U08026 Further Object-Oriented Programming 2013–2014

Coursework 2

Value

30%

Mode

Individual work

Deadline

Sunday 8 December 2013 at 17:00 (end of Week 11). Upload to *Moodle*.

Learning outcomes

This coursework is designed to test your attainment of the following learning outcomes:

- Create and test a software artefact by applying the principles of object orientation such as inheritance and interfaces and language features such as exception handling.
- Evaluate the correctness and usefulness of a software system
- Analyse a complex problem, structure it, collect relevant information, consider options and recommend a course of action.
- Identify and utilise trustworthy information sources which provide programming-language specific reference material.
- Apply self-awareness in evaluating their impact in team-based work and utilise appropriate communication and problem resolution strategies.

Create software that meets the requirements of an international community of users.Introduction

For this assignment you will create an extended version of the Java program you wrote for coursework 1. You may reuse your work form that as much as you wish.

The specification is similar, except that now the user will type:

- an *item number* (a positive integer)
- a *quantity* or a *weight* depending on the sort of the purchase.

There are still *two distinct* sorts of purchases:

- those sold by *number* of components in a pack – an *integer* (*int*) value
- those sold by *weight*—a *real* (*double*) value

Your program must *look up* the item number and obtain its description, the unit price or price per Kg, and the pricing mode (i.e. whether it is priced per unit or per Kg) from a suitable data structure that has these recorded.

So for example if bananas was associated with the item number 5 then the user would be able to purchase 3kg of bananas by entering a 5 and a 3 at appropriate points in the user interface.

The user interface should be modified from that of coursework 1 accordingly.

Your man program should contain calls to the method *recordItem* (see later) to store information about at least six items, including at least two of each sort.

Example of reporting erroneous input

The program should report ill-formed user input and also the situation where an item number does not match any recorded item about which details are stored.

Non-functional requirements

1. **Team work:** Because we are seeking to assess your ability develop software that could be implemented by a team we **require** that there you must use store details about the items that a user can purchase using a class that provides the following methods:

```
public boolean isKnownItemNumber(int itemNumber) {
// pre: true
// post: returns true iff there is information stored for this itemNumber
… }
public void recordItem(int itemNumber,
        String description,
        double unitPrice,
        int sort) {
// pre: ! isKnownItemNumber(itemNumber)
 // && unitPrice > 0
// && (sort == 0 || sort == 1)   sort == 0 => by number, sort == 1 ==> by weight
// post:
// description unit price and sort now stored for item with this itemNumber
… }
public String getDescription(int itemNumber) {
// pre: isKnownItemNumber(itemNumber)
// post: returns description for item with this itemNumber
… }
public double getUnitPrice(int itemNumber) {
// pre: isKnownItemNumber(itemNumber)
// post: returns unit price for item with this itemNumber
```

```
… }
public int getSort(int itemNumber) {
// pre: isKnownItemNumber(itemNumber)
// post: returns sort (0 or 1) for item with this itemNumber
… }

}
```

This information does *not* have to be *persistent*. In other words it does not have to survive between runs of the program.

You should choose suitable data structure(s) for representing this information.

Note that this class stores details about the types of item a user *can* purchase, not about the items (s)he *has* purchased.

It **must** be possible for this class to be substituted by an implementation made by a different programmer, thus facilitating team working. For this reason you **must not** change this interface.

2. **Internationalisation**: The program **must** be *internationalised*. In other words there must be no English-language strings in the program source, for messages or button captions. However, there is no need to translate the descriptions of items. You should make reference to a *resource bundle* (see materials on Internationalisation for this module – Week 6). You need only produce a resource bundle for English, but it should be easy for another (non-programmer) team member to create a translated version for a different *locale*. You may assume that it will be for a left-to-right, alphabetic writing system.
3. Note that there will never be any objects of the class *Purchase*.
4. There is no need for mouse interaction in this exercise.
5. You should include any *assertions* you think will be useful in documenting your program and highlighting programming errors. These are particularly useful for *pre-conditions*.
6. Because we are seeking to assess your ability create graphical user interface by program statements we require you **not to use** any interactive tools for creating the interface.

What you have to submit (*Word* and *Zip*)

A *Word* (.doc or .docx) document showing:

1. a *screenshot* of your user interface in its initial state, before any user interaction has taken place
2. *screenshots* demonstrating reporting of erroneous input of an item number.
3. *screenshots* demonstrating reporting of erroneous input of a number of components
4. *screenshots* demonstrating reporting of erroneous input of a weight
5. *screenshots* demonstrating at least three items successfully recorded
6. the Java *source text* that you used to create your user interface
7. the Java *source text* of each of your classes for

    a. *purchase*
    b. *purchase sold by number*
    c. *purchase sold by weight*
8. an *explanation* of how you programmed the handling of erroneous input.
9. an *explanation* of the data structure(s) you used in the solution of this task.
10. a *reflection* on the degree of success you have achieved in this work (be honest!)

A *zip* file containing:

- all the source texts of your program

N.B. we want *two* files. The first one must be in *.doc* or *.docx* format and the second must be in *zip* format (not *rar*, or *7zip*). You can use *Open-* or *Libre Office* to create the first document, but you must save your work in *.doc* or *.docx* format.

Marking scheme

For the program text that creates the visual interface

4 marks

For the program text that makes the interface react to events

4 marks

For correct reporting of erroneous input

4 marks

For correct output resulting from acceptable input

4 marks

For the successful *internationalisation* of your program.

4 marks

For successful implementation of the class to store details of items.

4 marks

For the *explanation* of the data structure(s) you used in the solution of this task.

3 marks

For the reflection of how successful you were in this task

3 marks

= 30 marks

Appendix: Hints

Data structures: You can find suitable data structures in the Java *collection* classes.