Lab Test [20 marks]

General Instructions:

1. You will perform the lab test on your personal laptop.
2. You are not allowed to communicate in any way during the test. Disable the following on your laptop before the test begins: WIFI, Bluetooth, and any other communication devices (e.g. 3G/4G modems).
3. You can refer to any file on your laptop.
   Any violation of the instructions above will result in a zero score for your lab test.

1. Make sure your code can generate exactly the same output as we show in the sample runs. You may get some marks deducted for missing spaces, missing punctuation marks, misspelling, etc. in the output.
2. Do not hardcode. We will use different test cases to test and grade your solutions.

Failure to do the following will attract a penalty up to 20% of your score for that question.

- Include your name as author in the comments of all your submitted source files. For example, if your registered name is "Sotong TAN" and email ID is sotong.tan.2020, include the following block of comments at the beginning of each source file (.java) you write.

```
/*
 * Name: Sotong Tan
 * Email ID: sotong.tan.2021
 */
```

- 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.

- Ensure that all your Java code can compile without compilation 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 remember that commented code will NOT be graded. **Only code without compilation errors SHALL be graded.** You may need to comment out parts of the Test class' code that you have not implemented in order to test your implemented solutions.

- Download the source code and rename the root folder in the zip file to your **Email ID (e.g. `sotong.tan.2021`)**

Question 1 [13 marks]

Refer to the class diagram in the appendix and implement the following classes accordingly.

You should only be writing codes in `App.java, Person.java, Employee.java, Dependent.java, InvalidDataException.java`. You can write additional helper methods in `App.java` if required. Do not modify `AppTest.java`.

a. Implement `Person`.
   - `toString()` returns a String in the following format
     - Name : <`firstName`> <`lastName`> - Age : <`age`>
     - e.g. Name : Johnson Lee - Age : 45

   If firstName is null, returns a String in the following format
     - Name : <`LastName`> - Age : <`age`>
     - e.g. Name: Lee - Age : 45

b. Implement `Employee`.

c. Implement `Dependent`.
   - There are two types of dependents: 'S' for Spouse and 'C' for Child.
   - `toString()` returns a String in the following format
     a. Child Name : <`firstName`> <`lastName`> - Age : <`age`>
        e.g. Child Name : Janice Lee - Age : 8
     b. Spouse Name : <`firstName`> <`lastName`> - Age : <`age`>
        e.g. Spouse Name : Gina Lee - Age : 28

   - `compareTo(Dependent o)` is used to sort the Dependents objects based on the following order.
     - Dependent's type in the following order (child, spouse) .
     - children `firstName` in **ascending** order (e.g. 'a' before 'b', 'ab' before 'ac') .
       note : `firstName` may be null (child without a `firstName` would be before the other child with a `firstName` - refer to the output given below for the order of such cases)

d. Implement `InvalidDataException`.

e. Implement `loadFile(String filename)` in `App.java` with the following requirements.
   - Reads a text file about `Employee` and his/her `Dependent` (could be a Spouse or Child) and stores them in a `Map<Integer, Employee>` where `Integer` refers to the employee's ID.

   - Each row of data is represented in the following format
   e.g. 1#Johnson Lee,45#Jane Tan,30#Janice Lee,8,June Lee,10,Jenny Lee,2

     1. The integer (e.g. 1) before the first "#" corresponds to the **employee's id**

     2. This is followed by the `firstName lastName`, `age` of the **employee**,

     3. `firstName lastName`, `age` of **spouse** (if any), and

     4. `firstName lastName`, `age` of **child** (if any).

Note:

1. `firstName` is optional while `lastName` will always be provided for employees, spouse and child. In this case, “Low” is the `lastName`.
   e.g. 3#**Low**,30#Daisy Tan,25#Danny Low,5,Daniel Low,3

2. The `lastName` consists of 1 word and is the **last word** in the name. For example, If the full name is "Nicholas Ah Beng Tan", "Nicholas Ah Beng" is the `firstName`, "Tan" is the `lastName`. If the full name is "Ah Beng Tan", "Ah Beng" is the `firstName`, "Tan" is the `lastName`. `firstName` can be optional or be any number of words.

3. The Employee has at most 1 spouse and any number of children.

4. It is not possible for the Employee to have 0 spouse with 1 or more children.

5. This method throws a `FileNotFoundException` if the input file is missing, and throws a `InvalidDataException` if the employee's ID is not a numeric valid.

f. Use `AppTest` to test your code.

- The employees are sorted by Employee Id, and
- the dependents to be sorted accordingly based on `Dependent’s` `Comparable` implementation. Refer provided output below.

Note: You must not *modify* AppTest to achieve the given output!

```
*** Test File : "testdata0.txt" ***
Expected : FileNotFoundException
  Actual : FileNotFoundException
  Result : Passed

*** Test File : "testdata1.txt" ***
Expected : InvalidDataException java.lang.NumberFormatException: For input
string: "b"
OR
Expected : InvalidDataException java.util.InputMismatchException
  Actual : InvalidDataException java.util.InputMismatchException

*** Test File : "testdata2.txt" ***
Expected : 4 employees loaded
  Actual : 4 employees loaded
  Result : Passed

Expected : Employee and Dependents
---------------------------------------
1.Name : Steven Lim - Age : 35
Spouse Name : Lee - Age : 20
---------------------------------------
2.Name : Johnson Lee - Age : 45
Child Name : Lee - Age : 2
Child Name : Ah Seng Lee - Age : 10
Child Name : Janice Lee - Age : 15
Spouse Name : Jane Tan - Age : 30
---------------------------------------
3.Name : Low - Age : 30
Child Name : Daniel Low - Age : 3
Child Name : Danny Low - Age : 5
Spouse Name : Daisy Tan - Age : 25
```

```
---------------------------------------
4.Name : Andy Tan - Age : 25
---------------------------------------

Actual : Employee and Dependents
---------------------------------------
1.Name : Steven Lim - Age : 35
Spouse Name : Lee - Age : 20
---------------------------------------
2.Name : Johnson Lee - Age : 45
Child Name : Lee - Age : 2
Child Name : Ah Seng Lee - Age : 10
Child Name : Janice Lee - Age : 15
Spouse Name : Jane Tan - Age : 30
---------------------------------------
3.Name : Low - Age : 30
Child Name : Daniel Low - Age : 3
Child Name : Danny Low - Age : 5
Spouse Name : Daisy Tan - Age : 25
---------------------------------------
4.Name : Andy Tan - Age : 25
---------------------------------------
```

Question 2 [4 marks]

Refer to the class diagram in appendix and complete the following:

1. Organize the source files in the source folder according to the class diagram so that your compile.sh/compile.bat and run.sh/run.bat will work correctly.

2. Update the Java source files to include the correct package and import statements with the help of the class diagram.

3. VendingMachineTest contains the main()method to test the above classes.

4. Write a one-liner compile.sh/compile.bat such that classes are compiled to folder output.

5. Write a one-liner run.sh/run.bat to run the main method in VendingMachineTest.

Note:

1. Do not include unnecessary folders/jars in the sourcepath/classpath.
2. If you get any warning while compiling, add the suggested option in compile.bat.

```
warning: Implicitly compiled files were not subject to annotation processing.
  Use -proc:none to disable annotation processing or -implicit to specify a policy
for implicit compilation.
```

For example, to include this option while compiling `Test.java`:

```
javac -proc:none Test.java
```

The directory should look like this:

```
q2\
 |- config\
 |   |- log4j2.properties
 |
 |- output\
 |   |- <your generated classes here>
 |
 |- references\
 |   |- log4j2\
 |   |    |- log4j-api-2.11.2.jar
 |   |    |- log4j-core-2.11.2.jar
 |   |- q2\
 |   |    |- vending\
 |   |          |- exception\
 |   |                 |- InsufficientMoneyException.class
 |   |                 |- OutOfStockException.class
 |   |- entity
 |   |   |- money.jar
 |
 |- src\
 |    |- ...
 |- logs\
 |
 |- compile.sh
 |- run.sh
```

If `run.sh` runs successfully, the logs folder will be populated with a `test.log` file and the following will be displayed on the console

```
numCash = 10
numCash = 110
Result: Pass
```

Question 3 [3 marks]

Refer to the class diagram in appendix and implement the following classes accordingly. **You should only be writing codes in `SinglyLinkedList.java`, you can write additional helper methods in `SinglyLinkedList.java` if required. Do not modify `SinglyLinkedListTest.java`.**

An implementation of a Single Linked List has been given in `SinglyLinkedList.java`. Implement the following in `SinglyLinkedList.java`

- Write a method named `sort` that sorts the elements based on the Class name of the elements in ascending order (e.g. 'a' before 'b', 'ab' before 'ac')

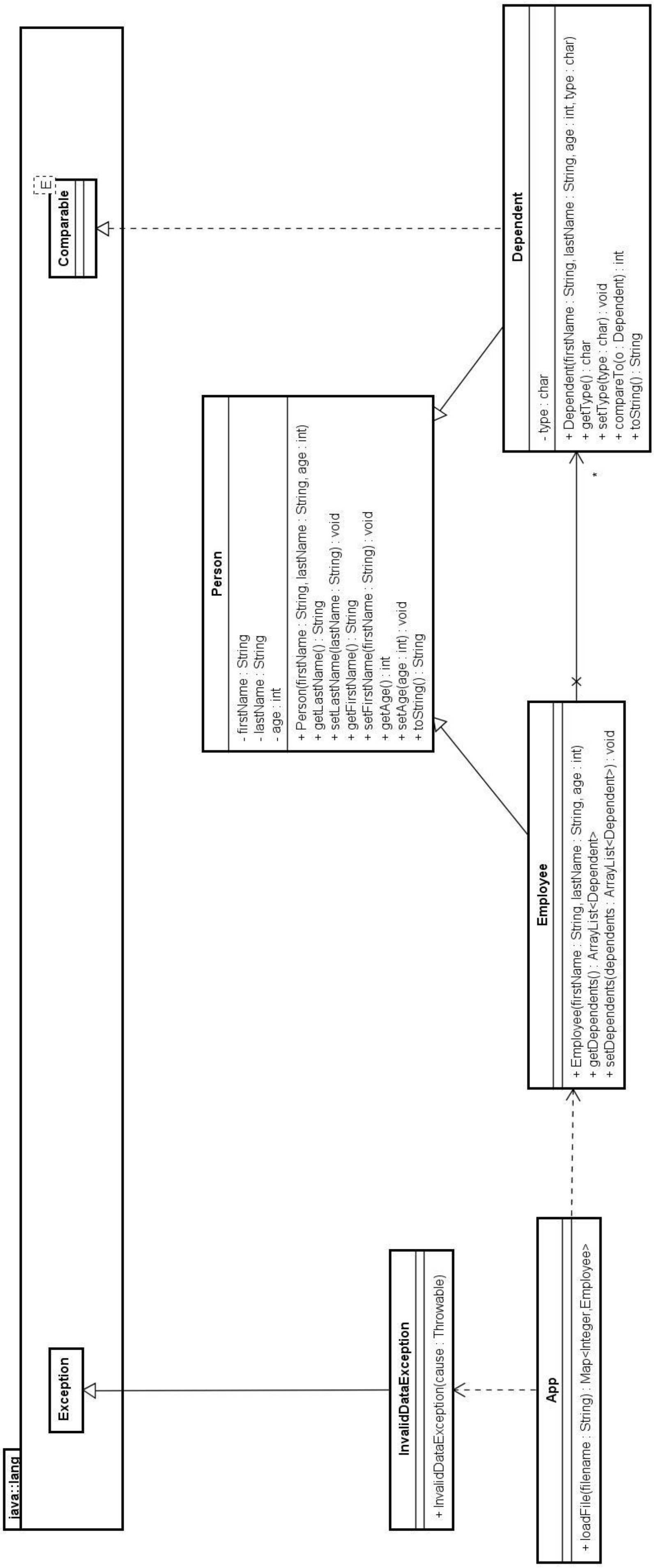
- Please do not assume there's only Square, Rectangle and Triangle objects. The program can take in other objects that inherit from the Shape class.

- You should not use any other data structures (i.e. array, ArrayList or any other collection classes).

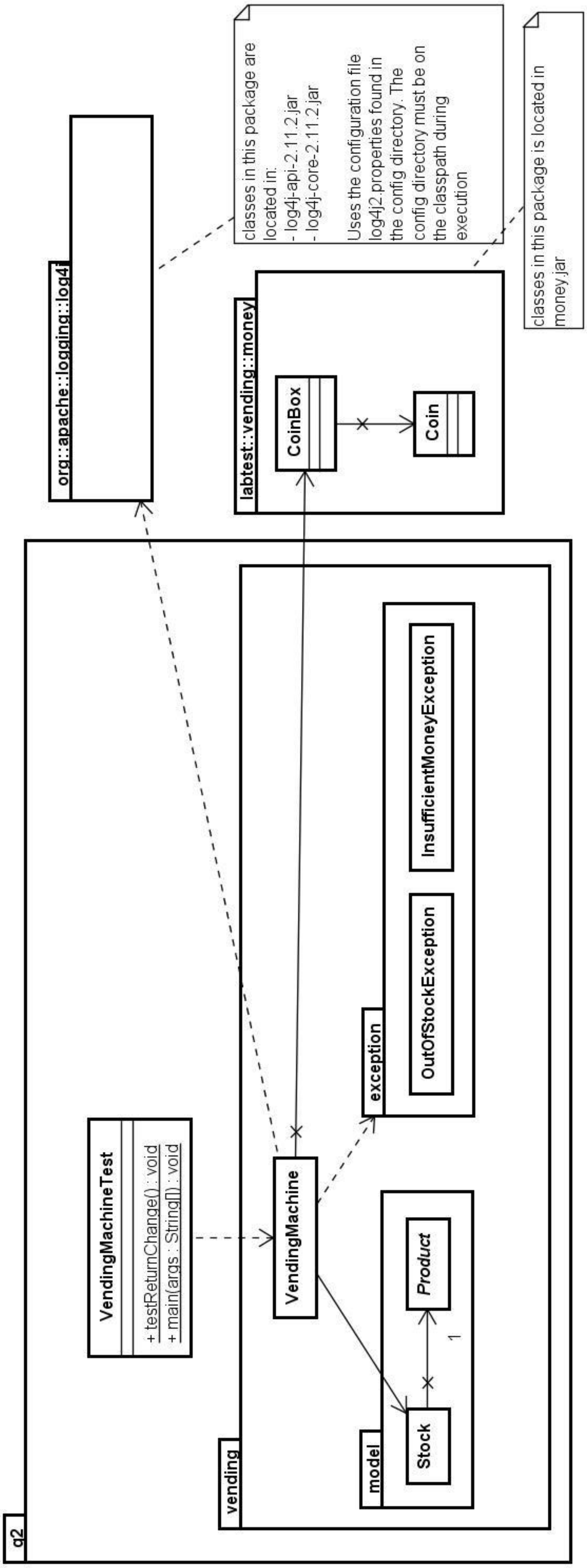
- Use `SinglyLinkedListTest` to test your code.

```
******************************************
Test Case 1 : (Triangle)(Rectangle)(Square)

Expected : (Rectangle)(Square)(Triangle)
     Actual : (Rectangle)(Square)(Triangle)
     Result : Passed
******************************************
Test Case 2 : (Square)(Rectangle)(Triangle)

Expected : (Rectangle)(Square)(Triangle)
     Actual : (Rectangle)(Square)(Triangle)
     Result : Passed
******************************************
Test Case 3 : (Rectangle)(Square)(Triangle)(Square)

Expected : (Rectangle)(Square)(Square)(Triangle)
     Actual : (Rectangle)(Square)(Square)(Triangle)
     Result : Passed
******************************************
Test Case 4 : (Square)(Rectangle)(Square)(Triangle)(Rectangle)

Expected : (Rectangle)(Rectangle)(Square)(Square)(Triangle)
     Actual : (Rectangle)(Rectangle)(Square)(Square)(Triangle)
     Result : Passed
```

APPENDIX 1


java::lang
Comparable
E
Exception
Person
- firstName : String
- lastName : String
- age : int
+ Person(firstName : String, lastName : String, age : int)
+ getLastName() : String
+ setLastName(lastName : String) : void
+ getFirstName() : String
+ setFirstName(firstName : String) : void
+ getAge() : int
+ setAge(age : int) : void
+ toString() : String
Dependent
- type : char
+ Dependent(firstName : String, lastName : String, age : int, type : char)
+ getType() : char
+ setType(type : char) : void
+ compareTo(o : Dependent) : int
+ toString() : String
*
Employee
+ Employee(firstName : String, lastName : String, age : int)
+ getDependents() : ArrayList<Dependent>
+ setDependents(dependents : ArrayList<Dependent>) : void
InvalidDataException
+ InvalidDataException(cause : Throwable)
App
+ loadFile(filename : String) : Map<Integer,Employee>

APPENDIX 2


g2
vending
model
Stock
Product
1
exception
OutOfStockException
InsufficientMoneyException
VendingMachine
VendingMachineTest
+ testReturnChange() : void
+ main(args : String[]) : void
org::apache::logging::log4j
labtest::vending::money
CoinBox
Coin
classes in this package are located in:
- log4j-api-2.11.2.jar
- log4j-core-2.11.2.jar
Uses the configuration file log4j2.properties found in the config directory. The config directory must be on the classpath during execution
classes in this package is located in money.jar