U08026 Further Object-Oriented Programming 2013–2014

Coursework 1

Value

30%

Mode

Individual work

Deadline

Sunday 3 November 2013 at 17:00 (end of Week 6). 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.

Introduction

For this assignment you will create a Java program that maintains a graphical user interface for a *very* simple shopping application. The user inputs information about purchases and the program composes a list of details of those purchases.

There are *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

You should devise a program that presents a graphical user interface of three parts:

- a *bill area* where the details of purchases are displayed. Set the preferred size of this to be large enough. You may assume that it will sufficiently large for the text.
- a small *error-reporting area* for showing messages if input is not well formed or has an inappropriate value. You need not set any upper limit for number, weight or price.
- an *input area*, consisting of
  - a *button* to request *add by number*
  - a *button* to request *add by weight*
  - *a text field* into which the user types the *name* of the item
  - a text field into which the user types the *number*/*weight*
  - a text field into which the user types the *price* (per unit or per weight unit)

When the user presses the button corresponding to the appropriate mode (by number / by weight), the values are retrieved from the text fields. If they are erroneous then this is reported in the error-reporting area and if they are acceptable the details of the purchase are appended to the list that appears in the *bill area*.

Example of reporting erroneous input

If the user types a value that is not well formed (integer for number of components, real (double) number for weight, real (double) for price) or which does not make sense, a message is displayed in the error-reporting area. For example if the user has typed 'abc' in the area where a weight was expected and then has pressed the key for adding by weight, then a message appears in the error-reporting area.

See the *Appendix* for how to deal with such input.

Non-functional requirements

1. Because we are seeking to assess your ability to make use of object orientation we *require* you to define a class to represent a *purchase* and to define two subclasses for:
   purchase sold by *number*
   purchase sold by *weight*
   When the user presses one of the two *add* buttons an object of the corresponding subclass should be created.
2. Note that there will never be any objects of the class *Purchase*.
3. There is no need for mouse interaction in this exercise.
4. You should include any *assertions* you think will be useful in documenting your program and highlighting programming errors.
5. 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* 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 a number of components
3. *screenshots* demonstrating reporting of erroneous input of a weight
4. *screenshots* demonstrating reporting of erroneous input of a price
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 how *object orientation* was used in the solution of this task and how you decided what fields and methods should go in what class
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

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 explanation of you decided what fields and methods should go in what class

4 marks

For the explanation of what programming statements you used to handle erroneous input

4 marks

For the explanation of how *object orientation* was 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

Sub-panes

This user interface will need sub-panes. You can see an example of how to do this in the *stopwatch* example of Week 3.

Retrieving numbers from text fields

Open a *scanner* (from package *java.util.Scanner*) on the string that you get from the text field:

```
String priceString = priceField.getText();`
Scanner priceScanner = new Scanner(priceString);
```

You can use the *predicate hasNextDouble* (or *hasNextInt*) which guards a call of *nextDouble* (or respectively *nextInt*)

```
if (priceScanner.hasNextDouble()) {
    double unitPrice = priceScanner.nextDouble(); …
```

Alternatively you can *try* calling *nextDouble* (*nextInt)* and *catch* the exception thrown when the string is not a well-formed *double* (*int*).

Once you have the value you can check whether it is sensible (no upper limits needed).