IS442 Trial Lab Test

Resource: http://blue.smu.edu.sg/trial-resource.zip

General Instructions:

- You can refer to any offline resources already on your laptop, but you must disable all networking and Bluetooth connections during the test. You must not communicate with anyone via any means during the test.
- Just before the test, you will be given instructions by the invigilator as to how to obtain resource files required for the lab test. You will need to submit your solutions on your personal thumb drive.
- No questions will be entertained during the test. If necessary, make your own assumptions.
- You are allowed to use only standard Java library classes in your solutions – do not use any third party libraries.

Failure to do the following will attract a penalty of up to **20%** of your score for the corresponding Java file.

You MUST:

a) Include your name as author in the comments of all your submitted source files. For example, if your registered name is "TAN So Tong" and email ID is tan.sotong. 2015, include the following comment at the beginning of each source (.java) file you write.

```
/*
Name: TAN So Tong
Email ID: tan.sotong.2015
*/
```

b) Follow standard Java coding conventions (e.g. naming of getter and setter methods, choice of identifier names for classes, methods and variables) as well as indent your code correctly. Use 4 spaces for indentation.

Ensure that all your Java code can compile without errors. They must compile with any test class(es) that are provided. (You may wish to comment out the parts in your code which cause compilation errors. But commented code will not be marked.)

If you code doesn't compile then your <u>0 mark</u> for the corresponding question or sub-question.

DO NOT TURN OVER UNTIL YOU ARE TOLD TO DO SO

Question 1

Human.java and **IllegalInputException.java** are provided and will be required for some parts of the question. Note that **IllegalInputException** is an <u>un</u>checked exception that has a constructor that takes in the exception message.

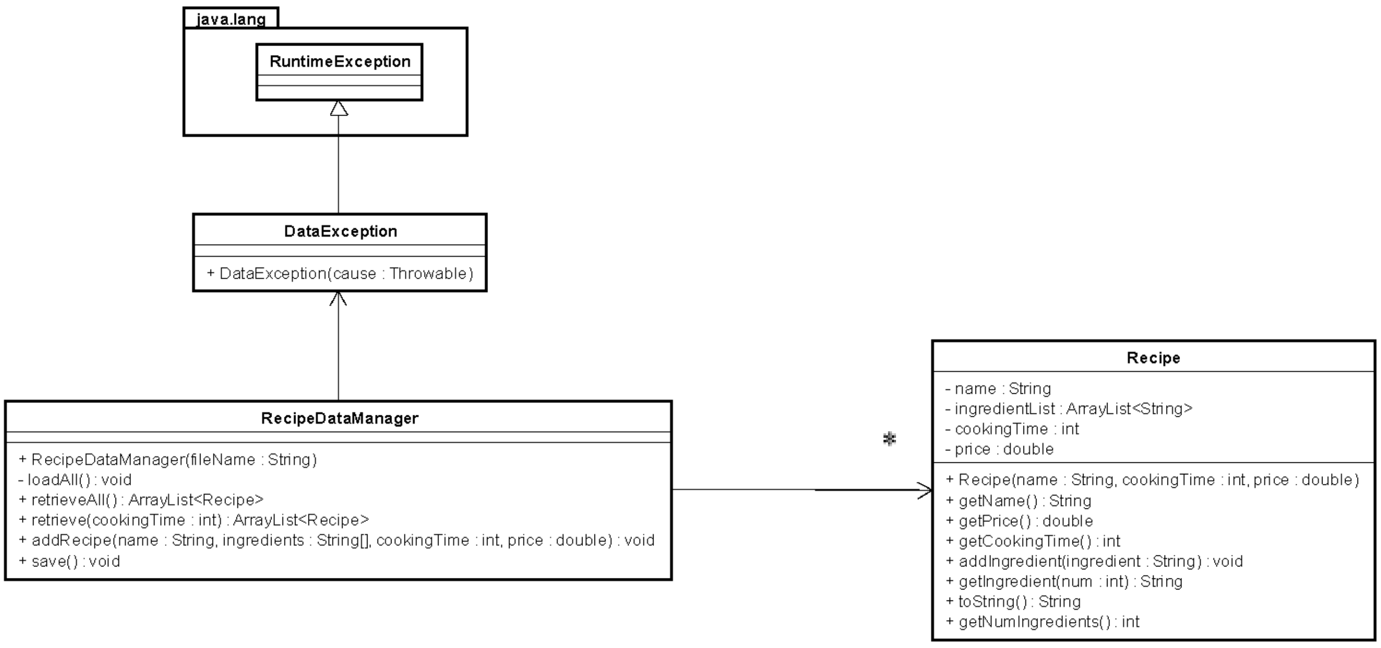
You are given six skeletal classes: **Q1a.java**, **Q1b.java**, **Q1c.java**… **Q1f.java**. Each contains a method that you need to fill up. Do not modify the method signatures. You are allowed to use only standard Java library classes (no third party libraries). Refer to the comments in each source file for a description of what each method should do.

(a) combineTwoCharArrays
(b) countNumberOfMatches
(c) getOldest
(d) combineTwoIntArraysSorted
(e) getHumansBelowAge
(f) flip

For each **Q1<x>.java** class, you are given a corresponding **Q1<x>Test.java** that you can use to verify that each method has been correctly written. Use them to your advantage.

Question 2

Given the following class diagram:


java.lang
RuntimeException
DataException
+ DataException(cause : Throwable)
RecipeDataManager
+ RecipeDataManager(fileName : String)
- loadAll() : void
+ retrieveAll() : ArrayList<Recipe>
+ retrieve(cookingTime : int) : ArrayList<Recipe>
+ addRecipe(name : String, ingredients : String[], cookingTime : int, price : double) : void
+ save() : void
*
Recipe
- name : String
- ingredientList : ArrayList<String>
- cookingTime : int
- price : double
+ Recipe(name : String, cookingTime : int, price : double)
+ getName() : String
+ getPrice() : double
+ getCookingTime() : int
+ addIngredient(ingredient : String) : void
+ getIngredient(num : int) : String
+ toString() : String
+ getNumIngredients() : int

1. Implement the exception class, DataException.

2. Implement the following four methods of the **RecipeDataManager** class:
   A. loadAll ()
   This method reads the file called "**recipe.txt**" provided and populates an **ArrayList** of **Recipe** objects.
   B. addRecipe(String name, String[] ingredients, int cookingTime, double price)

This method adds a new recipe to the `recipeList`. *No two Recipe object should have the same name.*

C. retrieveRecipes(int cookingTime)
This method returns all the recipes with cooking time longer than `cookingTime`**.**

D. save()
This method saves the complete `ArrayList<Recipe>` object to the file called "**recipe.txt**" overwriting the existent content.

Refer to the partial **RecipeDataManager.java** source code for more guidance.

The following files are provided:

- **Recipe.java** (Do NOT modify this file)
- **RecipeDataManager.java** (partial). You just need to complete the four methods requested into this same file and submit it**.**
- **RecipeDataManagerTest.java** (Use this file for your testing. Your answers must NOT be in this file!)

Below is a line from the sample data file:

```
steak#beef#beef#beef#beef#25#7.9
```

There are 7 values in the file as follows: steak (name), beef (ingredient1), beef (ingredient2), beef (ingredient3), beef (ingredient4), 25 (cookingTime), and 7.9 (price).

An execution of **ReportDataManagerTest** produces the following output:

```
java.io.FileNotFoundException: secret-recipe.txt (The system cannot find the file specified)
        at java.io.FileInputStream.open0(Native Method)
        at java.io.FileInputStream.open(Unknown Source)
        at java.io.FileInputStream.<init>(Unknown Source)
        at java.util.Scanner.<init>(Unknown Source)
        at RecipeDataManager.loadAll(RecipeDataManager.java:37)
        at RecipeDataManager.<init>(RecipeDataManager.java:28)
        at RecipeDataManagerTest.main(RecipeDataManagerTest.java:18)

All Recipes.
1. steak[ beef + beef + beef + beef ]
2. burger & fries[ beef + potato ]
3. bak kut teh[ coriander + pork + garlic ]
4. caesar salad[ salad + bread + egg ]
5. chicken korma[ chicken + coconut + chilli + onion ]
6. lobster salad[ lobster + tomato + coriander + salad ]
7. fluff[ smoke + feather ]

Recipes whose cooking time exceeds(more than) 20.
1. steak[ beef + beef + beef + beef ]
2. chicken korma[ chicken + coconut + chilli + onion ]
3. fluff[ smoke + feather ]

After add:
1. steak[ beef + beef + beef + beef ]
2. burger & fries[ beef + potato ]
3. bak kut teh[ coriander + pork + garlic ]
4. caesar salad[ salad + bread + egg ]
5. chicken korma[ chicken + coconut + chilli + onion ]
6. lobster salad[ lobster + tomato + coriander + salad ]
7. fluff[ smoke + feather ]
8. Sandwich[ bread + bread + tomato ]

Correct - cannot add another Sandwich.
```

```
Load again:
1. steak[ beef + beef + beef + beef ]
2. burger & fries[ beef + potato ]
3. bak kut teh[ coriander + pork + garlic ]
4. caesar salad[ salad + bread + egg ]
5. chicken korma[ chicken + coconut + chilli + onion ]
6. lobster salad[ lobster + tomato + coriander + salad ]
7. fluff[ smoke + feather ]
8. Sandwich[ bread + bread + tomato ]
```

Question 3

You are given a set of Java classes:

A. `Product.class`
B. `InvalidProductException.class`

Modify `ProductTest.java` so that it compiles and produces the following output:

```
Enter product name : pen
Enter product code : 322-156-787
Enter product price : 3.2
Invalid Product Code! Please try again!

Enter product name : pen
Enter product code : 012-158-707
Enter product price : 3.2
Invalid Product Code! Please try again!

Enter product name : pen
Enter product code : 012-156-777
Enter product price : 3.2
The discounted price is 2.8800000000000003
```

Note:

1. The program will keep prompting the user if he/she enters a negative price or invalid product code.
2. The program should be flexible enough to handle other values entered for price or code (i.e. no hard-coding).

- END -