OOAD Lab Test 1 (2 hours) [40 marks]

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.2016
*/
```

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.

You should attempt that all your Java code can compile without errors. You should attempt to 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 your code doesn't compile, then you will get <u>0 mark</u> for the corresponding question or sub-question.

DO NOT TURN OVER UNTIL YOU ARE TOLD TO DO SO

Question 1 **[24 marks]**

Refer to the class diagram in appendix 1

1. **[6 marks]** Create class `ItemPack` according to the class diagram.

Expected output:

```
C:\> java ItemPackTest
Test class ItemPack declaration: pass

Test ItemPack's constructors: pass

Test ItemPack has only 2 declared methods: pass

Test ItemPack's addItem(): pass

Test ItemPack's getItems(): pass

C:\>
```

2. **[2 marks]** Create interface `Biodegradable` according to the class diagram.

Expected output:

```
C:\> java BiodegradableTest
Test interface Biodegradable declaration: pass

Test Biodegradable has correct number of declared methods: pass

Test Biodegradable's getDaysToBiodegrade(): pass

C:\>
```

3. **[5 marks]** Create class `PaperBag` to match the class diagram. A paper bag takes 42 days to biodegrade.

Expected output:

```
C:\> java PaperBagTest
Test class PaperBag declaration: pass

Test PaperBag's constructors: pass

Test PaperBag has correct number of declared methods: pass

Test PaperBag's getDaysToBiodegrade(): pass

C:\>
```

4. **[5 marks]** Update class `Person` such that it is a Comparable.

   a. A person compares against another by the sum of the prices of all his non-depreciating valuable items in descending order. For example, person A compares against person B:
      - If A's sum of the prices is more than B's sum, A is less than B.
      - If A's sum of the prices is less than B's sum, A is more than B.
      - If A's sum of the prices is equal to B's sum, both are the same.

b. Code for class `Person` should not require changes when new sub classes of `Item`, `Valuable`, `ValuableItem` and `Depreciating` are added in the future.
c. You may add your own private methods to solve this question.

Expected output:

```
C:\>java PersonTest
Before sorting:
1. Adrian [grape, Mercedes ($90000.0), lime, Flat ($120000.0), BMW ($95000.0)]
2. Bryan [apple, Diamond Ring ($300.0), orange, Gold necklace ($500.0), papaya]
3. Cathy []
4. Devin [pear, Porsche ($110000.0), lemon, Villa ($200000.0), berry, Pearl
necklace ($3000.0), banana]

After sorting:
1. Devin [pear, Porsche ($110000.0), lemon, Villa ($200000.0), berry, Pearl
necklace ($3000.0), banana]
2. Adrian [grape, Mercedes ($90000.0), lime, Flat ($120000.0), BMW ($95000.0)]
3. Bryan [apple, Diamond Ring ($300.0), orange, Gold necklace ($500.0), papaya]
4. Cathy []

Result: Pass

C:\>
```

The test data used in class `PersonTest` are

- Adrian has grape, Mercedes ($90000.0), lime, **flat ($120000.0)**, BMW ($95000.0).
  - Total price of Adrian's non-depreciating valuables is $120000.0
- Bryan has apple, **diamond ring ($300.0)**, orange, **gold necklace ($500.0)**, papaya
  - Total price of Bryan's non-depreciating valuables is $800.0
- Cathy has nothing.
  - Total price of Cathy' non-depreciating valuables is $0.0
- Devin has pear, porsche ($110000.0), lemon, **villa ($200000.0)**, berry, **pearl necklace ($3000.0)**, banana.
  - Total price of Devin's non-depreciating valuables is $203000.0

Note, different sets of test data will be used for grading.

5. **[6 marks]** Create class `DepreciatingComparator` such that it is a `ValuableComparator` (see class diagram).

It overrides the `compare()` method to compare 2 `Valuable` objects according to the following rules in the order specified:

a. If both objects are `Depreciating` objects,
   a. Compare by their depreciating rate in ascending order.
   b. If their depreciating rates are different, return the result of comparison.
   c. If their depreciating rates are the same, continue with the following.
b. Non-`Depreciating` object is less than `Depreciating` object.
c. Otherwise, return the result of `ValuableComparator`'s `compare()` method.

Code for class `DepreciatingComparator` should not require changes when new sub classes of `Item`, `Valuable`, `ValuableItem` and `Depreciating` are added in the future.

Expected output:

```
C:\> java DepreciatingComparatorTest
=== Test 1 ===
Before sorting:
1. Gold necklace ($500.0)
2. Porsche ($110000.0) depreciates at 0.1
3. Flat ($120000.0)
4. BMW ($95000.0) depreciates at 0.2
5. Diamond Ring ($300.0)
6. Villa ($200000.0)
7. Mercedes ($90000.0) depreciates at 0.1
8. Pearl necklace ($3000.0)

After sorting:
1. Diamond Ring ($300.0)
2. Gold necklace ($500.0)
3. Pearl necklace ($3000.0)
4. Flat ($120000.0)
5. Villa ($200000.0)
6. Mercedes ($90000.0) depreciates at 0.1
7. Porsche ($110000.0) depreciates at 0.1
8. BMW ($95000.0) depreciates at 0.2

Result: Pass

=== Test 2 ===
Before sorting:
1. Porsche ($110000.0) depreciates at 0.1
2. Mercedes ($90000.0) depreciates at 0.1

After sorting:
1. Mercedes ($90000.0) depreciates at 0.1
2. Porsche ($110000.0) depreciates at 0.1

Result: Pass

=== Test 3 ===
Before sorting:
1. Villa ($200000.0)
2. Gold necklace ($500.0)

After sorting:
1. Gold necklace ($500.0)
2. Villa ($200000.0)

Result: Pass

C:\>
```

Question 2

Refer to the class diagram in appendix 2.

1. **[6 marks]** Implement the `StudentDAO`'s `read` method.

- If the data file does not exist, propagate any `FileNotFoundException` encountered.

- A sample of the file is as shown below.

```
Student,apple,F
SISStudent,orange,M,Java,Python,C#
```

  Each line of the file either describes a `Student` or a `SISStudent`.

  The `Student`'s format is:

```
Student,<name>,<gender>
```

  The `SISStudent`'s format is:

```
SISStudent,<name>,<gender>,<language 1>,<language 2>,...,<language N>
```

  An `SISStudent` will know 0 or more languages.

- The file is always of the correct format.

2. **[5 marks]** Each student's list of courses is loaded from another data file whose name is the student's name.

   A. For example, If the student's username is "`apple`", his list of courses is loaded from a data file named "`apple.txt`". A sample data file of items is as shown below.

```
COMM101:79
IS101:74
IS200:73
MGMT003:77
STAT101:59
```

   The format is as follows:

```
<CourseCode-1>:<CourseScore-1>
<CourseCode-2>:<CourseScore-2>
…
<CourseCode-N>:<CourseScore-N>
```

   - If the score of a course is not a number , ignore that course and continue to read the rest of the values. There will always be a value (i.e. not empty).
   - Do note that the Course's constructor requires a grade point value and not course score value (i.e. 3.7 instead of 80).
     Hint: Use the `getGradePoint` method in the `Utility` class.

   B. If the student's course data file is empty or does not exist, this means the student has yet to take any course.

   C. You may assume the data file is always of the correct format.
   You may add your own private methods to class `StudentDAO` to solve this question. Use `StudentDAOTest` to check your answer.

3. **[5 marks]** Implement `ResultGenerator`'s `output` method.

   A. The method should generate a data file of the following format in descending order of GPA then name:

```
<name1>:<Student|SISStudent>:<gpa1>
<name2>:<Student|SISStudent>:<gpa2>
```

   B. If the filename exists, this method will overwrite the existing file with the new content.

   C. If the file cannot be created, propagate any `FileNotFoundException` encountered.
   When `ResultGeneratorTest` is executed, a data file named "`result.txt`" with the following content is produced:

```
pear:SISStudent:3.9333333333333333
orange:SISStudent:3.6500000000000004
apple:SISStudent:2.72
ricky:Student:0.0
mary:Student:0.0
durian:SISStudent:0.0
```

END