MARKING REPORT

**Group number: 3\_\_\_\_**

|  |  |
| --- | --- |
| Name | Student ID |
| 1. Ang Yong Jie | 2100669 |
| 2. Lim Zhuo Jie | 2100163 |
| 3. Tan Swee Yang | 1904180 |
| 4. Tan Kia Kheng | 2006466 |

**Marks breakdown**

Part A: Test Plan (10 marks)

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Max Mark** | **Marks Obtained** | **Remark/Comment** |
| Test objective, scope and test basis | 5 |  |  |
| Test condition, entry and exit criteria | 5 |  |  |

Part B: Test Design (20 marks)

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Max Mark** | **Marks Obtained** | **Remark/Comment** |
| Decision table | 5 |  |  |
| Appropriateness of test cases | 15 |  |  |

Part C: Java Program (application code and test code) (70 marks)

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Max Mark** | **Marks Obtained** | **Remark/Comment** |
| Setup jar file location to C:\jars | 2 |  |  |
| Source directories | 3 |  |  |
| Appropriate used of assertsXXX methods. | 10 |  |  |
| Using parameterised tests correctly | 10 |  |  |
| Invalid values are checked for in implemented code, and tests for invalid values are performed. | 10 |  |  |
| Use of mocks or stubs for testing. | 10 |  |  |
| Combining test cases into test suites | 5 |  |  |
| Setting up some tests so that test values are read from a text file instead of hardcoding into test code | 10 |  |  |
| Perform integration testing after unit tests have been completed | 10 |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A: | B: | C: | Total:  /100 | /20 |

Table of Contents

[Test Plan 3](#_Toc100952824)

[Test Case Design 3](#_Toc100952825)

[Assumption 4](#_Toc100952826)

[UML diagram 5](#_Toc100952827)

[Link for the UML diagram 6](#_Toc100952828)

[Application Code 6](#_Toc100952829)

[Git Hub Link 6](#_Toc100952830)

[Code in text 7](#_Toc100952831)

[Main 7](#_Toc100952832)

[Client.java 7](#_Toc100952833)

[DeliveryItemList.java 8](#_Toc100952834)

[DeliveryStaff.java 11](#_Toc100952835)

[Document.java 12](#_Toc100952836)

[Driver.java 13](#_Toc100952837)

[Email.java 17](#_Toc100952838)

[Guest.java 18](#_Toc100952839)

[Item.java 19](#_Toc100952840)

[ItemList.java 20](#_Toc100952841)

[Parcel.java 21](#_Toc100952842)

[User.java 23](#_Toc100952843)

[UserList.java 24](#_Toc100952844)

[Test 27](#_Toc100952845)

[ClienTest.java 27](#_Toc100952846)

[IntergrationTest.java 29](#_Toc100952847)

[TestDeliveryItemList.java 31](#_Toc100952848)

[TestDocument.java 34](#_Toc100952849)

[TestEmail.java 36](#_Toc100952850)

[TestItem.java 37](#_Toc100952851)

[TestParcel.java 38](#_Toc100952852)

[TestSuite.java 40](#_Toc100952853)

[testUserList.java 41](#_Toc100952854)

# Test Plan



# Test Case Design



# Assumption

The email class is unfinish

The distance is calculated from other source, so we just need to key in the distance

The user is not able to register in this program that will be handle by third party site

The user is not able to add item

There is no admin module in this program that will be handle by third party site

The delivery staff is randomly assigning with random method

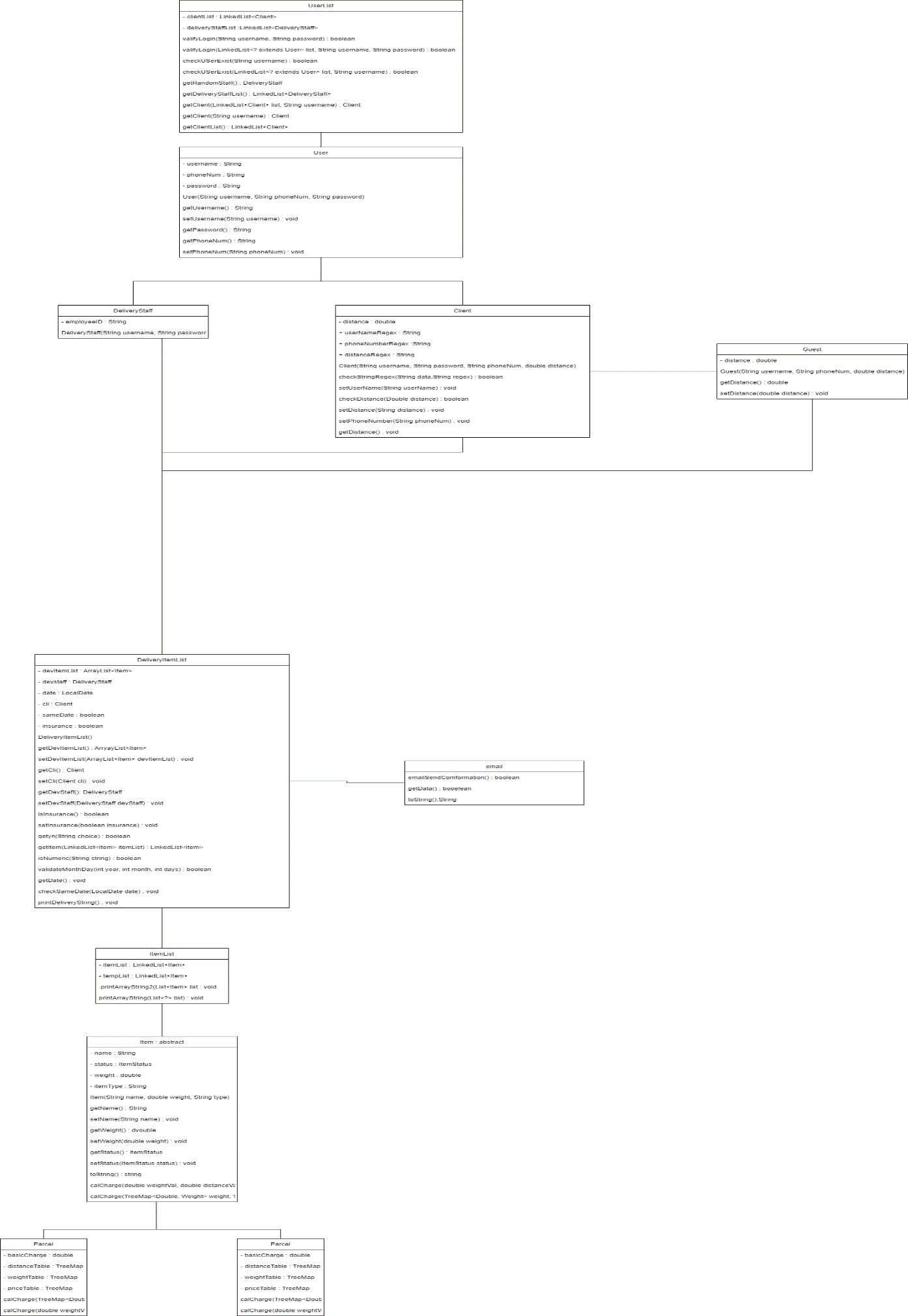
The limit of the date we set it to not less then today and not more than 3 months

The limit of the distance we set it to 0 to 100

The limit of weight we set it to 0 to 10000

The user is not able to set weight of the item they can only set distance of themselves

There is no way to add in item other then add it in the text file

UML diagram

# Link for the UML diagram

https://viewer.diagrams.net/?tags=%7B%7D&highlight=0000ff&edit=\_blank&layers=1&nav=1#R7Z1dk9o6EoZ%2FDVWzF7OFzfdlBpKc7E5mc0Lm5GTvNFiAN8YitskM%2BfUr27L8oQbMDJKdpKtSqbEwGKzHre63W1KnN908vQ3Idv2eOdTr2F3nqdObdWx7MLH4%2F3HDPm2wxt1J2rIKXEe05Q1z9wcVjV3RunMdGpZOjBjzIndbblww36eLqNRGgoA9lk9bMq981S1ZUaVhviCe2vrZdaJ12jq2R3n7H9RdrbMrW0Px%2BzYkO1n8knBNHPZYaOq97vSmAWNR%2BtfmaUq9%2BOZl9%2BXzu%2F1n7%2Fbr8O2%2F%2Fgy%2Fkfubf3%2B6%2B%2Bs6%2FbA357xF%2FoSA%2BtGzP%2FppfvfjtfdlNl09jbtk%2BbAZzL9dT4bpZ38n3k7csPuQBrduGIkfHe2zOxk%2BuhuP%2BPzoZsn8aC5e6fLjxdr1nFuyZ7v4%2B4URWXzNjm7WLHB%2F8POJx1%2ByeAN%2FOYgEKL1u6Yx5%2FE7xmQEN%2BTkfsh9tVZrek6fSibckjLJvwzyPbEP3Ifl%2B8Rs3JFi5%2Fg2LIrYRJ9W8p%2BLef6dBRJ8KRIl7%2FJayDY2CPT9FvDrpC1zEA3Pdy56Exxy%2F%2FkS0rQvo9YaikQjkV%2FLD5fU%2B8keE%2BCv%2Bw%2FIL9ioX7AMXHEPXq1yOeBENfBLRG7bznbAIE%2F%2Bj8FvzpgSxc3AbKbhdx8%2B%2B5%2FJeSKHjffOK%2F3%2Fr%2Bl%2BpkzYNvZijaXISP1rFR52p3XnVVRjlnRQliAXsK50yjwW83WcptK7nVZqI5658fujRZfy2uJddbjpeieaN6zjxJ9%2BEW7Jw%2FdVtctqsn7d8FLczbmL87UsvsRBr%2FkbKP%2BFmy1w%2FSu7v4Ib%2F41942v3noDPg33XKj638mP%2BLTw%2BiKfP51ydugiXlVD%2FSmOybgEUkIg%2FymVMBPv6EnyZ4X%2BbiJK9HcC2BczYlY5ASPjK5%2FFfsuZFYLnNYVFJmxRMRGO3ADBsHZqIAw%2F90l%2Ftbxs3%2B1TwK%2BJ3nr%2B%2FC2MBtaHwv7a5s3pIwfGSB8w9pfR64h0KJj7RooGXSNC1WzzqKi2pQOr03%2FHTe9zQZF7uxiyTtit31klNLSCFpLSDNshtHzVYt02JNF1%2Fv5zR4%2FcS5UY0TsmGEjUHjbGSffJiNS1oixMoIVuPmsVJD%2BhWNPhLfYZvEI77KUah4ygjExYGwa0f%2F%2BoAYQEDMqtFUAYu6IRUCowGYfvPA2BAwqf4CjEkVYabmKJS9CxG6PEKj5hHqHUEIqWhG1mueiv5hKk4OQVUzg4hcHpHGpV97NFF6ljormiWb%2BF1asxXzifc6b%2BU3aOc71BG3Jz%2FnlrFtJ80A%2FY9G0V4kncguYrxpHW2ylBSPpl7FucacD97yxo2%2F%2BzOSRSHbBQt67EGwxAgbkYA%2FADX6ML4HR3swoB6JuItWOOlIIon%2FWLIvnCBgzT%2F5Q9zQkTkmyxoJJ1KQ0q%2FkGs87n%2F%2BRfoOLJpfkXa0kMxWgMJF5FKUDicz6uu3z0pinLzesYYkMJDEtS3VurjtFzVeMYcLPwcRTvfEqf35%2FnlSlZakuzXXa49s177m73QZxMIZD44lIqwcVOKQ4iNQP4mAMhxZkGtVKhtgjOZ2RlrbjUPoQebk8Ly3IF2ZOUDlCvheUFOLjjAnkQAMHzecG%2B2qJQljg4IiC9p3xT0YsdGDRfG6vDyrzH8SwgObBCAdtSOmpZSUxB8JrQA7McNB8ps5SU7thgQPFn8RhwgAWjWff7NFQ6dpfUETvCxnupIguO%2FFyKvoLh3FVQDxeiYOK9dFuE6%2BOKjNhBkA8B4rIkyNP4mHFemiVL9cDHnyrW%2BPBN6FZ9yGdkj8Rm63H9pS%2Bm6HLcPbYIB%2Fjl%2BjVtbIalxsaMtQPmZ7akyUOKFYoTWkBB1C2DYMzGv0OPsXgZ%2FUp7PGgmf7hX%2F5v8f7k4Et8wJ8fcTh7Kr4428ujtvbrcNCqfpXfvGCwDxTJoZNYx0m0bOu0l9jvQ7b1efOzrX7%2FtJ%2FYltKGAewmOi6Hwl%2FkpQ0O28W9h2P9ybF%2BcAknEeZRm6w0UGWljn0jfMI77hN%2BpCv%2BWzBi0A8DVNJgGAZ1PlEKQ%2BbyP9CgwAPCoA8GqKDBMAxqfUsKQzZCoGUwBQNYrWCYBrW85dAUj%2FNUhLQ5czIkXCgu6CEJqncwTNKBefIpGYlRyZly%2BH3hN1IeB%2FHLOL%2FZDCtQEYRZVjKXSK2NuVNqY%2B6wNsYIFmBNhGEs1JKpxITMxOBxNQOGEzQZBtiA6iQMs2FDJkOSkY8sChloMrRhAdVJGMZC1VYL5TMPxUJtrKAxKYA1TwY4U1kaDKTAAAXN66BDVQd9u6O4NO5zUy%2FjzORn4qY9ULpYdnuxi2WEembqZTIsX7DNqZehqrNi6uVFNkc%2Bvi9JvYA8ajQ5qsCamJwzp4yhdGYGGCA9YxoYVYOFPRW0G%2FowABIzpjFQBdRihAvZA%2FRgNVMBZWjMYmGPLKVrf72qPbv2cjpSvW5JdZcNLFnzLqIb3H%2Fj%2BZMAJuWVh0Cdyb5gkDHplS9ojQEtfNSOIMMGl65JlqBwJXYHV2CLycT112qPGPZF1rMBUdU3YIDr2XDLSjdbpKNhOqBJAIbpAAoD09VteBQaJWu6pQGpfXUYDbE%2BLLqfmnEBghLTuKiSVhWUK2V1%2BxImGKbo5wQKUwyDMun9DmGK3a8Zpthtm1xkq15BatPFw0keYoYXGLM8M2bJwtIsZhkAIQSYqOg%2FM2axKxfsAjUaltWSoMVWvY7YJ4XW2sQB4uQAIZ%2Fln2jWsg3nxrhViHahZCA2SXPRhBxcnoPmJyHb8P6hj%2BnFMdOhnwEoqDDMALw7aKxffdpvcUQwQQEYMpjFANhhJLb%2FeZ48z5HL5FdmJ0qJ84hDg9lxPZgAE0tMYwIW8t3hYqrGGAAmjJhmAJwwUp4sgouomsEBmihi2oEAF8%2F8nFy7WC%2FzHd1IfRgAeoNpq2ArGIQSg4rLgJbBABJQnt4wEqBlSCWFgmVwUWfQrDc1TwK4%2FrogoagzCQ0KDYQBLJrXIYHtXSImEpk5AiGGE%2FogaF6EBPZT5v0yXcdZvarj8BfxDhTk81fU2uzDiTSE5xLwNK9eAruz5%2FB8Cih9T7Zy021BRczPZyFdyeqIopZ15H0zOX1IvjOfUQSIYyeATT4AmTXIbL95qXXUV7r2F6zOyCLS09UZWSc2vCnrqFsuPO51xch0aFfWSeZXH3iDnm1ZbWBPqw%2BEd4anUIXVIkdxytad71W7EfCJpPReNBFSfDuvWqS6rr49AC7Ylgp3YOesOD34QEJ3kQ6zygiGw9apYUs%2Bwi%2BpGQGR1DdqAZtsFOdTfyKp95KiID0oZOHyLNQ1TxpZgCvIUpcXSTBGAhB%2FmSYBriTbBi6aBIMgQAUkpklQ8z8tjMTr6UaIqAZEgeIV04iqxSuXUhqRGA3EAKUuhokZNbQLjGGpRqRhfhqpxpI7dbZdq1ETXKjVvGTJs2518xfdYo1lVa7YbrUGLuZGteZFI9eoNqbtUWvU0htUaxphoXm1JrsaqjXNktC8WjOAxXxUa8yC0AK1ZoBqDSLabrUG2KkT1ZoWE9O8WgPstkk3xMVg%2B5nBdq9SqGCNoPV5oWUt5Ah3ZrA9qgTbY2Bei%2FkFxpfeH%2F%2F59mV973bv3fW3279X%2F41G15bqSyWwzanvTNlmyYINiVzmX%2BG2OIr9URisY5LSp%2FucmBtE8xLmBwZCDbPilaP5zy8jQJEBXQzUNU%2FaGFBdlsJ8BtzMVWffQ2uG6%2Bp7%2BAtmDtAvnSySbt7pxaEzB77hZNGkskF9tljXoVSR1e8dfYOmVNH4d6gKP4OebH34ttEzOk6P3a2sX115gy56RkhP8cx%2BO23PcHicnhPn64FH3taC2zKjHr8pwR5Xz3%2FZ3IJJuT%2Btft1dGuU7z4ugx4PyBaEAWgbnTW%2FRNYbTEQ79%2Fq66en6CPS6P%2FvztVuQz%2FpIktuH9AsdqkkLwwa3GcinhyIzVPG1FGC4PAxBZm4bhwPwTbqIkCLeM99QsaUEILg8BEGKbhgCeeLLwXMnA1HPjH44AaAAASlybJuDA0sVkQ2dFU4Bau04OgOywaQ7gpYtdP9wFpf1dEQSdIABJX7Mg2F01iKgGr1e4Hq2W3ocWnzTd%2B3DSLY8gC7k3HkPuMYg0jAigP5hGRI0iwzIiJ8SFoiCBC9MZYKauRqaRGTXY5GaFhxYFc4KBhj4EoCUrTSOgFpOFKQJZx8dhJ9oDAzA0rkTaXVV3SN2MRHJMajsUHTLZyBGJ0EJE43Kk3VV1iLBAREWSjl2I5E%2B0FwboaFyntLuqOuGG7zJtAktCzYAArfVoGASgSpibiZwE2f%2B5coUmwgAZjQuYNlwuvPfzPVQWa%2BYWcUBboZOIFiiZ4HYJyS5cp7aHT%2FZQKOsTuKO8eYigfWZNQ6SuAOGGd7sNDdxFblrSJfXRtJihonEN1JqopoX%2F6cbFE%2B%2BZH61nZH%2BVFiLuKQnSqZTp8SZ%2BudjgkH2I4JgBp3Eh1Jqo5iSd1FQMYtBP1UXAsHEd1JqooexiTRdf56Li4qpQdJVUYyEXBrhoXBK1JqoAtuVORZQrX5VtfJAGbTQ0L4dObKVrf735JzIndHqhxMHFF0o80D%2F8MGAsKlbmB2S7fs8cGp%2Fxfw%3D%3D

# Application Code

## Git Hub Link

<https://github.com/ayj69/software-testing>

# Code in text

## Main

### Client.java

import java.util.\*;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class Client extends User {

private double distance;

String userNameRegex = "^[a-zA-Z0-9.\_-]{3,}$";

String phoneNumRegex = "^(01)[0-9][0-9]{7,8}$";

String distanceRegex = "^[0-9]+\\.?[0-9]\*$";

Client(String username, String password, String phoneNum, double distance) {

super(username, phoneNum, password);

this.distance = distance;

}

public boolean checkStringRegex(String data,String regex) {

if(data == null || regex == null) throw new IllegalArgumentException();

Pattern pat = Pattern.compile(regex);

Matcher mat;

mat = pat.matcher(data);

if (mat.find()) {

return true;

} else {

return false;

}

}

public void setUserName(String userName) {

if(this.checkStringRegex(userName,this.userNameRegex)) {

super.setUsername(userName);

}else {

throw new IllegalArgumentException("set Error,this username is not valid");

}

}

public boolean checkDistance(Double distance) {

if(distance == null)

throw new IllegalArgumentException("Error,thie value of the distance is null");

if(distance <= 0 || distance >= 100) {

throw new IllegalArgumentException("Error, Please enter your distance, the limit is 0 - 100");

}else {

return true;

}

}

public void setDistance(String distance) {

if(this.checkStringRegex(distance,this.distanceRegex)) {

this.distance = Double.parseDouble(distance);

}else {

throw new IllegalArgumentException("set Error,this distance is not valid");

}

}

public void setPhoneNumber(String phoneNum) {

if(this.checkStringRegex(phoneNum,this.phoneNumRegex)) {

super.setPhoneNum(phoneNum);

}else {

throw new IllegalArgumentException("set Error,this phone-num is not valid");

}

}

public double getDistance() {

return distance;

}

}

### DeliveryItemList.java

import java.util.\*;

import java.time.\*;

import java.time.format.DateTimeFormatter;

public class DeliveryItemList {

private ArrayList<Item> devItemList = new ArrayList<Item>();

private DeliveryStaff devStaff;

private LocalDate date;

private Client cli;

boolean sameDate = false;

boolean insurance = false;

// empty constructor for easy access // i am sure this practice is wrong but whatever

DeliveryItemList() {

}

public ArrayList<Item> getDevItemList() {

return devItemList;

}

public void setDevItemList(ArrayList<Item> devItemList) {

this.devItemList = devItemList;

}

public Client getCli() {

return cli;

}

public void setCli(Client cli) {

this.cli = cli;

}

public DeliveryStaff getDevStaff() {

return devStaff;

}

public void setDevStaff(DeliveryStaff devStaff) {

this.devStaff = devStaff;

}

public boolean isInsurance() {

return insurance;

}

public void setInsurance(boolean insurance) {

this.insurance = insurance;

}

// get the y/n answer

public boolean getyn(String choice) {

boolean flag = false;

Scanner scan = new Scanner(System.in);

if ((choice.equals("Y") || choice.equals("y"))) {

flag = true;

} else if ((choice.equals("N") || choice.equals("n"))) {

flag = false;

} else if (!(choice.equals("Y") || choice.equals("y") || choice.equals("n") || choice.equals("N"))) {

throw new IllegalArgumentException("Error,this choice is not valid");

}

return flag;

}

// print a list of item

// and let the user choose the item based on the index number

public LinkedList<Item> getItem(LinkedList<Item> itemList) {

LinkedList<Item> tempArr = new LinkedList<Item>();

String choice = "-";

ItemList itemlist = new ItemList();

Scanner scan = new Scanner(System.in);

while (!(choice.equals("x") || choice.equals("X"))) {

itemlist.printArrayString(itemList);

System.out.print("Please enter your choice or press X to exit : ");

choice = scan.next();

if (!(choice.equals("x") || choice.equals("X")) && isNumeric(choice)) {

int number = Integer.parseInt(choice);

if (number <= 0 || number > itemList.size()) {

System.out.println("out of range");

} else if (itemList.get(number - 1).getStatus() == ItemStatus.DELIVER) {

System.out.println("sry but this item is unavalible");

} else {

itemList.get(number - 1).setStatus(ItemStatus.DELIVER);

tempArr.add(itemList.get(number - 1));

}

}

}

if (tempArr.isEmpty())

System.out.println("the user did not chose any item");

else

System.out.println("this is your selected item : ");

itemlist.printArrayString(tempArr);

return tempArr;

}

// i copy from web

// check if a string is able to parse or not

public boolean isNumeric(String string) {

int intValue;

System.out.println(String.format("Verify Choice: \"%s\"", string));

if (string == null || string.equals("")) {

System.out.println("Invalid Choice, it is null or empty.");

return false;

}

try {

intValue = Integer.parseInt(string);

return true;

} catch (NumberFormatException e) {

System.out.println("Input String cannot be parsed to Integer.");

}

return false;

}

// validate the date and check if the date is before for 3 month in the future

public boolean validateMonthDay(int year, int month, int days) {

boolean flag;

LocalDate dateNow = LocalDate.now();

LocalDate date = LocalDate.of(year, month, days);

switch (month) {

case 1, 3, 5, 7, 9, 11:

if (days <= 31 && days > 0)

flag = true;

else

flag = false;

break;

case 4, 6, 8, 10, 12:

if (days <= 30 && days > 0)

flag = true;

else

flag = false;

break;

case 2:

if ((year % 4 == 0) && !(year % 100 == 0)) {

if (days <= 29 && days > 0)

flag = true;

else

flag = false;

} else {

if (days <= 28 && days > 0)

flag = true;

else

flag = false;

}

break;

default:

flag = false;

break;

}

if (date.isBefore(dateNow)) {

System.out.println("ERROR, Please enter today or future date.");

flag = false;

}

if (date.isAfter(dateNow.plusMonths(3))) {

System.out.println("Please enter the date within 3 month.");

flag = false;

}

return flag;

}

// get valid date

public void getDate() {

//LocalDate date = null;

int year = LocalDate.now().getYear();

int month;

int days;

Scanner scan = new Scanner(System.in);

boolean get = false;

do {

try {

System.out.print("Days : ");

days = scan.nextInt();

System.out.print("Month : ");

month = scan.nextInt();

if (validateMonthDay(year, month, days) == true)

date = LocalDate.of(year, month, days);

get = true;

} catch (Exception e) {

System.out.println(e.getMessage());

System.out.println("ERROR, invalid input");

scan.nextLine();

}

} while (!get || date == null);

scan.close();

this.date = date;

}

// check if its is same date

public void checkSameDate(LocalDate date) {

LocalDate dateNow = LocalDate.now();

if (date.isEqual(dateNow)) {

System.out.println("same date plus rm15");

this.sameDate = true;

}else {

throw new DateTimeException("The date format is invalid!");

}

}

// printing report

public void printDeliveryString() {

double basicPrice = 0;

double insurancePrice = this.insurance ? 15.0 : 0.0;

double totalPrice;

double samedatePrice = this.sameDate ? 10.0 : 0.0;

System.out.print(

"\nDelivery Detail\n---------------------------------------------------------------------------------------------------------------------------\n");

for (int i = 0; i < this.devItemList.size(); i++) {

double price = this.getDevItemList().get(i).calCharge(this.getDevItemList().get(i).getWeight(),

this.getCli().getDistance());

basicPrice += price;

System.out.println(String.format("%3d", i + 1) + " : " + this.devItemList.get(i).toString()

+ " Distance : " + String.format("%-10.2f",this.getCli().getDistance()) + " Price : " + String.format("%-10.2f", price));

}

System.out.print(

"---------------------------------------------------------------------------------------------------------------------------\n");

totalPrice = basicPrice + insurancePrice + samedatePrice;

System.out.println("Basic price = " + basicPrice);

if (this.insurance)

System.out.println("insurance = " + insurancePrice);

if (this.sameDate)

System.out.println("Samedate delivery fee = " + samedatePrice);

System.out.println("TotalPrice = " + totalPrice);

System.out.println("the parcel will be deliver at " + this.date);

System.out.print("your item will be deliver by : ");

System.out.println(this.getDevStaff().getUsername() + " Staff Phone-Num : " + this.devStaff.getPhoneNum());

}

}

### DeliveryStaff.java

import java.util.\*;

public class DeliveryStaff extends User {

private String employeeID;

DeliveryStaff(String username, String password, String phoneNum) {

super(username, phoneNum, password);

}

}

### Document.java

import java.util.ArrayList;

import java.util.Arrays;

import java.util.TreeMap;

public class Document extends Item {

private double basicCharge;

Document(String name, double weight) {

super(name, weight,"D");

}

@SuppressWarnings("serial")

private TreeMap<Double, Weight> weightTable = new TreeMap<Double, Weight>() {

{

put((double) 0, Weight.ZERO);

put((double) 300, Weight.LIGHT);

put((double) 1000, Weight.MEDIUM);

put((double) 3000, Weight.HEAVY);

put((double) 5000, Weight.TOOHEAVY);

put((double) 10000, Weight.IMPOSSIBLE);

}

};

@SuppressWarnings("serial")

private TreeMap<Double, Distance> distanceTable = new TreeMap<Double, Distance>() {

{

put((double) 0, Distance.ZERO);

put((double) 10, Distance.NEAR);

put((double) 30, Distance.MEDIUM);

put((double) 100, Distance.FAR);

}

};

private ArrayList<ArrayList<Double>> priceTable = new ArrayList<ArrayList<Double>>(Arrays.asList(

// distance = column 16 medium

// weight = row 12 light

new ArrayList<Double>(Arrays.asList((double) 0, (double) 0, (double) 0, (double) 0,(double) 0,(double) 0)),

new ArrayList<Double>(Arrays.asList((double) 0, (double) 3, (double) 4, (double) 6,(double) 12,(double) 25)),

new ArrayList<Double>(Arrays.asList((double) 0, (double) 3, (double) 5, (double) 8,(double) 18,(double) 30)),

new ArrayList<Double>(Arrays.asList((double) 0, (double) 3, (double) 6, (double) 10,(double) 25,(double) 35))));

@Override

public double calCharge(TreeMap<Double, Weight> weight, TreeMap<Double, Distance> distance, double weightVal,double distanceVal) {

Integer row = 0;

Integer col = 0;

try {

row = weight.higherEntry(weightVal).getValue().ordinal();

col = distance.higherEntry(distanceVal).getValue().ordinal();

}catch(Exception e) {

System.out.println("ERROR");

e.getMessage();

}

return priceTable.get(col).get(row);

}

@Override

public double calCharge(double weightVal, double distanceVal) {

return calCharge(weightTable, distanceTable, weightVal, distanceVal);

}

}

### Driver.java

import java.util.\*;

import java.io.File;

import java.io.FileNotFoundException;

import java.math.\*;

public class Driver {

public static void main(String[] args) {

boolean loop = false;

Scanner scan = new Scanner(System.in);

ItemList itemlist = new ItemList();

UserList userlist = new UserList();

// load the data file

ArrayList<Item> tempItemList = readItemFile();

ArrayList<Client> tempClientList = readClientFile();

ArrayList<DeliveryStaff> tempStaffList = readStaffFile();

for (Item i : tempItemList)

itemlist.getItemList().add(i);

for (Client c : tempClientList)

userlist.getClientList().add(c);

for (DeliveryStaff d : tempStaffList)

userlist.getDeliveryStaffList().add(d);

int userflag;// 1 = guest 2 = user

Client tempClient = new Client("", "", "", 00); // for storing guest data

// programm start

System.out.println("\n\nWelcome To Delivery Expert");

System.out.println("Enter as guess or login ?");

// if user or worker login else skip this part;

System.out.println("\t1. Enter as guess");

System.out.println("\t2. Let me login ");

// verify the user input for the menu choice

while (true) {

try {

// Some Code

// break out of loop, or return, on success

userflag = 0;

System.out.print("Please enter your choice : ");

userflag = scan.nextInt();

if (userflag == 1 || userflag == 2)

break;

else

System.out.println("ERROR, pls enter the number in range");

} catch (Exception e) {

// handle exception

System.out.println("ERROR, pls enter a number");

}

scan.nextLine();

}

// if user choose to login

if (userflag == 2) {

String username = "";

String password = "";

System.out.println(" LOGIN");

try {

do {

System.out.print("Pls enter your username to login : ");

try {

loop = false;

username = scan.next();

}catch(Exception e) {

System.out.println(e.getLocalizedMessage());

scan.nextLine();

loop = true;

}

}while(loop);

do {

System.out.print("Pls enter your password to login : ");

try {

loop = false;

password = scan.next();

}catch(Exception e) {

System.out.println(e.getLocalizedMessage());

scan.nextLine();

loop = true;

}

}while(loop);

userlist.valifyLogin(username, password);

tempClient = userlist.getClient(username);

}catch(Exception e) {

System.out.println(e.getLocalizedMessage());

System.out.println("[PROGRAME ENDED]");

System.exit(0);

}

}

// show the list of item to the user

// the user can choose an item

DeliveryItemList dil = new DeliveryItemList();

System.out.println("pls select the item you want to deliver");

// add the delivery item into the delivery list

dil.getDevItemList().addAll(dil.getItem(itemlist.getItemList()));

// if the user did not choose any item end the program

if (dil.getDevItemList().isEmpty()) {

System.out.println("The user did not choss any item procedd to exit the program");

System.out.println("[PROGRAME ENDED]");

System.exit(0);

}

// if guest

// enter guest detail

if (userflag == 1) {

System.out.println("You are a guess please enter your detail so that we can deliver to you");

do {

System.out.print("Please enter your username : ");

try {

loop = false;

String username = scan.next();

tempClient.setUserName(username);

}catch(Exception e) {

System.out.println(e.getLocalizedMessage());

loop = true;

}

}while(loop);

do {

System.out.print("Please enter your phone number : ");

try {

loop = false;

String phoneNum = scan.next();

tempClient.setPhoneNumber(phoneNum);

}catch(Exception e) {

System.out.println(e.getLocalizedMessage());

loop = true;

}

}while(loop);

do {

System.out.print("Please enter your distance : ");

try {

loop = false;

String distance = scan.next();

tempClient.setDistance(distance);

}catch(Exception e) {

System.out.println(e.getLocalizedMessage());

scan.nextLine();

loop = true;

}

}while(loop);

dil.setCli(tempClient);

} else if (userflag == 2) {

dil.setCli(tempClient);

}

// ask for insurance

boolean result = false;

do {

System.out.print("Insurance for delivery? (Y/N) : ");

try {

loop = false;

String choice = scan.next();

result = dil.getyn(choice);

}catch(Exception e) {

System.out.println(e.getLocalizedMessage());

scan.nextLine();

loop = true;

}

}while(loop);

if (result == true)

dil.setInsurance(true);

// ask for date

System.out.println("Please enter the date that you want the item to be deliver to you : ");

dil.getDate();

//dil.checkSameDate(dil.date);

// assign the delivery man

dil.setDevStaff(userlist.getRandomStaff());

// print the detail

dil.printDeliveryString();

System.out.print("[PROGRAME ENDED]");

scan.close();

}

// load item file

public static ArrayList<Item> readItemFile() {

System.out.println("Load Item Data...");

String linedata;

List data;

List<String> dataline = new ArrayList<String>();

ArrayList<Item> itemList = new ArrayList<Item>();

try {

File file = new File("Data/Item.txt");

Scanner scanf = new Scanner(file);

Document doc;

Parcel par;

scanf.useDelimiter(",");

while (scanf.hasNextLine()) {

String str = scanf.nextLine();

List<String> stratt = Arrays.asList(str.split(","));

if (stratt.get(2).equals("D")) {

doc = new Document(stratt.get(0), Double.parseDouble(stratt.get(1)));

itemList.add(doc);

} else if (stratt.get(2).equals("P")) {

par = new Parcel(stratt.get(0), Double.parseDouble(stratt.get(1)));

itemList.add(par);

}

}

scanf.close();

} catch (FileNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

return itemList;

}

// load staff file

public static ArrayList<DeliveryStaff> readStaffFile() {

System.out.println("Load Staff Data...");

String linedata;

List data;

List<String> dataline = new ArrayList<String>();

ArrayList<DeliveryStaff> devList = new ArrayList<DeliveryStaff>();

try {

File file = new File("Data/Staff.txt");

Scanner scanf = new Scanner(file);

scanf.useDelimiter(",");

while (scanf.hasNextLine()) {

String str = scanf.nextLine();

List<String> stratt = Arrays.asList(str.split(","));

DeliveryStaff dev = new DeliveryStaff(stratt.get(0), stratt.get(1), stratt.get(2));

devList.add(dev);

}

scanf.close();

} catch (FileNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

return devList;

}

// load client file

public static ArrayList<Client> readClientFile() {

System.out.println("Load Client Data...");

String linedata;

ArrayList<Client> cliList = new ArrayList<Client>();

List data;

List<String> dataline = new ArrayList<String>();

try {

File file = new File("Data/Client.txt");

Scanner scanf = new Scanner(file);

scanf.useDelimiter(",");

while (scanf.hasNextLine()) {

String str = scanf.nextLine();

List<String> stratt = Arrays.asList(str.split(","));

Client cli = new Client(stratt.get(0), stratt.get(1), stratt.get(2), Double.parseDouble(stratt.get(3)));

cliList.add(cli);

}

scanf.close();

} catch (FileNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

return cliList;

}

}

### Email.java

public class Email {

public boolean emailSendComformation() {

boolean flag = false;

//xxxxx

//get info from dil

//send it

//if send successful return true

//else return false

return flag;

}

public boolean getData() {

boolean flag = false;

//xxxxx

//get info from dil

//send it

//if send successful return true

//else return false

return flag;

}

public String toString() {

String str = "";

//returm delivery detail as email detail

return str;

}

}

### Guest.java

import java.util.\*;

public class Guest extends Client {

private double distance;

// constructor

Guest(String username, String phoneNum, double distance) {

super(username, phoneNum, "-", distance);

this.setDistance(distance);

}

public double getDistance() {

return distance;

}

public void setDistance(double distance) {

this.distance = distance;

}

}

### Item.java

import java.util.\*;

enum ItemStatus {

AVALIBLE, DELIVER, CANCEL

};

public abstract class Item {

private String name;

private ItemStatus status = ItemStatus.AVALIBLE;

private double weight;

private String itemType;

Item(String name, double weight, String type) {

this.name = name;

if(weight<0)

throw new IllegalArgumentException("cant be nagative");

else

this.weight = weight;

this.itemType = type;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getWeight() {

return weight;

}

public void setWeight(double weight) {

this.weight = weight;

}

public ItemStatus getStatus() {

return status;

}

public void setStatus(ItemStatus status) {

this.status = status;

}

public String toString() {

return "Name : " + String.format("%-15s", name) + " " + "Weight : " + String.format("%-10.2f", weight) + " "

+ " Status : " + String.format("%-10s", status);

}

public abstract double calCharge(double weightVal, double distanceVal);

public abstract double calCharge(TreeMap<Double, Weight> weight, TreeMap<Double, Distance> distance,

double weightVal, double distanceVal);

}

### ItemList.java

import java.util.\*;

public class ItemList {

private LinkedList<Item> itemList = new LinkedList<Item>();

private LinkedList<Item> tempList = new LinkedList<Item>();

// print item list based on status //currently unused

public void printArrayString2(List<Item> list) {

for (int i = 0; i < list.size(); i++) {

if (list.get(i).getStatus() == ItemStatus.AVALIBLE) {

tempList.add(list.get(i));

}

}

for (int i = 0; i < list.size(); i++) {

System.out.println(i + 1 + " : " + list.get(i).toString());

}

}

// print item list

public void printArrayString(List<?> list) {

System.out.print(

"\nITEM LIST\n----------------------------------------------------------------------------------\n");

for (int i = 0; i < list.size(); i++) {

System.out.println(String.format("%3d", i + 1) + " : " + list.get(i).toString());

}

System.out.println("----------------------------------------------------------------------------------");

}

public LinkedList<Item> getItemList() {

return itemList;

}

}

### Parcel.java

import java.util.\*;

enum Weight {

ZERO, LIGHT, MEDIUM, HEAVY, TOOHEAVY, IMPOSSIBLE

};

enum Distance {

ZERO, NEAR, MEDIUM, FAR

};

public class Parcel extends Item {

private double basicCharge;

Parcel(String name, double weight) {

super(name, weight,"P");

}

@SuppressWarnings("serial")

private TreeMap<Double, Weight> weightTable = new TreeMap<Double, Weight>() {

{

put((double) 0, Weight.ZERO);

put((double) 1000, Weight.LIGHT);

put((double) 2000, Weight.MEDIUM);

put((double) 3000, Weight.HEAVY);

put((double) 5000, Weight.TOOHEAVY);

put((double) 10000, Weight.IMPOSSIBLE);

}

};

@SuppressWarnings("serial")

private TreeMap<Double, Distance> distanceTable = new TreeMap<Double, Distance>() {

{

put((double) 0, Distance.ZERO);

put((double) 10, Distance.NEAR);

put((double) 30, Distance.MEDIUM);

put((double) 100, Distance.FAR);

}

};

private ArrayList<ArrayList<Double>> priceTable = new ArrayList<ArrayList<Double>>(Arrays.asList(

// distance = col

// weight = row

new ArrayList<Double>(Arrays.asList((double) 0, (double) 0, (double) 0, (double) 0,(double) 0,(double) 0)),

new ArrayList<Double>(Arrays.asList((double) 0, (double) 5, (double) 15, (double) 23,(double) 35,(double) 45)),

new ArrayList<Double>(Arrays.asList((double) 0, (double) 8, (double) 18, (double) 28,(double) 40,(double) 50)),

new ArrayList<Double>(Arrays.asList((double) 0, (double) 10, (double) 25, (double) 35,(double) 50,(double) 60))));

@Override

public double calCharge(TreeMap<Double, Weight> weight, TreeMap<Double, Distance> distance, double weightVal,double distanceVal) {

int row = weight.higherEntry(weightVal).getValue().ordinal();

int col = distance.higherEntry(distanceVal).getValue().ordinal();

return priceTable.get(col).get(row);

}

@Override

public double calCharge(double weightVal, double distanceVal) {

if(weightVal <0 || distanceVal<0)

throw new IllegalArgumentException("cant be nagative");

return calCharge(weightTable, distanceTable, weightVal, distanceVal);

}

}

### User.java

import java.util.\*;

enum UserStatus {

MEMBER, DELIVERSTAFF, GUEST, DEF

};

public class User {

private String username;

private String phoneNum;

private String password;

User(String username, String phoneNum, String password) {

this.username = username;

this.phoneNum = phoneNum;

this.password = password;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

// TODO Auto-generated method stub

return this.password;

}

public String getPhoneNum() {

return phoneNum;

}

public void setPhoneNum(String phoneNum) {

this.phoneNum = phoneNum;

}

}

### UserList.java

import java.util.\*;

public class UserList {

private LinkedList<Client> clientList = new LinkedList<Client>();

private LinkedList<DeliveryStaff> deliveryStaffList = new LinkedList<DeliveryStaff>();

// create a text file with exiting user

// i want to die

// your time management skill is shit bruh

public boolean valifyLogin(String username, String password) {

return valifyLogin(clientList, username, password);

}

public boolean checkUSerExist(String username) {

return checkUSerExist(clientList, username);

}

public boolean checkUSerExist(LinkedList<? extends User> list, String username) {

if(list == null || username == null) throw new IllegalArgumentException();

boolean flag = false;

for (User u : list) {

if (username.equals(u.getUsername())) {

flag = true;

break;

} else {

flag = false;

}

}

if (flag == false) {

throw new IllegalArgumentException("account not found");

}

return flag;

}

public boolean valifyLogin(LinkedList<? extends User> list, String username, String password) {

if(list == null || username == null || password == null) throw new IllegalArgumentException();

boolean flag = false;

boolean found = false;

for (User u : list) {

if (username.equals(u.getUsername())) {

if (password.equals(u.getPassword())) {

flag = true;

break;

} else {

throw new IllegalArgumentException("Incorrect Password");

// flag = false;

// break;

}

}

}

for (User u : list) {

if (username.equals(u.getUsername())) {

found = true;

}

}

if (!found) {

throw new IllegalArgumentException("account not found");

}

if (flag == true)

System.out.println("login successful");

return flag;

}

public DeliveryStaff getRandomStaff() {

int randomNumber = (int) Math.floor(Math.random() \* this.deliveryStaffList.size());

return this.deliveryStaffList.get(randomNumber);

}

public LinkedList<DeliveryStaff> getDeliveryStaffList() {

return deliveryStaffList;

}

public Client getClient(LinkedList<Client> list, String username) {

if(list == null || username == null) throw new IllegalArgumentException();

boolean found = false;

Client cil = null;

Optional<Client> checkNull = Optional.ofNullable(cil);

for (Client u : list) {

if (username.equals(u.getUsername())) {

cil = u;

found = true;

break;

}

if (checkNull.isPresent()) {

System.out.println("account found");

} else {

return cil;

}

}

return cil;

}

public Client getClient(String username) {

return getClient(this.clientList, username);

}

public LinkedList<Client> getClientList() {

return clientList;

}

}

## Test

### ClienTest.java

import static org.junit.Assert.assertEquals;

import static org.junit.Assert.assertTrue;

import org.junit.Test;

import org.junit.runner.RunWith;

import junitparams.JUnitParamsRunner;

import junitparams.Parameters;

@RunWith(JUnitParamsRunner.class)

public class ClientTest {

Client client = new Client("Adam", "1234", "016-789065", 69.42);

@Test

@Parameters(method = "testCheckStringRegexParamValid")

public void testCheckStringRegex(String data, String regex, boolean expectedResult) {

boolean result = client.checkStringRegex(data, regex);

assertEquals(expectedResult, result);

}

private Object[] testCheckStringRegexParamValid() {

return new Object[] {

//userName

new Object[] {"Ali.-\_123", client.userNameRegex, true},

new Object[] {"al", client.userNameRegex, false},

new Object[] {"&^%$@!", client.userNameRegex, false},

//phone

new Object[] {"0111234567", client.phoneNumRegex, true},

new Object[] {"01112345678", client.phoneNumRegex, true},

new Object[] {"011123456789", client.phoneNumRegex, false},

new Object[] {"12345667982", client.phoneNumRegex, false},

new Object[] {"012-12345567", client.phoneNumRegex, false},

new Object[] {"aA\*&^%-&123", client.phoneNumRegex, false},

//distance

new Object[] {"0.00", client.distanceRegex, true},

new Object[] {"1000", client.distanceRegex, true},

new Object[] {"-1", client.distanceRegex, false},

new Object[] {"-1.0", client.distanceRegex, false},

new Object[] {"abc^&\*123", client.distanceRegex, false},

};

}

@Test(expected = IllegalArgumentException.class)

@Parameters(method = "testCheckStringRegexInvalidParam")

public void testCheckStringRegexInvalid(String data, String regex) {

client.checkStringRegex(data, regex);

}

private Object[] testCheckStringRegexInvalidParam() {

return new Object[] {

new Object[] {"Ali", null},

new Object[] {"0111234561", null},

new Object[] {"0.5", null},

new Object[] {null, client.userNameRegex},

new Object[] {null, client.phoneNumRegex},

new Object[] {null, client.distanceRegex},

new Object[] {null, null},

};

}

}

### IntergrationTest.java

import java.io.BufferedReader;

import java.io.ByteArrayInputStream;

import java.io.ByteArrayOutputStream;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.InputStream;

import java.util.ArrayList;

import java.util.Scanner;

import org.junit.After;

import org.junit.AfterClass;

import org.junit.Assert;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Ignore;

import org.junit.Test;

import org.junit.runner.RunWith;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.LinkedList;

import junitparams.JUnitParamsRunner;

import junitparams.Parameters;

import org.junit.Before;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.mockito.InOrder;

import org.mockito.internal.util.io.IOUtil;

import static org.junit.Assert.\*;

import static org.mockito.Mockito.mock;

import static org.mockito.Mockito.never;

import static org.mockito.Mockito.times;

import static org.mockito.Mockito.verify;

import static org.mockito.Mockito.when;

import static org.mockito.Mockito.anyList;

import static org.mockito.Mockito.eq;

import static org.mockito.Mockito.inOrder;

@RunWith(JUnitParamsRunner.class)

public class IntergrationTest {

ItemList il;

DeliveryItemList dil;

@Before

public void setupForAllTests() {

il = new ItemList();

dil = new DeliveryItemList();

}

Document doc1 = new Document("doc1",11);

Document doc2 = new Document("doc2",12);

Parcel par3 = new Parcel("par3",13);

private Object[] getParamsForTestDisplayItem() {

return new Object[] {

new Object[] {"Name : " + String.format("%-15s", "doc1") + " " + "Weight : " + String.format("%-10.2f", 11.0) + " "

+ " Status : " + String.format("%-10s", "AVALIBLE"),doc1},

new Object[] {"Name : " + String.format("%-15s", "doc2") + " " + "Weight : " + String.format("%-10.2f", 12.0) + " "

+ " Status : " + String.format("%-10s", "AVALIBLE"),doc2},

new Object[] {"Name : " + String.format("%-15s", "par3") + " " + "Weight : " + String.format("%-10.2f", 13.0) + " "

+ " Status : " + String.format("%-10s", "AVALIBLE"),par3}

};

}

@Test

@Parameters(method = "getParamsForTestDisplayItem")

public void testItemToString(String expectedResult, Item item) {

assertEquals(expectedResult,item.toString());

// set both the student records and selected records to inputList

// so that if the sort operation does not return a proper result

// selected records still has a default value (inputList) to be

// compared to

}

}

### TestDeliveryItemList.java

import static org.junit.Assert.assertEquals;

import static org.junit.Assert.assertFalse;

import static org.junit.Assert.assertTrue;

import java.io.ByteArrayInputStream;

import java.io.InputStream;

import java.time.DateTimeException;

import java.time.Instant;

import java.time.LocalDate;

import java.time.LocalDateTime;

import java.time.ZoneId;

import javax.print.attribute.standard.DateTimeAtCompleted;

import javax.swing.plaf.TreeUI;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.junit.validator.PublicClassValidator;

import junitparams.JUnitParamsRunner;

import junitparams.Parameters;

@RunWith(JUnitParamsRunner.class)

public class TestDeliveryItemList {

@Test

@Parameters(method = "TestGetynValidParam")

public void TestGetynValid(String choice, boolean expectedResult) {

DeliveryItemList dil = new DeliveryItemList();

boolean choiceResult=dil.getyn(choice);

assertEquals(expectedResult, choiceResult);

}

private Object[] TestGetynValidParam() {

return new Object[] {

new Object[] { "Y", true},

new Object[] { "y", true },

new Object[] { "N", false },

new Object[] { "n", false },

};

}

@Test(expected = IllegalArgumentException.class)

@Parameters(method = "TestGetynInvalidParam")

public void TestGetynInvalid(String choice, boolean expectedResult) {

DeliveryItemList dil = new DeliveryItemList();

boolean choiceResult=dil.getyn(choice);

assertEquals(expectedResult, choiceResult);

}

private Object[] TestGetynInvalidParam() {

return new Object[] {

new Object[] { "", false},

new Object[] { "1", false },

new Object[] { "q", false },

new Object[] { "Q", false },

};

}

@Test

@Parameters(method = "TestIsNumericValidParam")

public void TestIsNumericValid(String string, boolean expectedResult) {

DeliveryItemList dil = new DeliveryItemList();

boolean choiceResult=dil.isNumeric(string);

assertEquals(expectedResult, choiceResult);

}

private Object[] TestIsNumericValidParam() {

return new Object[] {

new Object[] { "1", true},

new Object[] { "12", true },

new Object[] { "123", true },

new Object[] { "1234", true },

};

}

@Test

@Parameters(method = "TestIsNumericInvalidParam")

public void TestIsNumericInvalid(String string, boolean expectedResult) {

DeliveryItemList dil = new DeliveryItemList();

boolean choiceResult=dil.isNumeric(string);

assertEquals(expectedResult, choiceResult);

}

private Object[] TestIsNumericInvalidParam() {

return new Object[] {

new Object[] { "a", false},

new Object[] { "aA", false },

new Object[] { "A", false },

new Object[] { "AA", false },

};

}

@Test

@Parameters(method="TestValidateMonthDayValidParam")

public void TestValidateMonthDayValid(int year,int month,int days,boolean expectedResult) {

DeliveryItemList dil = new DeliveryItemList();

boolean YearMonthDay=dil.validateMonthDay(year, month, days);

assertEquals(expectedResult, YearMonthDay);

}

private Object[]TestValidateMonthDayValidParam(){

return new Object[] {

new Object[] {2022,5,12,true},

new Object[] {2022,7,12,true},

new Object[] {2022,4,20,true},

new Object[] {2022,6,12,true},

new Object[] {2022,2,28,false},

new Object[] {2024,2,29,false},

};

}

@Test(expected = DateTimeException.class)

@Parameters(method="TestValidateMonthDayInvalidParam")

public void TestValidateMonthDayInvalid(int year,int month,int days,boolean expectedResult) {

DeliveryItemList dil = new DeliveryItemList();

boolean YearMonthDay=dil.validateMonthDay(year, month, days);

assertEquals(expectedResult, YearMonthDay);

}

private Object[]TestValidateMonthDayInvalidParam(){

return new Object[] {

new Object[] {2022,5,32,false},

new Object[] {2022,7,-1,false},

new Object[] {2022,4,31,false},

new Object[] {2022,6,-1,false},

new Object[] {2022,2,29,false},

new Object[] {2024,2,30,false},

};

}

}

### TestDocument.java

import static org.junit.Assert.assertEquals;

import java.io.File;

import java.io.FileNotFoundException;

import java.util.ArrayList;

import java.util.Scanner;

import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Ignore;

import org.junit.Test;

import org.junit.runner.RunWith;

import junitparams.JUnitParamsRunner;

import junitparams.Parameters;

@RunWith(JUnitParamsRunner.class)

public class TestDocument {

static Scanner inputStream;

static ArrayList<String[]> linesRead;

@Before

public void setupData() {

linesRead = new ArrayList<String[]>();

String fileName = "Data/DocTestData.txt";

try {

inputStream = new Scanner(new File(fileName));

} catch (FileNotFoundException e) {

System.out.println("Error opening the file " + fileName);

System.exit(0);

}

while (inputStream.hasNextLine()) {

String singleLine = inputStream.nextLine();

String[] tokens = singleLine.split(",");

linesRead.add(tokens);

}

}

@After

public void closeFile() {

inputStream.close();

}

@Test

public void testDocumentCalChargeValid() {

for(int i=0;i<linesRead.size();i++) {

String[] inputStr = linesRead.get(i);

double weightVal=Double.valueOf(inputStr[0]);

double distanceVal=Double.valueOf(inputStr[1]);

double expected=Double.valueOf(inputStr[2]);

Document D = new Document("", weightVal);

assertEquals(expected, D.calCharge(weightVal, distanceVal),0);

}

}

@Test(expected=IllegalArgumentException.class)

@Parameters(method="testDocumentInvalidCalChargeParam")

public void testDocumentCalChargeInvalid(double weightVal,double distanceVal) {

Parcel P = new Parcel("", weightVal);

P.calCharge(weightVal, distanceVal);

}

private Object[] testDocumentInvalidCalChargeParam() {

return new Object[] {

new Object[] {null, null},

new Object[] {-10, null},

new Object[] {null, -10},

new Object[] {-10, -10},

};

}

}

### TestEmail.java

import static org.junit.Assert.assertEquals;

import static org.junit.Assert.assertTrue;

import org.junit.Test;

import org.mockito.Mockito;

public class TestEmail {

@Test

public void testemailSendComformation() {

final Email testEmail = Mockito.spy(new Email());

Mockito.when(testEmail.emailSendComformation()).thenReturn(true);

System.out.println(testEmail.emailSendComformation());

assertTrue(testEmail.emailSendComformation());

}

@Test

public void testgetData() {

final Email testEmail = Mockito.spy(new Email());

Mockito.when(testEmail.getData()).thenReturn(true);

System.out.println(testEmail.getData());

assertTrue(testEmail.getData());

}

@Test

public void testtoString() {

final Email testEmail = Mockito.spy(new Email());

Mockito.when(testEmail.toString()).thenReturn("email detail");

System.out.println(testEmail.toString());

assertEquals("email detail",testEmail.toString());

}

}

### TestItem.java

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class TestItem {

@Test

public void testToString() {

Item P = new Parcel("A",20.9);

Item D = new Document("B",30.7);

assertEquals("Name : A Weight : 20.90 Status : AVALIBLE ", P.toString());

assertEquals("Name : B Weight : 30.70 Status : AVALIBLE ", D.toString());

}

@Test(expected=IllegalArgumentException.class)

public void testToStringInvalid() {

Item P = new Parcel("A",-20);

Item D = new Document("B",-30);

assertEquals("Name : A Weight : 20.90 Status : AVALIBLE ", P.toString());

assertEquals("Name : B Weight : 30.70 Status : AVALIBLE ", D.toString());

}

}

### TestParcel.java

import static org.junit.Assert.assertEquals;

import java.io.File;

import java.io.FileNotFoundException;

import java.util.ArrayList;

import java.util.Scanner;

import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Ignore;

import org.junit.Test;

import org.junit.runner.RunWith;

import junitparams.JUnitParamsRunner;

import junitparams.Parameters;

@RunWith(JUnitParamsRunner.class)

public class TestParcel {

static Scanner inputStream;

static ArrayList<String[]> linesRead;

@Before

public void setupData() {

linesRead = new ArrayList<String[]>();

String fileName = "Data/ParTestData.txt";

try {

inputStream = new Scanner(new File(fileName));

} catch (FileNotFoundException e) {

System.out.println("Error opening the file " + fileName);

System.exit(0);

}

while (inputStream.hasNextLine()) {

String singleLine = inputStream.nextLine();

String[] tokens = singleLine.split(",");

linesRead.add(tokens);

}

}

@After

public void closeFile() {

inputStream.close();

}

@Test

public void testParcelCalChargeValid() {

for(int i=0;i<linesRead.size();i++) {

String[] inputStr = linesRead.get(i);

double weightVal=Double.valueOf(inputStr[0]);

double distanceVal=Double.valueOf(inputStr[1]);

double expected=Double.valueOf(inputStr[2]);

Parcel P = new Parcel("", weightVal);

assertEquals(expected, P.calCharge(weightVal, distanceVal),0);

}

}

@Test(expected=IllegalArgumentException.class)

@Parameters(method="testParcelInvalidCalChargeParam")

public void testParcelCalChargeInvalid(double weightVal,double distanceVal) {

Parcel P = new Parcel("", weightVal);

P.calCharge(weightVal, distanceVal);

}

private Object[] testParcelInvalidCalChargeParam() {

return new Object[] {

new Object[] {null, null},

new Object[] {-10, null},

new Object[] {null, -10},

new Object[] {-10, -10},

};

}

}

### TestSuite.java

import org.junit.runner.RunWith;

import org.junit.runners.Suite;

import org.junit.runners.Suite.SuiteClasses;

@RunWith(value = Suite.class)

@SuiteClasses(value = { ClientTest.class,

TestDeliveryItemList.class,

TestDocument.class,

TestEmail.class,

TestItem.class,

TestParcel.class,

testUserList.class })

public class TestSuite {

}

### testUserList.java

import static org.junit.Assert.assertEquals;

import static org.junit.Assert.assertFalse;

import static org.junit.Assert.assertTrue;

import static org.mockito.Mockito.mock;

import static org.mockito.Mockito.when;

import java.util.LinkedList;

import org.junit.Test;

import org.junit.runner.RunWith;

import junitparams.JUnitParamsRunner;

import junitparams.Parameters;

@RunWith(JUnitParamsRunner.class)

public class testUserList {

LinkedList<User> linkedListUser = new LinkedList<User>();

@Test

public void testCheckUSerExistValid() {

UserList userList = new UserList();

User userAli = new User("Ali", "011-15679800", "12345");

linkedListUser.add(userAli);

assertTrue(userList.checkUSerExist(linkedListUser, "Ali"));

}

@Test(expected = IllegalArgumentException.class)

@Parameters(method = "testCheckUSerExistInvalidParam")

public void testCheckUSerExistInvalid(LinkedList<User> list, String username) {

UserList userList = new UserList();

userList.checkUSerExist(list, username);

}

private Object[] testCheckUSerExistInvalidParam() {

return new Object[] {

new Object[] {null, "Ali"},

new Object[] {linkedListUser, null},

new Object[] {null, null}

};

}

@Test

public void testValifyLoginValid() {

UserList userList = new UserList();

User userAli = new User("Ali", "011-15679800", "12345");

linkedListUser.add(userAli);

assertTrue(userList.valifyLogin(linkedListUser, "Ali", "12345"));

}

@Test(expected = IllegalArgumentException.class)

@Parameters(method = "testValifyLoginInvalidParam")

public void testValifyLoginInvalid(LinkedList<User> list, String username, String password) {

UserList userList = new UserList();

userList.valifyLogin(list, username, password);

}

private Object[] testValifyLoginInvalidParam() {

return new Object[] {

new Object[] {null, null, null},

new Object[] {linkedListUser, null, null},

new Object[] {null, "Ali", null},

new Object[] {null, null, "12345"},

new Object[] {linkedListUser, "Ali", null},

new Object[] {null, "Ali", "12345"},

new Object[] {linkedListUser, null, "12345"}

};

}

LinkedList<Client> linkedListClient = new LinkedList<Client>();

@Test

public void testGetClientValid() {

Client client = new Client("Chong", "12345", "019-9871423", 43.45);

linkedListClient.add(client);

UserList userList = new UserList();

Client resultClient = userList.getClient(linkedListClient, "Chong");

assertEquals(client, resultClient);

Client clientNull = null;

resultClient = userList.getClient(linkedListClient, "Ali");

assertEquals(clientNull, resultClient);

}

@Test(expected = IllegalArgumentException.class)

@Parameters(method = "testGetClientInvalidParam")

public void testGetClientInvalid(LinkedList<Client> linkedListClient, String username) {

UserList userList = new UserList();

userList.getClient(linkedListClient, username);

}

private Object[] testGetClientInvalidParam() {

return new Object[] {

new Object[] {linkedListClient, null},

new Object[] {null, "Chong"},

new Object[] {null, null},

};

}

}