database;
       1 usage
23     Company company = new Company( priority: 1, name: "Seacraft 2018 Boat Company", password: "cps");
       1 usage
24     RegistrationSystem registrationSystem = new RegistrationSystem();
       1 usage
25     Map<String, Client> userInfo =  registrationSystem.loadUserDatabaseFromFile();
26
       1 usage  Albert Zhao
27     public CompanyEnd(Database database) { this.database = database; }
30
       4 usages  Albert Zhao
31 @   public void ui(Scanner input) throws NotFoundByGivenInfo {
32         System.out.println("Verify your identity:");
33         System.out.println("Please choose the function you want to use:");
34         System.out.println("1 Login");
35         System.out.println("2 Exit");
36         System.out.print("Your choice: ");
37         int choice = input.nextInt();
38         switch (choice) {
39             case 1:
40                 System.out.print("Please input the company unified password: ");
```

```
                String password = input.next();
                if (company.getPassword().equals(password)) {
                    System.out.println("Login successfully.");
                    this.menu(input);
                } else {
                    System.out.println("Wrong password. Please try again.");
                    ui(input);
                }
                break;
            case 2:
                System.out.println("Thank you for using the WKU Boat Company!");
                System.exit( status: 0);
                break;
            default:
                System.out.println("Invalid input. Please try again.");
                ui(input);
        }
    }

    8 usages   Albert Zhao *
    public void menu(Scanner input) throws NotFoundByGivenInfo {
        String s = null;
        System.out.println("Please choose the function you want to use:");
        System.out.println("1 Add a new boat");
        System.out.println("2 Delete a boat");
        System.out.println("3 Modify a boat");
        System.out.println("4 Search a boat");
        System.out.println("5 show user information");
        System.out.println("6 show boats information");
        System.out.print("Enter Your choice please: ");

        int choice = input.nextInt();

        switch (choice) {
            case 1:
                System.out.println("Please input the information of the boat you want to add:");
```

```
                System.out.println("Please input the make of the boat:");
                String make = input.next();
                System.out.println("Please input the variant of the boat:");
                String variant = input.next();
                System.out.println("Please input the length of the boat:");
                int length = input.nextInt();
                System.out.println("Please input the region of the boat:");
                String region = input.next();
                System.out.println("Please input the year of the boat:");
                int year = input.nextInt();
                System.out.println("Please input the cost price of the boat:");
                double costPrice = input.nextDouble();
                System.out.println("Please input the sell price of the boat:");
                double sellPrice = input.nextDouble();
                System.out.println("Please input the rent price of the boat:");
                double rentPrice = input.nextDouble();
                Boat boat = new Boat(make, variant, length, region, costPrice, sellPrice, rentPrice, year,  index: allBoats.size()-1);
                database.addBoat(boat);
                System.out.println("continue?\ny/n");
                s = input.next();
                if(s.equals("y")){
                    menu(input);
                }
                else{
                    System.out.println("Thank you for using the WKU Boat Company!");
                    System.exit( status: 0);
                }

                break;
            case 2:
                database.deleteBoat(askBoatInfo(input, action: "delete"));
                System.out.println("continue?\ny/n");
                s = input.next();
                if(s.equals("y")){
                    menu(input);
                }
```

```
            else{
                System.out.println("Thank you for using Company End System!");
                System.exit( status: 0);
            }
            break;
        case 3:
            database.modifyBoat(input,askBoatInfo(input, action: "modify"));
            System.out.println("continue?\ny/n");
            s = input.next();
            if(s.equals("y")){
                menu(input);
            }
            else{
                System.out.println("Thank you for using the Seacraft2018 Boat Company!");
                System.exit( status: 0);
            }
            break;
        case 4:try {
            searchBoat(input);

            System.out.println("continue?\ny/n");
            s = input.next();
            if(s.equals("y")){
                menu(input);
            }
            else{
                System.out.println("Thank you for using the Seacraft2018 Boat Company!");
                System.exit( status: 0);
            }
        }catch (NotFoundByGivenInfo e){
            System.out.println(e.getMessage());
        }
            break;
        case 5:
            System.out.println("user information:");
            userInfo.forEach((k,v)->{
                System.out.printf("[name:%-10spassword:%10semail:%20s]\n",k,v.getPassword(),v.getEmail());
            });
            System.out.println("continue?\ny/n");
             s=input.next();
            if(s.equals("y")){
                menu(input);
            }
            else{
                System.out.println("Thank you for using the Seacraft2018 Boat Company!");
                System.exit( status: 0);
            }
            break;
        case 6:
            database.show(input);
            System.out.println("continue?\ny/n");
            s = input.next();
            if(s.equals("y")){
                menu(input);
            }
            else{
                System.out.println("Thank you for using the Seacraft2018 Boat Company!");
                System.exit( status: 0);
            }
            break;
        default:
            System.out.println("Invalid input. Please try again.");
            menu(input);
    }

}

/**
 * This method is used to ask the user to input the information of the boat.
 * @param input
 * @param action
 * @return Boat
```

```java
package backend;

import ...

/**
 * This class is to store all the boats and offer methods to search the boats by different attributes
 * @version 1.0
 * @since 2023-12-17
 */

8 usages  Albert Zhao *
public class Database implements Serializable {
    16 usages
    HashMap<String,ArrayList<Boat>> byAttributeBoats = new HashMap<>();
    17 usages
    static HashMap<Boat,Boat> allBoats = new HashMap<>();
    9 usages
    TreeMap<Double,ArrayList<Boat>> rPriceBoats = new TreeMap<>();
    9 usages
    TreeMap<Double,ArrayList<Boat>> sPriceBoats = new TreeMap<>();
    6 usages
    TreeMap<Integer,ArrayList<Boat>> lengthBoats = new TreeMap<>();
    11 usages
    ArrayList<Boat> boats = loadBoatsFromFile();
    7 usages
    TreeMap<Integer,ArrayList<Boat>> yearBoats = new TreeMap<>();

    6 usages  new *
    public Database(){

    }

    7 usages  new *
    public Database(ArrayList<Boat> boats) {...}

    1 usage  Albert Zhao
```

```java
    public Database(ArrayList<Boat> boats) {
        this.boats = boats;
        //use different attribute to file the database

        for (Boat boat : boats
        ) {
            byAttributeBoats.computeIfAbsent(boat.getMake(), k -> new ArrayList<>()).add(boat);//by MAKE
        }
        for (Boat boat : boats
        ) {
            byAttributeBoats.computeIfAbsent(boat.getRegion(), k -> new ArrayList<>()).add(boat);//by REGION
        }
        for (Boat boat : boats
        ) {
            byAttributeBoats.computeIfAbsent(boat.getVarient(), k -> new ArrayList<>()).add(boat);//by VARIANT
        }
        for (Boat boat : boats
        ) {
            rPriceBoats.computeIfAbsent(boat.getPrice()[0], k -> new ArrayList<>()).add(boat);//by PRICE
            sPriceBoats.computeIfAbsent(boat.getPrice()[1], k -> new ArrayList<>()).add(boat);//by PRICE
        }
        for (Boat boat : boats
        ) {
            yearBoats.computeIfAbsent(boat.getYear(), k -> new ArrayList<>()).add(boat);//by PRICE}
        }
        for (Boat boat : boats
        ) {
            allBoats.put(boat,boat);
        }
        for (Boat boat : boats
        ) {
            lengthBoats.computeIfAbsent( boat.getLength(), k -> new ArrayList<>()).add(boat);//by PRICE
        }
    }
```

```
64      public Boat returnNewBoat(String make, String variant, int length, String region, double sellPrice, double costPrice
65                                int year){
66          return new Boat(make,variant,length,region,sellPrice,costPrice,rentPrice,year, index: boats.size()-1);
67      }
68
        3 usages  Albert Zhao
69      private static void saveBoatsToFile(List<Boat> boats) {
70          String filePath = "resources/createdFiles/allboats";
71
72          try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(filePath))) {
73              oos.writeObject(boats);
74              System.out.println("Boats list has been saved to file: " + filePath);
75          } catch (IOException e) {
76              e.printStackTrace();
77          }
78      }
79
        2 usages  Albert Zhao
80      public void addBoat(Boat boat){
81          allBoats.put(boat,boat);
82          byAttributeBoats.computeIfAbsent(boat.getMake(), k -> new ArrayList<>()).add(boat);//by MAKE
83          byAttributeBoats.computeIfAbsent(boat.getRegion(), k -> new ArrayList<>()).add(boat);//by REGION
84          byAttributeBoats.computeIfAbsent(boat.getVarient(), k -> new ArrayList<>()).add(boat);//by VARIANT
85          rPriceBoats.computeIfAbsent(boat.getPrice()[0], k -> new ArrayList<>()).add(boat);//by PRICE
86          sPriceBoats.computeIfAbsent(boat.getPrice()[1], k -> new ArrayList<>()).add(boat);//by PRICE
87          yearBoats.computeIfAbsent(boat.getYear(), k -> new ArrayList<>()).add(boat);//by PRICE}
88          lengthBoats.computeIfAbsent( boat.getLength(), k -> new ArrayList<>()).add(boat);//by PRICE
89          boats.add(boat);
90          saveBoatsToFile(boats);
91      }
92
        2 usages  Albert Zhao
93      public void deleteBoat(Boat boat){
94          allBoats.remove(boat);
95          byAttributeBoats.get(boat.getMake()).remove(boat);
96          byAttributeBoats.get(boat.getRegion()).remove(boat);
168                     System.out.println("Thank you for using the Seacraft2018 Boat Company!");
169                     System.exit( status: 0);
170                 }
171                 break;
172             default:
173                 System.out.println("Invalid input. Please try again.");
174                 menu(input);
175         }
176
177     }
178
179     /**
180      * This method is used to ask the user to input the information of the boat.
181      * @param input
182      * @param action
183      * @return Boat
184      * @throws NotFoundByGivenInfo
185      */
        2 usages  Albert Zhao
186     public Boat askBoatInfo(Scanner input,String action) throws NotFoundByGivenInfo {
187         System.out.println("Please input the information of the boat you want to"  + action +":");
188         Boat boat = database.searchBoat(input);
189         return boat;
190
191     }
192
193
194
195
196 }
197
```

```
                allBoats.forEach((k,v)->{
                    if (k.getLength() == Double.parseDouble(infos[0])){
                        result.add(k);
                    }
                } );
                break;
            case '3':
                infos[0] = infos[0].substring( beginIndex: 2);
                allBoats.forEach((k,v)->{
                    if (k.getRegion().equals(infos[0])){
                        result.add(k);
                    }
                } );
                break;
            case '4':
                infos[0] = infos[0].substring( beginIndex: 2);
                allBoats.forEach((k,v)->{
                    if (k.getPrice()[0] == Double.parseDouble(infos[0])){
                        result.add(k);
                    }
                } );
                break;
            case '5':
                infos[0] = infos[0].substring( beginIndex: 2);
                allBoats.forEach((k,v)->{
                    if (k.getPrice()[1] == Double.parseDouble(infos[0])){
                        result.add(k);
                    }
                } );
                break;
            case '6':
                infos[0] = infos[0].substring( beginIndex: 2);
                allBoats.forEach((k,v)->{
                    if (k.getPrice()[2] == Double.parseDouble(infos[0])){
                        result.add(k);
                    }
                } );
                break;
            case '7':
                infos[0] = infos[0].substring( beginIndex: 2);
                allBoats.forEach((k,v)->{
                    if (k.getYear() == Integer.parseInt(infos[0])){
                        result.add(k);
                    }
                } );
                break;
        }


        //delete boats that do not match the second condition
        Iterator<Boat> it = result.iterator();
        for (int i = 1; i < infos.length; i++) {
            switch (infos[i].charAt(0)) {
                case '0':
                    infos[i] = infos[i].substring( beginIndex: 2);
                    while (it.hasNext())
                    {
                        Boat boat = it.next();
                        if (!boat.getMake().equals(infos[i])) {
                            it.remove();
                        }
                    }
                    break;
                case '1':
                    infos[i] = infos[i].substring( beginIndex: 2);
                    while (it.hasNext())
                    {
                        Boat boat = it.next();
                        if (!boat.getVarient().equals(infos[i])) {
                            it.remove();
                        }
                    }
```

```
                    break;
                case '2':
                    infos[i] = infos[i].substring( beginIndex: 2);
                    while (it.hasNext())
                    {
                        Boat boat = it.next();
                        if (boat.getLength() != Double.parseDouble(infos[i])) {
                            it.remove();
                        }
                    }
                    break;
                case '3':
                    infos[i] = infos[i].substring( beginIndex: 2);
                    while (it.hasNext())
                    {
                        Boat boat = it.next();
                        if (!boat.getRegion().equals(infos[i])) {
                            it.remove();
                        }
                    }
                    break;
                case '4':
                    infos[i] = infos[i].substring( beginIndex: 2);
                    while (it.hasNext())
                    {
                        Boat boat = it.next();
                        if (boat.getPrice()[0] != Double.parseDouble(infos[i])) {
                            it.remove();
                        }
                    }
                    break;
                case '5':
                    infos[i] = infos[i].substring( beginIndex: 2);
                    while (it.hasNext())
                    {
                        Boat boat = it.next();
                        if (boat.getPrice()[1] != Double.parseDouble(infos[i])) {
                            it.remove();
                        }
                    }
                    break;
                case '6':
                    infos[i] = infos[i].substring( beginIndex: 2);
                    while (it.hasNext())
                    {
                        Boat boat = it.next();
                        if (boat.getPrice()[2] != Double.parseDouble(infos[i])) {
                            it.remove();
                        }
                    }
                    break;
                case '7':
                    infos[i] = infos[i].substring( beginIndex: 2);
                    while (it.hasNext())
                    {
                        Boat boat = it.next();
                        if (boat.getYear() != Integer.parseInt(infos[i])) {
                            it.remove();
                        }
                    }
                    break;
            }

        }
        if (result.size()==0){
            throw new NotFoundByGivenInfo("No such boats by given information");
        }
        if (result.size()==1){
            System.out.println(result.get(0));
            return result.get(0);
        }
        else {
```

```
            System.out.println("We find more than one boat by your given information, please choose one of them");
            for (int i = 0; i < result.size(); i++) {
                System.out.printf(i +": "+"%-30s%-40s%-15s%-20s%-15.2f%-15.2f%-10d%n",result.get(i).getMake(),
                        result.get(i).getVarient(),result.get(i).getLength(),result.get(i).getRegion(),
                        result.get(i).getPrice()[1],result.get(i).getPrice()[2],result.get(i).getYear());
            }
            System.out.println("Please enter your choice:");
            int choice = Integer.parseInt(input.nextLine());
            return result.get(choice);
        }

    }


    no usages  Albert Zhao
    public void showAllMakes(){
        Set<Boat> makes = allBoats.keySet();
        int count = 0;
        HashMap<String,Integer> makeCount = new HashMap<>();

        for (Boat make : makes
        ) {
            if (!makeCount.containsKey(make.getMake())){
                makeCount.put(make.getMake(),1);//count the number of each make
            }
            else {
                makeCount.put(make.getMake(),makeCount.get(make.getMake())+1);
            }
        }
        for (String make : makeCount.keySet()
        ) {
            count++;
            System.out.print("|" + makeCount.get(make) + " items for "+ make + "| ");
            if (count%5==0){
                System.out.println();
            }
        }
    }

    //show rent price
    no usages  Albert Zhao
    public void showAllrPrice(){
        TreeMap <Double,ArrayList<Boat>> rPrice = rPriceBoats;
        int count = 0;
        HashMap<String,Integer> makeCount = new HashMap<>();
        rPrice.forEach((k,v)->{
            int sum = v.size();
            System.out.print("| " +sum +" items at " + k +" $  each | ");
        });
    }

    //show sell price
    no usages  Albert Zhao
    public void showAllsPrice(){
        TreeMap <Double,ArrayList<Boat>> sPrice = sPriceBoats;
        int count = 0;
        HashMap<String,Integer> makeCount = new HashMap<>();
        sPrice.forEach((k,v)->{
            int sum = v.size();
            System.out.print("| " +sum +" items at " + k +" $  each | ");
        });
    }


    //get average sell price by make
    no usages  Albert Zhao
    public double getAveragePrice(String make){
        ArrayList<Boat> boats = byAttributeBoats.get(make);
        double sum = 0;
        for (Boat boat : boats
        ) {
            sum += boat.getPrice()[1];
```

```
384     public void show(Scanner input)  {
385
386         System.out.println("If you want to show all the boats, please enter 1");
387         System.out.println("If you want to show the boats by make, please enter 2");
388         System.out.println("If you want to show the boats by region, please enter 3");
389         System.out.println("If you want to show the boats by variant, please enter 4");
390         System.out.println("If you want to show the boats by rent price, please enter 5");
391         System.out.println("If you want to show the boats by sell price, please enter 6");
392         System.out.println("If you want to show the boats by year, please enter 7");
393         System.out.println("If you want to show the boats by price range, please enter 8");
394         System.out.println("If you want to show the boats by length range, please enter 9");
395         System.out.println("If you want to show the boats by year range, please enter 10");
396         System.out.println("If you want to show the boats by make with a largest cost please enter 11");
397         System.out.println("If you want to exit, please enter 0");
398         System.out.println("Please enter your choice:");
399         boolean valid = false;
400         do {
401             try{
402                 int i = Integer.parseInt(input.nextLine());
403                 switch (i) {
404                     case 1:
405                         showAllBoats();
406                         break;
407                     case 2:
408                         System.out.println("Please enter the make:");
409                         String make = input.nextLine();
410                         showBoatsByMake(make);
411                         break;
412                     case 3:
413                         System.out.println("Please enter the region:");
414                         String region = input.nextLine();
415                         showBoatsByRegion(region);
416                         break;
417                     case 4:
418                         System.out.println("Please enter the variant:");
419                         String variant = input.nextLine();
426                         break;
427                     case 6:
428                         System.out.println("Please enter the sell price:");
429                         double sPrice = Double.parseDouble(input.nextLine());
430                         showBoatsBysPrice(sPrice);
431                         break;
432                     case 7:
433                         System.out.println("Please enter the year:");
434                         int year = Integer.parseInt(input.nextLine());
435                         showBoatsByYear(year);
436                         break;
437                     case 8:
438                         System.out.println("Please enter the min price:");
439                         double minPrice = Double.parseDouble(input.nextLine());
440                         System.out.println("Please enter the max price:");
441                         double maxPrice = Double.parseDouble(input.nextLine());
442                         showBoatsByPriceRange(minPrice, maxPrice);
443                         break;
444                     case 9:
445                         System.out.println("Please enter the min length:");
446                         double minLength = Double.parseDouble(input.nextLine());
447                         System.out.println("Please enter the max length:");
448                         double maxLength = Double.parseDouble(input.nextLine());
449                         showBoatsByLengthRange(minLength, maxLength);
450                         break;
451                     case 10:
452                         System.out.println("Please enter the min year:");
453                         int minYear = Integer.parseInt(input.nextLine());
454                         System.out.println("Please enter the max year:");
455                         int maxYear = Integer.parseInt(input.nextLine());
456                         showBoatsByYearRange(minYear, maxYear);
457                         break;
458                     case 11:
459                         System.out.println("Please enter the price:");
460                         double price = Double.parseDouble(input.nextLine());
461                         System.out.println("Please enter the make:");
```

```
                    String make1 = input.nextLine();
                    showBoatsBysPriceAndMake(price, make1);
                    break;
                case 0:
                    break;

            }
            valid = true;
        }catch (NumberFormatException e){
            System.out.println("Please enter a valid number");
        }
        catch (NullPointerException e){//it's not a good idea to use null pointer exception, but It works here
            // cause nothing gonna cause null pointer exception here, and define a new exception is too much work
            System.out.println("If you want to show all the boats, please enter 1");
            System.out.println("If you want to show the boats by make, please enter 2");
            System.out.println("If you want to show the boats by region, please enter 3");
            System.out.println("If you want to show the boats by variant, please enter 4");
            System.out.println("If you want to show the boats by rent price, please enter 5");
            System.out.println("If you want to show the boats by sell price, please enter 6");
            System.out.println("If you want to show the boats by year, please enter 7");
            System.out.println("If you want to show the boats by price range, please enter 8");
            System.out.println("If you want to show the boats by length range, please enter 9");
            System.out.println("If you want to show the boats by year range, please enter 10");
            System.out.println("If you want to show the boats by price and make, please enter 11");
            System.out.println("If you want to exit, please enter 0");
            System.out.println("Please enter your choice:");
            System.out.println("We do not have this boat by your GIVEN INFORMATION,please enter 0 to exit or do the " +
                    "search again");
        }
        catch (NotFoundByGivenInfo e){
            System.out.println(e+"\n\n\n");
            System.out.println("If you want to show all the boats, please enter 1");
            System.out.println("If you want to show the boats by make, please enter 2");
            System.out.println("If you want to show the boats by region, please enter 3");
            System.out.println("If you want to show the boats by variant, please enter 4");
            System.out.println("If you want to show the boats by rent price, please enter 5");
            System.out.println("If you want to show the boats by sell price, please enter 6");
            System.out.println("If you want to show the boats by year, please enter 7");
            System.out.println("If you want to show the boats by price range, please enter 8");
            System.out.println("If you want to show the boats by length range, please enter 9");
            System.out.println("If you want to show the boats by year range, please enter 10");
            System.out.println("If you want to show the boats by price and make, please enter 11");
            System.out.println("If you want to exit, please enter 0");
            System.out.println("Please enter your choice:");
            System.out.println("We do not have this boat by your GIVEN INFORMATION,please enter 0 to exit or do the "
                    "search again");
        }
    }
    while (!valid);

}




/*
From now till the end of the blocks of methods, are the methods to show the boats by different attributes.
we offer a helper methods up here, so you do need to look at the methods below.
 */
1 usage  Albert Zhao
public void showAllBoats(){
    System.out.printf("%-30s%-35s%-30s%-30s%-15s%-15s%-10s%n","Make",
            "Variant","Length","Region","SellPrice","RentPrice","Year");
    for (Boat boat : allBoats.keySet()
    ) {
        System.out.printf("%-30s%-35s%-30s%-30s%-15.2f%-15.2f%-10d%n",boat.getMake(),
                boat.getVarient(),boat.getLength(),boat.getRegion(),
                boat.getPrice()[1],boat.getPrice()[2],boat.getYear());
    }
}
1 usage  Albert Zhao
public void showBoatsByMake(String make){
```

```
532         ArrayList<Boat> boats = byAttributeBoats.get(make);
533         System.out.printf("%-30s%-35s%-30s%-30s%-15s%-15s%-10s%n","Make",
534                 "Variant","Length","Region","SellPrice","RentPrice","Year");
535         for (Boat boat : boats
536         ) {
537             System.out.printf("%-30s%-35s%-30s%-30s%-15.2f%-15.2f%-10d%n",boat.getMake(),
538                     boat.getVarient(),boat.getLength(),boat.getRegion(),
539                     boat.getPrice()[1],boat.getPrice()[2],boat.getYear());
540         }
541     }
542
            1 usage   Albert Zhao
543     public void showBoatsByRegion(String region){
544         ArrayList<Boat> boats = byAttributeBoats.get(region);
545         System.out.printf("%-30s%-35s%-30s%-30s%-15s%-15s%-10s%n","Make",
546                 "Variant","Length","Region","SellPrice","RentPrice","Year");
547         for (Boat boat : boats
548         ) {
549             System.out.printf("%-30s%-35s%-30s%-30s%-15.2f%-15.2f%-10d%n",boat.getMake(),
550                     boat.getVarient(),boat.getLength(),boat.getRegion(),
551                     boat.getPrice()[1],boat.getPrice()[2],boat.getYear());
552         }
553     }
554
            1 usage   Albert Zhao
555     public void showBoatsByVariant(String variant){
556         ArrayList<Boat> boats = byAttributeBoats.get(variant);
557         System.out.printf("%-30s%-35s%-30s%-30s%-15s%-15s%-10s%n","Make",
558                 "Variant","Length","Region","SellPrice","RentPrice","Year");
559         for (Boat boat : boats
560         ) {
561             System.out.printf("%-30s%-35s%-30s%-30s%-15.2f%-15.2f%-10d%n",boat.getMake(),
562                     boat.getVarient(),boat.getLength(),boat.getRegion(),
563                     boat.getPrice()[1],boat.getPrice()[2],boat.getYear());
564         }
565     }
566
            1 usage   Albert Zhao
567     public void showBoatsByrPrice(double price){
568         ArrayList<Boat> boats = rPriceBoats.get(price);
569         System.out.printf("%-30s%-35s%-30s%-30s%-15s%-15s%-10s%n","Make",
570                 "Variant","Length","Region","SellPrice","RentPrice","Year");
571         for (Boat boat : boats
572         ) {
573             System.out.printf("%-30s%-35s%-30s%-30s%-15.2f%-15.2f%-10d%n",boat.getMake(),
574                     boat.getVarient(),boat.getLength(),boat.getRegion(),
575                     boat.getPrice()[1],boat.getPrice()[2],boat.getYear());
576         }
577     }
578
            1 usage   Albert Zhao
579     public void showBoatsBysPrice(double price){
580         ArrayList<Boat> boats = sPriceBoats.get(price);
581         System.out.printf("%-30s%-35s%-30s%-30s%-15s%-15s%-10s%n","Make",
582                 "Variant","Length","Region","SellPrice","RentPrice","Year");
583         for (Boat boat : boats
584         ) {
585             System.out.printf("%-30s%-35s%-30s%-30s%-15.2f%-15.2f%-10d%n",boat.getMake(),
586                     boat.getVarient(),boat.getLength(),boat.getRegion(),
587                     boat.getPrice()[1],boat.getPrice()[2],boat.getYear());
588         }
589     }
590
            1 usage   Albert Zhao
591     public void showBoatsByYear(int year){
592         ArrayList<Boat> boats = yearBoats.get(year);
593         System.out.printf("%-30s%-35s%-30s%-30s%-15s%-15s%-10s%n","Make",
594                 "Variant","Length","Region","SellPrice","RentPrice","Year");
595         for (Boat boat : boats
596         ) {
597             System.out.printf("%-30s%-35s%-30s%-30s%-15.2f%-15.2f%-10d%n",boat.getMake(),
598                     boat.getVarient(),boat.getLength(),boat.getRegion(),
```

```
1 usage  Albert Zhao
602 public void showBoatsByPriceRange(double minPrice, double maxPrice) throws NotFoundByGivenInfo {
603     TreeMap<Double,ArrayList<Boat>> boats = sPriceBoats;
604     System.out.printf("%-30s%-35s%-30s%-30s%-15s%-15s%-10s%n","Make",
605             "Variant","Length","Region","SellPrice","RentPrice","Year");
606     AtomicBoolean foundMatchingEntries = new AtomicBoolean( initialValue: false);
607     //why use atomic boolean? because we need to use it in the lambda expression
608     // and boolean is forbidden in lambda expression
609     Set<Map.Entry<Double, ArrayList<Boat>> > entries
610             = sPriceBoats.entrySet();
611     entries.forEach(entry -> {
612         if (entry.getKey()>=minPrice&&entry.getKey()<=maxPrice){
613             foundMatchingEntries.set(true);
614             ArrayList<Boat> boat = entry.getValue();
615
616             for (Boat b : boat
617             ) {System.out.printf("%-30s%-35s%-30s%-30s%-15.2f%-15.2f%-10d%n",b.getMake(),
618                     b.getVarient(),b.getLength(),b.getRegion(),
619                     b.getPrice()[1],b.getPrice()[2],b.getYear());
620
621             }
622         }
623     });
624     if (!foundMatchingEntries.get()) {
625         throw new NotFoundByGivenInfo("No such boats by given information");
626     }
627 }
628
1 usage  Albert Zhao
629 public void showBoatsByLengthRange(double minLength, double maxLength) throws NotFoundByGivenInfo {
630     TreeMap<Integer, ArrayList<Boat>> lengthRange =  lengthBoats;
631     System.out.printf("%-30s%-35s%-30s%-30s%-15s%-15s%-10s%n","Make",
632             "Variant","Length","Region","SellPrice","RentPrice","Year");
633     AtomicBoolean foundMatchingEntries = new AtomicBoolean( initialValue: false);
634     //why use atomic boolean? because we need to use it in the lambda expression
635     // and boolean is forbidden in lambda expression
636     Set<Map.Entry<Integer, ArrayList<Boat>>> entries
637             = lengthRange.entrySet();
638     entries.forEach(entry -> {
639         if (entry.getKey()>=minLength&&entry.getKey()<=maxLength){
640             ArrayList<Boat> boat = entry.getValue();
641             foundMatchingEntries.set(true);
642             for (Boat b : boat
643             ) {
644                 System.out.printf("%-30s%-35s%-30s%-30s%-15.2f%-15.2f%-10d%n",b.getMake(),
645                         b.getVarient(),b.getLength(),b.getRegion(),
646                         b.getPrice()[1],b.getPrice()[2],b.getYear());
647             }
648         }
649     });
650     if (!foundMatchingEntries.get()) {
651         throw new NotFoundByGivenInfo("No such boats by given information");
652     }
653 }
654
1 usage  Albert Zhao
655 public void showBoatsByYearRange(double minYear, double maxYear) throws NotFoundByGivenInfo {
656     TreeMap<Integer,ArrayList<Boat>> boats = yearBoats;
657     System.out.printf("%-30s%-35s%-30s%-30s%-15s%-15s%-10s%n","Make",
658             "Variant","Length","Region","SellPrice","RentPrice","Year");
659     AtomicBoolean foundMatchingEntries = new AtomicBoolean( initialValue: false);
660     //why use atomic boolean? because we need to use it in the lambda expression
661     // and boolean is forbidden in lambda expression
662     Set<Map.Entry<Integer, ArrayList<Boat>> > entries
663             = boats.entrySet();
664     entries.forEach(entry -> {
665         if (entry.getKey()>=minYear&&entry.getKey()<=maxYear) {
666             ArrayList<Boat> boat = entry.getValue();
667             foundMatchingEntries.set(true);
668             for (Boat b : boat
669             ) {
670                 System.out.printf("%-30s%-35s%-30s%-30s%-15.2f%-15.2f%-10d%n",b.getMake(),
```

```
                        b.getVarient(),b.getLength(),b.getRegion(),
                        b.getPrice()[1],b.getPrice()[2],b.getYear());
                }

            }
        });
        if (!foundMatchingEntries.get()) {
            throw new NotFoundByGivenInfo("No such boats by given information");
        }

    }
    /**
     * This method is to show the boats by make and find the price that is less than the condition
     *
     */
    1 usage  Albert Zhao
    public void showBoatsBysPriceAndMake(double price, String make) {
        ArrayList<Boat> boats = byAttributeBoats.get(make);
        ArrayList<Boat> result = new ArrayList<>();
        for (Boat boat : boats
        ) {
            if (boat.getPrice()[1] <= price) {
                result.add(boat);
            }
        }
        System.out.printf("%-30s%-35s%-30s%-30s%-15s%-15s%-10s%n","Make",
                "Variant","Length","Region","SellPrice","RentPrice","Year");
        for (Boat boat : result
        ) {
            System.out.printf("%-30s%-35s%-30s%-30s%-15.2f%-15.2f%-10d%n",boat.getMake(),
                    boat.getVarient(),boat.getLength(),boat.getRegion(),
                    boat.getPrice()[1],boat.getPrice()[2],boat.getYear());
        }

    }//end of the blocks of methods to show the boats by different attributes.

    }//end of the blocks of methods to show the boats by different attributes.


    1 usage  Albert Zhao
    public static ArrayList<Boat> loadBoatsFromFile() {
        String filePath = "resources/createdFiles/allboats";

        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(filePath))) {
            return (ArrayList<Boat>) ois.readObject();
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
            return null;
        }
    }

    1 usage  Albert Zhao
    public ArrayList<Boat> getBoats() { return boats; }

    no usages  Albert Zhao
    public HashMap<String, ArrayList<Boat>> getByAttributeBoats() { return byAttributeBoats; }

    no usages  Albert Zhao
    public void setByAttributeBoats(HashMap<String, ArrayList<Boat>> byAttributeBoats) {
        this.byAttributeBoats = byAttributeBoats;
    }

    no usages  Albert Zhao
    public static HashMap<Boat, Boat> getAllBoats() { return allBoats; }

    no usages  Albert Zhao
    public static void setAllBoats(HashMap<Boat, Boat> allBoats) { Database.allBoats = allBoats; }

    no usages  Albert Zhao
    public TreeMap<Double, ArrayList<Boat>> getrPriceBoats() { return rPriceBoats; }
```

```java
package backend;

import boat.Boat;
import exceptions.FailedTransactionException;
import exceptions.InvalidEmailAddress;
import exceptions.NotFoundByGivenInfo;
import person.Client;
import person.Company;

import javax.mail.MessagingException;
import java.io.*;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.InputMismatchException;
import java.util.Map;
import java.util.Scanner;

public class Menu implements Serializable {
    private static Scanner scanner = new Scanner(System.in);
    // !!!!!!!!!!!!!!!!! To read the database
    static RegistrationSystem system = new RegistrationSystem();
    static ArrayList<Boat> boats = loadBoatsFromFile();
    static Database database = new Database(boats);
    static CompanyEmailSender companyEmailSender;


    public static void main(String[] args) throws ClassNotFoundException, N
        //Initializing
        System.out.println("We are sending your information to our server,


        LocalTime constructedTime = LocalTime.now();


        try {
            companyEmailSender = new CompanyEmailSender();
            companyEmailSender.sent("System is working at " + getCurrentTim
```

```
    companyEmailSender.sent("System is working at " + getCurrentTime(), new
} catch (MessagingException e) {
    e.printStackTrace();
} catch (InvalidEmailAddress e) {
    System.out.println("Something badly happen in our server, email notifica
    ;
}



//first, client log in their accounts
System.out.println("Welcome to our company!");
System.out.println("Time: " + getCurrentTime());

Boolean flag = true;
a:
do {

    System.out.println("You can choose 1. register  2. login  3. quit ");
    try{
        int log = scanner.nextInt();
        switch (log) {
            case 1:
                flag = true;
                // Use the register method in the RegistrationSystem class t
                Map<String, Client> userDatabase = system.registerUser();
                system.saveUserDatabaseToFile(userDatabase);
                break;
            case 2:
                flag = true;
                system.loadUserDatabaseFromFile();
                login();
                saveBoatsToFile(boats);
                System.out.println("................................");
                break a;
            case 3:
                flag = true;
                System.out.println("Thank you for your visit!");
                flag = false;
                return;

        }
```

```
        }catch (InputMismatchException e){
            System.out.println("Wrong input, please enter 1 or 2 or 3");
            scanner.next();
        }
    }while (flag);
}

public static String getCurrentTime() {
    return java.time.LocalDateTime.now().toString();
}

public static void login() throws exceptions.NotFoundByGivenInfo, FailedT

    // to log in;
    System.out.print("Please enter your user name:");
    String name = scanner.next();
    scanner.nextLine();

    Map<String, Client> userDatabase = system.loadUserDatabaseFromFile();
    // Check if the username exists
    if (!userDatabase.containsKey(name)) {
        System.out.println("Username not found. Please register first.");
        return;
    }
    System.out.print("Enter your password: ");
    String enteredPassword = scanner.nextLine();
    System.out.println("\n\nEnter any key to continue...");
    scanner.nextLine();
    // Get the user info for the entered username

    String storedPassword = userDatabase.get(name).getPassword();


    // Check if the entered password matches the stored password
    while (true) {
        if (enteredPassword.equals(storedPassword)) {
            System.out.println("Login successfully!");
            System.out.println("We are sending your login information to y
```

```
            try {
                if (companyEmailSender.isValidEmail(userDatabase.get(name).getEmail()) == -1)
                    System.out.println("Invalid email address, so we cannot send email to you
                }
                companyEmailSender.sent("Someone is trying to log in your account at \n" + ge
            } catch (InvalidEmailAddress e) {
                System.out.println("Invalid email address, so we cannot send email to you ple
            }
            System.out.println("Login Information has sent to your email");
            System.out.println("----------------------------------------");

            // if login successful, then start to choose function
            fiveRequest(name);
            break;
            // Determine which module to enter based on the last input int
            //           scanner.close();
        } else {
            // If the passwords don't match, display an error message
            System.out.println("Invalid password. Please try again.");
            enteredPassword = scanner.nextLine();
        }
    }
}

// display basic function: choose borrow, buy or sell;
public static void fiveRequest(String name) throws exceptions.NotFoundByGivenInfo,
FailedTransactionException, InvalidEmailAddress {
    int input1;

    while (true) {

        System.out.println("What is your request?");
        System.out.println("1. borrow  2. buy  3. return  4. search appointment  5. display l
        input1 = scanner.nextInt();
        switch (input1) {
            case 1:
                borrowBoat(name);
                break;
            case 2:
                buyBoat(name);
                break;
            case 3:
                returnBoat(name);
                break;
            case 4:
                searchAppointment();
                break;
            case 5:
                displayLog(name);
                break;
            case 6:
```

```
            default:
                System.out.println("Invalid input.");
                break;
        }
        if (input1 == 6) {
            break;
        }
    }
    scanner.nextLine();
}

public static void borrowBoat(String name) throws exceptions.NotFoundByGivenInfo, InvalidEmailAd
    Map<String, Client> userDatabase = system.loadUserDatabaseFromFile();
    Client currentClient = userDatabase.remove(name);
    try {
        Boat boat = recommendBoat();
        borrowTransaction(name, boat);
    }catch (FailedTransactionException e) {
        companyEmailSender.sent("Sorry, the dealing is failed, please try again later\n\n"+e.get
    }
}

public static void buyBoat(String name) throws exceptions.NotFoundByGivenInfo, FailedTransaction
    Map<String, Client> userDatabase = system.loadUserDatabaseFromFile();
    Client currentClient = userDatabase.remove(name);
    try {
        Boat boat = recommendBoat();
        buyTransaction(name, boat);
    }catch (FailedTransactionException e) {
        companyEmailSender.sent("Sorry, the dealing is failed, please try again later\n\n"+e.get
    }
}

public static void returnBoat(String name) {
    Map<String, Client> userDatabase = system.loadUserDatabaseFromFile();
    Client currentClient = userDatabase.get(name);
    System.out.println("user:" + currentClient.getName());
    if (currentClient.getUse().size() == 0 || currentClient.getUse() == null) {
        System.out.println("no boat can return");
    } else {
        int i;
        for (i = 0; i < currentClient.getUse().size(); i++) {
            System.out.println("Boat" + (i+1) + " " + currentClient.getUse().get(i));
        }
        System.out.print("Which boat do you want to return? input No.x");
        int deleted = scanner.nextInt();
        Boat deletedBoat = currentClient.getUse().get(deleted-1);
        currentClient.getUse().remove(deletedBoat);
        deletedBoat.setUser(new Company());

        saveBoatsToFile(boats);
        system.saveUserDatabaseToFile(userDatabase);
```

```
        saveBoatsToFile(boats);
        system.saveUserDatabaseToFile(userDatabase);
    }

}

public static void searchAppointment() {

}

public static void displayLog(String name) {
    Map<String, Client> userDatabase = system.loadUserDatabaseFromFile();
    Client currentClient = userDatabase.get(name);
}

public static Boat recommendBoat() throws exceptions.NotFoundByGivenInfo, FailedTransactionExcept
    System.out.println("Are there any requirements for the boat? ");
    System.out.println("You can select your boat's make, model, length, current docking area and
    System.out.print("y/n : ");
    String input2;
    Boat borrowBoat = new Boat();
    while (true) {
        try {
            input2 = scanner.next();
            if (input2.equals("y") || input2.equals("n")) {
                break;
            }
            System.out.println("Please enter y or n ~");
        } catch (Exception e) {
            scanner.nextLine();
            System.out.println("Please enter y or n ~");
        }
    }
    if (input2.equals("y")){
        database.show(scanner);
        Boat currentBoat;
        try{
            currentBoat = Database.searchBoat(scanner);
        }
        catch (NotFoundByGivenInfo e){
            System.out.println("Sorry, we cannot find the boat you want, please try again later"
            return borrowBoat;
        }
        scanner.nextLine();

        return currentBoat;
    }
    if (input2.equals("n")) {
        int i = 0;
        System.out.println("I will recommend any boat to you. The following is information about
        for (Map.Entry<Double, ArrayList<Boat>> entry : database.rPriceBoats.subMap(480000.0,500
            System.out.println("---> Price: " + entry.getKey());
```

```
                i++;
                System.out.println("boat " + i + " : " + boat);
            }
            System.out.println();
        }
        System.out.print("Choose one: ");
        int count = scanner.nextInt();
        int j = 0;
        b:
        for (Map.Entry<Double, ArrayList<Boat>> entry : database.rPriceBoats.subMap(480000.0,50
            for (Boat boat : entry.getValue()) {
                j++;
                if(j == count)
                    return boat;
            }
        }
    }
    return borrowBoat;
}

public static void borrowTransaction(String name, Boat boat) throws FailedTransactionException
    System.out.println("Dealing.....\nPlease Wait a moment");
    Map<String, Client> userDatabase = system.loadUserDatabaseFromFile();
    Client currentClient = userDatabase.get(name);
    if (!(boat.getOwner() instanceof Company))
        throw new FailedTransactionException("The boat is sold out");
    else if (boat.getUser() instanceof Client)
        throw new FailedTransactionException("The boat is rented");
    boat.setUser(currentClient);
    //Boat oldBoat=boats.get(boat.getIndex());
    currentClient.getUse().add(boat);
    System.out.println("Now, you can use: " + currentClient.getUse());
    userDatabase.put(name, currentClient);
    system.saveUserDatabaseToFile(userDatabase);
}
public static void buyTransaction(String name, Boat boat) throws FailedTransactionException {
    System.out.println("Dealing.....\nPlease Wait a moment");
    Map<String, Client> userDatabase = system.loadUserDatabaseFromFile();
    Client currentClient = userDatabase.get(name);
    if (!(boat.getOwner() instanceof Company))
        throw new FailedTransactionException("The boat is sold out");
    else if (boat.getUser() instanceof Client)
        throw new FailedTransactionException("The boat is rented");
    boat.setUser(currentClient);
    boat.setOwner(currentClient);
    //saveBoatsToFile(boats);
    ArrayList<Boat> set = currentClient.getOwn();
    set.add(boat);
    System.out.println("Now, you own : " + currentClient.getOwn());
    userDatabase.put(name, currentClient);
    system.saveUserDatabaseToFile(userDatabase);

    //currentClient.getTransaction().add(boat);
```

```
    system.saveUserDatabaseToFile(userDatabase);

    //currentClient.getTransaction().add(boat);
}

public static ArrayList<Boat> loadBoatsFromFile() {
    String filePath = "resources/createdFiles/allboats";

    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(filePath))) {
        return (ArrayList<Boat>) ois.readObject();
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
        return null;
    }
}

public static void saveBoatsToFile(ArrayList<Boat> boats) {
    String filePath = "resources/createdFiles/allboats";
    try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(filePath))) {
        oos.writeObject(boats);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

RegistrationSystem.java

> Users > larei > Desktop > CPS2232_FinalProject-main > src > backend > J RegistrationSystem.java

```java
package backend;

import boat.Boat;
import person.Client;

import java.io.*;
import java.util.*;

public class RegistrationSystem implements Serializable {
    // This class describes the registration system of the WKU Boat Company.

    // Build a database to store the information of the clients.
    private String FILE_PATH1 = "resources/createdFiles/clients.csv";
    private String FILE_PATH2 = "resources/createdFiles/userData";
    private Map<String, Client> userDatabase = loadUserDatabaseFromFile();
    private Scanner input = new Scanner(System.in);
    private ArrayList<Boat> own = new ArrayList<>();

    public RegistrationSystem() {
    }

    public Map<String, Client> getUserDatabase() {
        return userDatabase;
    }

    private ArrayList<Boat> use = new ArrayList<>();

    public void saveUserDatabaseToFile(Map<String, Client> userDatabase) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(FILE_P
            oos.writeObject(userDatabase);
            System.out.println("User database has been saved ");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public Map<String, Client> loadUserDatabaseFromFile() {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_PATH
            return (Map<String, Client>) ois.readObject();
        } catch (IOException | ClassNotFoundException | NullPointerException exception)
            return new HashMap<>();
        }
    }

    // The method to register a new user.
    public Map<String, Client> registerUser() {
        String name = getInputName();

    public ArrayList<Boat> getUse() {
        return use;
    }

    public void setUse(ArrayList<Boat> use) {
        this.use = use;
    }

    public ArrayList<Boat> getOwn() {
        return own;
    }

    public void setOwn(ArrayList<Boat> own) {
        this.own = own;
    }
}
```

Boat.java ●

C: > Users > larei > Desktop > CPS2232_FinalProject-main > src > boat > Boat.java > Boat

```java
package boat;

import interfaces.Asset;
import interfaces.Person;
import person.Company;

import java.io.Serializable;

public class Boat implements Asset,Comparable<Boat>, Serializable {
    //
    private String make;
    private String variant;
    private int length;
    private String region;
    private double costPrice;
    private int year;
    private double[] prices = new double[3];
    private Person owner = new Company(0,"CPS","2232");
    private Person user = new Company(0,"CPS","2232");
    private int index;


    public Boat(String make, String variant, int length, String region,
    double costPrice, double sellPrice, double rentPrice,
            int year,int index) {
        this.make = make;
        this.variant = variant;
        this.length = length;
        this.region = region;
        this.year = year;
        this.index=index;
        prices[0] = costPrice;
        prices[1] = sellPrice;
        prices[2] = rentPrice;

    }


    public String getMake() {
        return make;
    }

    public String getVarient() {
        return variant;
    }
```

```
public String getMake() {
    return make;
}

public String getVarient() {
    return variant;
}

/**
 * @return
 */
@Override
public double[] getPrice() {
    return prices;
}

/**
 *
 */
@Override
public void setPrice(int rentPrice,int sellPrice,int costPrice) {

    this.costPrice = costPrice;

    prices[0] = rentPrice;
    prices[1] = sellPrice;
    prices[2] = costPrice;
}

/**
 * @return
 */
@Override
public Person getOwner() {
    return owner;
}

/**
 * @param owner
 */
@Override
public void setOwner(Person owner) {
    this.owner = owner;

}

/**
```

```java
/**
 * @return
 */
@Override
public Person getUser() {
    return user;
}

/**
 * @param user
 */
@Override
public void setUser(Person user) {
    this.user = user;
}

public String getMaker() {
    return make;
}

public void setMaker(String maker) {
    this.make = maker;
}

public int getLength() {
    return length;
}

public void setLength(int length) {
    this.length = length;
}

public String getRegion() {
    return region;
}

public void setRegion(String region) {
    this.region = region;
}

public double getCostPrice() {
    return prices[0];
}

public void setCostPrice(int costPrice) {
    this.costPrice = costPrice;
}
```

```
public void setCostPrice(int costPrice) {
    this.costPrice = costPrice;
}



public int getYear() {
    return year;
}

public void setYear(int year) {
    this.year = year;
}


/**
 * @param o the object to be compared.
 * @return
 */
@Override
public int compareTo(Boat o) {
    return (int) (this.getPrice()[1] - o.getPrice()[1]);
}

@Override
  public int hashCode() {
    return (int) (this.getMake().hashCode() + this.getVarient().hashCode() + this.getLength()
                + this.getYear() + this.getPrice()[1]);
}*/

public boolean equals(Object obj) {
    if (obj instanceof Boat) {
        Boat boat = (Boat) obj;
        return this.getMake().equals(boat.getMake()) && this.getVarient().equals(boat.getVarie
                && this.getLength() == boat.getLength() && this.getRegion().equals(boat.getReg
                && this.getYear() == boat.getYear() && this.getPrice()[1] == boat.getPrice()[1
    }
    return false;
}

public double getSellPrice() {
    return costPrice * (30 + year * 20) / 30000;
}

@Override
public String toString() {
    return "Make: " + make + " Variant: " + variant + " Sell Price: " + prices[1]+ "  Rent Pri
}

public Boat(){};

public int getIndex() {
    return index;
```

```java
package boat;

import java.io.*;
import java.util.ArrayList;
import java.util.List;
  //Create boat array

public class BoatCreator implements Serializable {
    public static void main(String[] args) {
        String csvFile = "resources/2023_MCM_Problem_Y_Boats.csv";
        String line;
        String csvSplitBy = ",";
        List<Boat> boats = new ArrayList<>();

        try (BufferedReader br = new BufferedReader(new FileReader(csvFile))) {
            br.readLine();
            br.readLine();

            while ((line = br.readLine()) != null) {
                line = line.replace("\"", "");
                String[] data = line.split(csvSplitBy);
                String make = data[0];
                String variant = data[1];
                int length = Integer.parseInt(data[2]);
                String region = data[3];
                int year = Integer.parseInt(data[6]);
                String price = data[5].replaceAll("[^\\d.]", ""); // Remove non-numeric characte
                double costPrice = Double.parseDouble(price) * 100 ;
                double sellPrice = costPrice * (4-( 2030d - year )/10) ;
                double rentPrice = Double.parseDouble(price) ;
                // Create a new Boat object using the constructor
                Boat boat = new Boat(make, variant, length, region, costPrice, sellPrice, rentPr
                boats.add(boat);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        saveBoatsToFile(boats);
        System.out.println("Number of boats created: " + boats.size());
        System.out.println(boats.get(1000).getMake());
        System.out.println(boats.get(100).getYear());
        System.out.println(boats);
    }

    private static void saveBoatsToFile(List<Boat> boats) {
        String filePath = "resources/createdFiles/allboats";

        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(filePath))) {
            oos.writeObject(boats);
            System.out.println("Boats list has been saved to file: " + filePath);
        } catch (IOException e) {
            e.printStackTrace();
        }
```

```
        saveBoatsToFile(boats);
        System.out.println("Number of boats created: " + boats.size());
        System.out.println(boats.get(1000).getMake());
        System.out.println(boats.get(100).getYear());
        System.out.println(boats);
    }

    private static void saveBoatsToFile(List<Boat> boats) {
        String filePath = "resources/createdFiles/allboats";

        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(filePath))) {
            oos.writeObject(boats);
            System.out.println("Boats list has been saved to file: " + filePath);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

FailedTransactionException.java

C: > Users > larei > Desktop > CPS2232_FinalProject-main > src > exception

```
1   package exceptions;
2
3   public class FailedTransactionException extends Exception{
4       private String message;
5       public FailedTransactionException(String message) {
6           super(message);
7           this.message = message;
8       }
9   }
10
```

FailedTransactionException.java 1 | InvalidArgumentException.java

C: > Users > larei > Desktop > CPS2232_FinalProject-main > src > exceptions >

```
1   package exceptions;
2   public class InvalidArgumentException extends Exception {
3       public InvalidArgumentException(String message) {
4           super(message);
5       }
6   }
7
8
```

```java
package exceptions;

public class InvalidEmailAddress extends Exception {
    public InvalidEmailAddress(String message) {
        super(message);
    }
}
```

NotFoundByGivenInfo.java

C: > Users > larei > Desktop > CPS2232_FinalProject-main > src > exceptio

```java
package exceptions;

public class NotFoundByGivenInfo extends Exception{
    public NotFoundByGivenInfo(String message) {
        super(message);
    }
}

```

Person.java

C: > Users > larei > Desktop > CPS2232_FinalProject-main > src > interfaces > Person.java > {} i

```java
package interfaces;
/*
    * This interface is used to define the methods that all clients must have.
    * A client is a person who can borrow or buy a boat.
    *
 */


import java.io.Serializable;

public interface Person extends Serializable {
     String getName();
     void setName();
     String getPassword();
     void setPassword();
     String getEmail();

}
```

Asset.java ●

> Users > larei > Desktop > CPS2232_FinalProject-main > src > interfaces > J Asset.jav

```java
package interfaces;

/*
 * This interface is used to define the methods that all assets must have
 * This interface is implemented by the Boat class.
 *
 * An asset is anything that can be bought, sold or borrowed.
 * its price, owner and user can be modified and got.
 *
*/

import java.io.Serializable;

public interface Asset extends Serializable {
    //return an array to store the information of the
    //asset,arr[0] is the rent price, arr[1] is the sell price,
    // arr[2] is the cost price,
    double[] getPrice();
    void setPrice(int rentPrice,int sellPrice,int costPrice);
    Person getOwner();
    void setOwner(Person owner);
    Person getUser();
    void setUser(Person user);


}

```

Client.java 2 ×

> Users > larei > Desktop > CPS2232_FinalProject-main > src > person > J Client.java > { } person

```java
package person;


import boat.Boat;
import interfaces.Person;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.HashSet;

public class Client implements Person, Serializable {

    //  This class describes the attributes and operations of clients.
    private String name;
    private String password;
    private String email;

    private ArrayList transaction;    //  The credit of the client depends on the money tha
    private String log;

    //  The appointment is the set of boats that the client has booked.
    private HashSet<Boat> appointment = new HashSet<Boat>();
    //  The own is the set of boats that the client has bought.
    private ArrayList<Boat> own ;
    private ArrayList<Boat> use ;

    /*    Comparator<Client> comparator = new Comparator<Client>() {
          @Override
          public int compare(Client c1, Client c2) {
              return c1.credit().compareTo(c2.credit());
          }
      };*/
    private String uniqueID;

    public Client(String name, String password, String email, ArrayList<Boat> use, ArrayLis
        this.name = name;
        this.password = password;
        this.email = email;
        this.use = use;
        this.own = own;
    }

    public String getUniqueID() {
        return uniqueID;
    }

    public String getName() {
        return name;
    }
```

```java
public String getEmail() {
    return email;
}

/**
 *
 */
@Override
public void setName() {
    this.name = name;
}

public String getPassword() {
    return password;
}

/**
 *
 */
@Override
public void setPassword() {
    this.password = password;
}

@Override
public String toString() {
    return "Client{" +
            ", name='" + name + '\'' +
            ", password=" + password +
            '}';
}

public ArrayList getTransaction() {
    return transaction;
}

public void setTransaction(ArrayList transaction) {
    this.transaction = transaction;
}

public ArrayList<Boat> getOwn() {
    if(own == null){
        own = new ArrayList<>();
    }
    return own;
}
```

```
    public String toString() {
        return "Client{" +
                ", name='" + name + '\'' +
                ", password=" + password +
                '}';
    }

    public ArrayList getTransaction() {
        return transaction;
    }

    public void setTransaction(ArrayList transaction) {
        this.transaction = transaction;
    }

    public ArrayList<Boat> getOwn() {
        if(own == null){
            own = new ArrayList<>();
        }
        return own;
    }

    public void setOwn(ArrayList<Boat> own) {
        this.own = own;
    }

    public ArrayList<Boat> getUse() {
        if(use == null){
            use = new ArrayList<>();
        }
        return use;
    }


    public void setUse(ArrayList<Boat> use) {
        this.use = use;
    }
}
```

Client.java 2 | Company.java 2

C: > Users > larei > Desktop > CPS2232_FinalProject-main > src > person > Company.java > { } person

```java
package person;
import java.io.Serializable;
import interfaces.Person;

public class Company implements Person, Serializable {
    private String name;
    private String password;
    String email = "zhaoq@kean.edu";

    public Company(int priority, String name, String password) {
        this.
        this.password = password;
    }


    public String getName() {
        return name;
    }

    /**
     *
     */
    @Override
    public void setName() {
        this.name = name;
    }

    public String getEmail() {

        return email;
    }

    public String getPassword() {
        return password;
    }

    /**
     *
     */
    @Override
    public void setPassword() {
        this.password = password;
    }

    @Override
    public String toString() {
        return "Sailor{" +
                ", name='" + name + '\'' +
                ", password=" + password +
                '}';
    }
}

```

String password

Reports and Analysis

a) Discussions

1. Data structure selection

In this project, we used three different data structures: TreeMap, PriorityQueue, ArrayList, HashMap. Each data structure plays a role in different scenarios. The following is an analysis of their space complexity and time complexity, as well as their advantages and disadvantages.

a. TreeMap

Insertion operation: O(log n)

Search operation: O(log n)

Delete operation: O(log n)

advantage:

Ordered storage: TreeMap is implemented based on red-black trees and maintains the orderliness of keys. In this project, we use price, length and other values as keys to show users our company's ship prices from high to low. The values stored in TreeMap do not need to be sorted by yourself, which is very convenient. Provides various useful methods, such as subMap and headMap, to facilitate range queries. In the recommendation link, we recommend affordable boats to customers, that is, using the subMap method.

shortcoming:

Compared with other data structures, insertion and deletion operations are slightly time-consuming. The space overhead is large.

b. PriorityQueue

Insertion operation: O(log n)

Delete operation (get the minimum value): O(1)

advantage:

High-priority content can be output first. In the output log, we use PriorityQueue to output exception information first, and then output transaction success information.

shortcoming:

Does not support regular search and delete operations and is not ordered storage.

c. ArrayList

Insertion operation (insert at the end): O(1)

Search operation: O(n)

Delete operation: O(n)

advantage:

Supports fast random access. We can use ArrayList to quickly traverse the various attributes of the ship we have stored, and directly call out the ship to operate the ship.

shortcoming:

The performance of insert and delete operations varies with location and can be slow at worst.

2. Exception handling

In the project, we focused on handling some abnormal situations, including transaction failure and email sending failure. Through meticulous exception handling, the stability of the system and user experience are improved.

3. Optimization of real-life problems

a. The user entered incorrect content

By implementing an effective input validation mechanism, we can prevent users from entering incorrect content. This includes format validation, range checking, etc. to ensure that the data received by the system is valid and as expected.

b. Fast reading and operation of large amounts of data

By using binary file format

, we successfully optimized the reading and manipulation of large amounts of data. The format of binary files is more compact than text files, reducing storage and transmission overhead, and improving reading and writing speed. We directly write the data structure into the file as an object, which is more convenient.

4. Integration of email sending function

The email sending function was introduced in the project, which provides an effective way for timely

notification and alarm. By integrating the email function, we can respond to events that occur in the system in a timely manner, improving the real-time and monitorability of the system.

5. Introduction of company operations

In order to meet the company's internal needs, we introduced the company's operation terminal, through which the system can be more conveniently managed and monitored. This further improves the management efficiency and maintainability of the system.

6. Summary and outlook

Through reasonable selection of different data structures and optimization of real-world problems, we successfully designed a robust and efficient system. However, as an ongoing development project, we will continue to pay attention to user feedback, technical developments and business needs, and continuously improve and expand the functionality of the system to meet future challenges and opportunities.

IV. Conclusions

The boat management system that will be developed for the "Wenzhou Ken University Fisher Boat Management Company" is expected to address the challenges faced by local fishermen in Wenzhou. The system will provide comprehensive functions for both company employees and customers, streamlining the boat management process and enhancing collaboration among fishermen.

Through the implementation of the boat management system, several key achievements are anticipated:

1. Efficient Collaboration: The system will allow for the collective utilization of boats, enabling efficient collaboration among fishermen. By recording and managing boat information, fishermen will be able to easily access available boats and coordinate their operations effectively. This is expected to lead to improved productivity and resource utilization within the fishing industry in Wenzhou.

2. Boost to Tourism Industry: The boat management system will also have the potential to boost the tourism industry in Wenzhou. By leasing idle boats to tourists, the system will facilitate recreational activities such as boat tours and fishing trips. This will not only provide additional income opportunities for local fishermen but also attract tourists to explore the natural beauty of Wenzhou's waters.

3. Enhanced User Experience: For customers, the boat management system will offer a user-friendly interface and convenient features. Users will be able to easily register an account, browse available boats based on various attributes, and make purchases or lease agreements

with just a few clicks. The system will also keep track of transaction records, ensuring transparency and accountability.

In terms of technical implementation, the development team will overcome challenges related to boat storage, lookup optimization, and user recommendation system design. By adopting an `ArrayList` data structure for efficient boat storage and leveraging pre-sorted data using `TreeMap` sorters, the system will achieve fast retrieval speed and optimize lookup operations. Furthermore, the team will utilize the `computeIfAbsent` function to create various filtering methods, improving the efficiency of the filtering process.

Although there will be areas for further refinement, such as the user recommendation system design, it is expected that the boat management system will provide a valuable solution for the "Wenzhou Ken University Fisher Boat Management Company" and have the potential to bring significant benefits to the fishing industry and tourism sector in Wenzhou. The successful implementation of this project will showcase the importance of leveraging technology to optimize resource utilization and enhance collaboration within industries.

References

Bajracharya, D., & Kathmandu, N. A REVIEW ON JAVA HASHMAP AND TREEMA

Gil, J., & Shimron, Y. (2011, October). Smaller footprint for java collections. In Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion (pp. 191-192).

Jacobson, N., & Thornton, A. (2004). It is time to emphasize arraylists over arrays in Java-based first programming courses. In Working group reports from ITiCSE on Innovation and technology in computer science education (pp. 88-92).

Long, F., Mohindra, D., Seacord, R., & Svoboda, D. (2010). Java Concurrency Guidelines.

M. Zheng, J. Yang, M. Wen, H. Zhu, Y. Liu and H. Jin, "Why Do Developers Remove Lambda Expressions in Java?," 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 2021, pp. 67-78, doi: 10.1109/ASE51524.2021.9678600.

Muscolino, M. S. (2009). Fishing Wars II: The Cuttlefish Feud, 1932–1934. In Fishing Wars and Environmental Change in Late Imperial and Modern China (pp. 127-150). Harvard University Asia Center.

Reges, S., & Stepp, M. (2014). Building Java Programs. Pearson.

Wang, X. (2021). Maritime supervision over yacht in Wenzhou.

Zhang, W., & Wen, Y. (2023). A Study on the Sense of Relative Deprivation of Elderly Fishers from the Perspective of Ocean Sociology: The Example of Island S and G in Zhoushan City, China. In Maritime Professions (pp. 225-238). Brill.

Appendix:

Contribution of Group Members

Student Name:Zhao qinjian Student Number: 1235624

Main Contribution:

1. Find and conceive of suitable data structures to store data
2. Mainly responsible for the realization of the Database class. Build various methods based on data structures such as "PriorityQueue" and "TreeMap". Greatly improved the speed of user accessing the data.
3. Realize the function of sending emails, logging in to the company end and displaying log.
4. Test code.
5. Report writing

Student Name: Jiao Luyao Student Number: 1235723

Main Contribution:

1. Find and conceive of suitable data structures to store data
2. Mainly responsible for the realization and design of class Menu. Implement user-oriented methods such as buying and borrowing boats. Designed the logic loops of menu operations.
3. Greatly improved the login system.
4. Responsible for searching for data, importing data into files, and exporting data from files. Greatly improved the process of inputting and storing the data.
5. Test code. Improve the robustness of the code.
6. Report writing

Student Name: Zhang Lei Student Number:1235752

Main Contribution:

1. Literature review and find reference papers.
2. Choosing topic, conduct topic availability analysis. Proposed the boat dataset.
3. UML
4. Class relationship and early code framework design.
5. Build old version boat class and early version of boat creator, early version of the login system.
6. Trials to use some other data structures.
7. Test code.
8. Report writing.