Computer Science and Creative Technologies

Coursework Specification

Module Details

Module Code	UFCFC3-30-1
Module Title	Introduction to OO systems development
Module Leader	Jun Hong, Rakib Abdur
Module Tutors	James Lear, Kun Wei, Abdullahi Arobo, Stewart Green, Emmanuel Ogunshile, Barkha Javed, Rakib Abdur, Jun Hong
Year	2019-20
Component/Element Number	A
Total number of assessments for this module	One assessment
Weighting	This coursework is worth 100 marks representing 50% of your total course grade.
Element Description	This assignment is to be completed in groups of three students.

Dates

Date issued to students	October 31, 2019
Making results and feedback to students	January 17, 2020
Submission Date	December 9, 2019 Demo session in the same week (exact time slot TBA)
Submission Place	Blackboard
Submission Time	14:00
Submission Notes	Please note that all the members of a group must be present during the demonstration session, and all the members of a group must upload the required portfolio/documents to blackboard individually. That is, all the members of a group must upload the same piece of work as a zip file (perhaps, considering your group name, e.g., CP-01-group-1.zip) individually. Your zip file will contain: UML diagram, test cases, and the .java code. For example, if you draw your UML in a word file, then you can create a table in the same file to write the test cases. Then your zip file should contain that word/pdf file and the .java code. I recommend that, if you are drawing your UML diagram in word, then convert it into pdf before submission (this is because lower/upper case letter of Class name/attribute/method will be considered during marking).

Feedback

Feedback provision will be	On the spot verbal feedback during the demo session + as appropriate written feedback uploaded to Blackboard.

Contents

Section 1: Overview of Assessment

This assignment assesses the following module learning outcomes:

- Demonstrate knowledge of the object oriented (OO) paradigm by producing software solutions to simple problems.
- Design a simple OO system using UML class diagram (without using a formal tool)
- Implement and test a simple OO software system using a suitable Integrated Development Environment (Netbeans)
- Comprehend and explain object oriented programming concepts (question/answer during the demo session)
- Code re-use, apply good practice in code design/testing

The assignment is worth ***50%*** of the overall mark for the module.

Broadly speaking, the assignment requires you to review the ideas of superclass and subclass relationship and provide opportunities to examine your level of knowledge in basic hierarchical class design and object-oriented programming.

The assignment is described in more detail in section 2.

This is a **GROUP** assignment.

Working on this assignment will help you to understand more clearly the concepts, problems, and techniques of basic object-oriented programming, and how these can be used to design and implement a simple problem while working in a team.

If you have questions about this assignment, please contact/discuss with your lab tutors.

Section 2: Task Specification

Please read the following INSTRUCTION before starting your assignment.

This coursework consists of three parts. Completing only Part-I represents 15% of your total course grade, so you will only be awarded partial marks. Competing Part-II (which includes Part-I) represents 30% of your total course grade, and you will still be awarded partial marks. Completing Part-III (which includes Part-I and Part-II) represents full 50% of your total course grade.

If you complete Part-III, you do not have to demonstrate Part-I and Part-II separately.

If you complete Part-II, you do not have to demonstrate Part-I separately.

Part-I
(Only 15% of your total course grade)

Imagine a Car Parts and Accessories shop, which requires a software system to keep track of stock items and prices. The shop will sell different kinds of stock items. However, to start with, you have been tasked with designing and implementing a class called StockItem with the following properties.

- An instance (object) of the StockItem class represents a particular item which the shop sells, with a string representing *fixed stock code*, an integer representing *variable quantity in stock* and a double representing *variable price of the stocked item*.
- A constructor that creates a Stock Item with the specified quantity, price, and the *fixed stock code*.
- All the appropriate '*setters*' and '*getters*' methods, including a *getStockName()* method which returns the string "Unknown Stock Name" and a *getStockDescription()* method which returns the string "Unknown Stock Description".
- An *addStock()* method that increases the stock level by the given amount. If the value is less than one or the stock exceeds 100, a suitable error message should be printed.
- A *sellStock()* method that attempts to reduce the stock level by the given amount. If it is less than one, a suitable error message should be printed. If the amount is otherwise less than or equal to the stock level, then the reduction is successful and true is returned. Else there is no effect, but false is returned.
- A *getVAT()* method that returns the standard percentage VAT rate, e.g., you can use 17.5
- Appropriate '*setters*' method for price (without VAT) and '*getters*' methods for price with and without VAT
- A method named *toString()* that returns a string giving the stock code, the stock name, the description, the quantity in stock, the price before VAT and the price after VAT. It must use the appropriate methods above to obtain the stock name, description, quantity and prices.

Task 1.1. Design and draw the corresponding UML class diagram of the StockItem class described above (no UML modelling tool is required, you can just draw using a simple editor, e.g., MS Word).

Task 1.2. Code and Test it! (This is recommended and purely for your benefit! This will help you in producing some working code before proceeding to the next step in Part-II. However, you can directly proceed to Part-II.)

Implement the above class and test it with a program called TestStockItem. That is, your TestStockItem class will contain the main() method, which you have known and perhaps practiced many similar programs in the practical lab exercises.

You should create some instances of StockItem class, add stock, sell some stock and change the price, whilst printing out the items in between.

Testing is typically a part of the program development – you should use a test strategy to test your program thoroughly. You may look at practical exercises week 3 (Step 4), identify suitable test cases for the StockItem class, write and document them in the form of a table along with the UML class design file.

Test Case	Purpose	Expected result

An example **run** might be as follows.

```
Creating a stock with 10 units Unknown item, price 99.99 each, and item code W101
Printing item stock information:
Stock Type: Unknown Stock Name
Description: Unknown Stock Description
StockCode: W101
PriceWithoutVAT: 99.99
PriceWithVAT: 117.48825
Total unit in stock: 10
Increasing 10 more units
Printing item stock information:
Stock Type: Unknown Stock Name
Description: Unknown Stock Description
StockCode: W101
PriceWithoutVAT: 99.99
PriceWithVAT: 117.48825
Total unit in stock: 20
Sold 2 units
Printing item stock information:
Stock Type: Unknown Stock Name
Description: Unknown Stock Description
StockCode: W101
PriceWithoutVAT: 99.99
PriceWithVAT: 117.48825
Total unit in stock: 18
Set new price 100.99 per unit
Printing item stock information:
Stock Type: Unknown Stock Name
```

Description: Unknown Stock Description
StockCode: W101
PriceWithoutVAT: 100.99
PriceWithVAT: 118.66324999999999
Total unit in stock: 18
Increasing 0 more units
The error was: Increased item must be greater than or equal to one

Part-II
(Only 30% of your total course grade)

The Car Parts and Accessories shop has got plenty of *GeoVision Sat Nav* navigation system at very competitive prices, which are going to be the first item on sale. You need to design and implement a class **NavSys** which is a sub-class of StockItem. A parameterised constructor of the NavSys class must call the StockItem's constructor using *super* to initialise the instance variables. The NavSys class will **override** the instance methods getStockName() and getStockDescription() with ones that **return** "Navigation system" and " GeoVision Sat Nav" respectively. NavSys class will also **override** the toString() method using the concept of *super* in Java.

Task 2.1. Revise your UML diagram in Task 1.1, to incorporate the **NavSys** class and show their relationship using appropriate UML notations.

Task 2.2. Implement the **NavSys** class and test this with a program called **TestNavSys** by creating an instance of **NavSys**, adding and then selling some navigation system stock and changing the price, whilst printing out the item in between.

Testing is typically a part of the program development – you should use a test strategy to test your program thoroughly. You may look at practical exercises week 3 (Step 4), identify suitable test cases for the StockItem class, write and document them in the form of a table along with the UML class design file.

Test Case	Purpose	Expected result

An example **run** might be as follows.

```
Creating a stock with 10 units Navigation system, price 99.99, and item code NS101
Printing item stock information:
Stock Type: Navigation system
Description: GeoVision Sat Nav
StockCode: NS101
PriceWithoutVAT: 99.99
PriceWithVAT: 117.48825
Total unit in stock: 10
Increasing 10 more units
Printing item stock information:
Stock Type: Navigation system
Description: GeoVision Sat Nav
StockCode: NS101
PriceWithoutVAT: 99.99
PriceWithVAT: 117.48825
Total unit in stock: 20
Sold 2 units
Printing item stock information:
Stock Type: Navigation system
```

Description: GeoVision Sat Nav
StockCode: NS101
PriceWithoutVAT: 99.99
PriceWithVAT: 117.48825
Total unit in stock: 18
Set new price 100.99 per unit
Printing item stock information:
Stock Type: Navigation system
Description: GeoVision Sat Nav
StockCode: NS101
PriceWithoutVAT: 100.99
PriceWithVAT: 118.66324999999999
Total unit in stock: 18
Increasing 0 more units
The error was: Increased item must be greater than or equal to one

Part-III
(Full 50% of your total course grade)

You now invent three more subclasses of the StockItem class, and explore **polymorphism** and **dynamic method binding**. Like **NavSys** class in Part-II, for each of these invented classes, design the appropriate constructors, get and set methods. You also need the toString method to print the information to the console.

Task 3.1. Revise your UML diagram in Task 2.1, to incorporate your newly invented classes and show their relationship using appropriate UML notations.

Task 3.2. Implement all these classes and write a program called **TestPolymorphism** which has a **class** method **itemInstance()** to test just one instance of a **StockItem** given to it as a method parameter. This will increase the stock, sell some stock and change the price, printing out the item in between. The class will also have a **main()** method which builds an array containing one instance of each of the three subclasses of StockItem you have written so far, and then, in a loop, calls the class method to test each one.

Hint. Fragment of code given below.

```
      .
      .
      .

class TestPolymorphism {

    public static void itemInstance(StockItem s){
         .
         .
         .
        System.out.println("Printing item stock information:");
        System.out.println(s);
         .
         .
         .

    }

    public static void main(String[] args) {

        StockItem [] s = new StockItem[4];
           .
           .
           .
   }
}
```

Testing is typically a part of the program development – you should use a test strategy to test your program thoroughly. You may look at practical exercises week 3 (Step 4), identify suitable test cases for the StockItem class, write and document them in the form of a table along with the UML class design file.

Test Case	Purpose	Expected result

Section 3: Deliverables

UML diagram, test cases, and the .java code.

Section 4: Marking Criteria

1. Class name, attribute, and method follow the standard naming convention correctly (class: mixed case with the first letter uppercase, attribute and method: mixed case with the first letter lowercase)

None	*Inadequate*	*Most*	*All*	
0 %	**5 %**	**7 %**	**10 %**	
☐	☐	☐	☐	=

2. Attributes of the class given along with their visibility and type

None	*Inadequate*	*Most*	*All*	
0 %	**1 %**	**3 %**	**5 %**	
☐	☐	☐	☐	=

3. Methods of the class given along with visibility, return type and parameters

None	*Inadequate*	*Most*	*All*	
0 %	**1 %**	**3 %**	**5 %**	
☐	☐	☐	☐	=

4. Suitable test cases have been identified and documented

None	*Inadequate*	*Most*	*All*	
0 %	**5 %**	**7 %**	**10 %**	
☐	☐	☐	☐	=

5. Appropriate variables and constant declared correctly?

None	*Inadequate*	*Most*	*All*	
0 %	**3 %**	**7 %**	**10 %**	
☐	☐	☐	☐	=

6. COMPLETED program compiles and runs

None	*All*	
0 %	**10 %**	
☐	☐	=

7. Program displays input and output messages correctly

None	*Inadequate*	*Most*	*All*	
0 %	**3 %**	**7 %**	**10 %**	
☐	☐	☐	☐	=

8. Overall group demonstration

Absent	*Poor (Did not know how to run the system)*	*Good (Able to run the system)*	*Excellent (Able to show additional/new ideas/codes)*	
0 %	**3 %**	**7 %**	**10 %**	
☐	☐	☐	☐	=
Group assessment percentage (out of 70%):				

9. **Individual Q&A**

Absent	*Inadequate (barely able to explain the codes and/or work done)*	*Good (good explanation of codes and/or work done)*	*Very Good (very good explanation of codes and/or work done)*	*Excellent (excellent explanation of codes and/or work done)*	
0 %	**7 %**	**15 %**	**20%**	**30 %**	

Section 5: Feedback mechanisms

On the spot verbal feedback during the demo session + as appropriate written feedback uploaded to Blackboard. Formative feedback provided in the In-class tests will be useful to complete the assignment.

This page is intentionally left blank