Lab 1. Object-Oriented Design Revisited

Theme. In this lab, you will:

- recap on how to use eclipse
- practise string manipulations
- practise the use of Java interfaces to implement a flexible object-oriented design (OOD)
- practise the use of polymorphism in OOP
- perform error handling using exceptions

Key concepts: Java interface, polymorphism, exceptions, arrays and collections, iteration, package

Required file(s): lab1.zip

1 Introduction

N.B.: This lab sheet assumes the use of the IDE eclipse 4.4 (i.e. Luna) to develop your Java programs. You may use the latest version of eclipse, but please bear in mind that the detailed instructions shown in the lab sheets may differ.

If you are unfamiliar with using eclipse, you might like to consult the eclipse help system at http://help.eclipse.org/.

In eclipse, a workspace is the central hub where your data and project files reside. Within a workspace, you may keep your working projects. Within each project, you may create your own packages[1] to group your Java classes, interfaces, etc.

To ensure that we all start up from the same point, prepare your eclipse workspace as follows:

1. Create a folder named CS2310. From now on, all files relating to this module should be stored in this folder.
2. Create a sub-folder named Labs under CS2310. From now on, all lab files should be stored in this folder.
3. Start up eclipse.
4. In eclipse's File menu, select Switch Workspace ➟ Other...

   When the Workspace Launcher appears, select your CS2310/Labs/ folder using the Browse... button OR enter the following path at the Workspace prompt:

   CS2310/Labs/

N.B.: If you are using Microsoft Windows (as opposed to Linux), you will need to use back-slashes (i.e. \) rather than forward-slashes (/).

   If you do not want eclipse to keep asking you to a workspace to work in every time when you start it up, check the check-box Use this as default and do not ask me again.

[1] Theoretically, you do not have to explicitly specify to which package your Java class belongs. This will mean that your class is grouped under a so-called unnamed package. In even a medium size software system this practice can lead to a large unwieldy unnamed package which is difficult to maintain or modify. As a good software development practice, always use the package statement to group together related classes and interfaces.

5. When the `Welcome` window appears, select the icon `Workbench`.

6. In a web browser, download the archive `lab1.zip` from Blackboard (`http://vle.aston.ac.uk/`). Extract the contents of this archive (i.e. a directory named `seminars`) into your newly created `eclipse` workspace, i.e. the folder `CS2310/Labs/`.

7. Create a new *Java project* using the given **seminars** project.

Hint:

(a) In the `File` menu, select `New` ➠ `Java Project`.

(b) When the `New Java Project` window appears:

- Uncheck the `Use default location` checkbox.
- Use the `Browse` button to invoke the file chooser.
- Navigate to the `seminars` folder in your `eclipse` workspace, i.e. the folder `CS2310/Labs/`, and click `OK`.
- Click on the `Finish` button.

If you encounter difficulty in creating a new Java project (e.g. a message window pops up and informing you of some transformation error), it could be because `eclipse` does not know which Java compiler to use for your project. In that case, check the **JDK Compliance** to ensure that `eclipse` is using Java 7.0 or above.
Also, to ensure that `eclipse` knows where to find a Java compiler on your computer, have a look at **Preferences** under the **Window** drop down menu. In the **Preferences** window, select:

Java ➠ Installed JREs

This should show you a table in the right hand side frame with *three* columns:

- **Name**
- **Location**
- **Type**

If the location of a Java compiler has been made known to `eclipse`, you should find *at least one record* in this table which specifies the location of the Java compiler (e.g. `c:\jdk1.6.0` on Windows). On the left hand side of the name attribute of this record, there should also be a check-box which has been checked.

2 Your Tasks: Object-Oriented Programming Revisited

In a computer science conference, some participants cannot hear what the speakers say very well. To improve the hearing condition, the organisers decided to use a Java application to display each speaker's speech while they give their seminar. This system has a GUI front-end which displays the exact speech in text format of the current speaker as it is delivered. This GUI has a button for the participants to 'see' the next speech and another button for closing the application. The development of this Java application has begun, but it is not yet fully completed. Your task is to finish off the implementation of this application according to the following requirements.

Hint: The approximate locations where you are expected to add your Java code and relevant hints for accomplishing the tasks have been marked throughout the given Java programs. Look out for block comments that include a sequence of *four* exclamation marks, i.e.:

```
/* !!!!  ...  ...  */
```

1. There should be a **realisation** relation[2] between the class `Dog` and the interface `Speaker`.
   Modify the `Dog` class header accordingly to reflect this relationship.

2. Complete the `speak` method for class `Dog`.

   A dog can speak like a human being(!). However, every now and then the old speaking habits creep into its speech. Thus, after a dog has uttered *five* words, it will automatically utter a "Woof!". Hence, if a dog wants to say "*My name is Super Dog. Thank you for listening to me.*", you will hear "*My name is Super Dog. Woof! Thank you for listening to Woof! me.*".

3. There should also be a **realisation** relation between the class `Philosopher` and the interface `Speaker`.
   Modify the `Philosopher` class accordingly to reflect this relationship.

> **Hint**: When a class implements an interface, it inherits *all* abstract methods from the interface.

   Philosophers like to clear their throats when they make a speech. Every time they clear their throats, an "Ah-Hem!" is uttered. Hence, when a philosopher delivers a speech, "Ah-Hem!" is scattered throughout the speech in a *random* manner. For example, if a philosopher wants to say "*Today is a sunny day. We are going to look at how the global climate affects our lives.*", you may hear "*Today is a sunny day. We are going to look at how Ah-Hem! the global climate affects our lives.*" or "*Today Ah-Hem! is a sunny day. We Ah-Hem! are going to look at how the global climate Ah-Hem! affects our lives.*".

4. The constructor `Conference(String)` in the `Conference` class is responsible for creating `Conference` objects. This constructor reads seminar records from a plain text data file using a `java.util.Scanner` object. For each seminar record, a `Seminar` object is created accordingly. Each seminar record is expected to occupy one line in the data file and it has four attributes:

   - the title of the seminar
   - the content of the speech
   - the name of the speaker
   - the speaker type, represented as an integer
     (e.g. 0 means that it is a dog and 1 refers to a philosopher)

   Each attribute is separated by a **Tab** character (i.e. the character `'\t'` in Java).
   During the creation of a `Conference` object, if the input data file does not conform to the expected data format (e.g. a seminar record has two attributes only), the constructor should throw a `BadDataFormatException` exception. The exception should contain an appropriate error message.

   Add suitable error check(s) to this constructor.

3 Testing

Now test your implementation to see if it meets the above requirements. You will find a plain text file named `seminars.txt` in your `seminars` folder. This file contains two seminar records. You may use it for testing your application.

[2] When a class implements an interface, there exists a *realisation* relation between them.

Hint: To run a Java application within `eclipse`:

1. Select the top level class of this Java application (i.e. `GUIConference.java`) within the `Package Explorer` tab.
2. Select from the pull-down menu: `Run` ➡ `Run Configurations...`
   A new window titled "Run Configurations" will then pop up.
3. On the left hand side of the "Run Configurations" window, there is a navigation menu with the item `Java Application`. If the name of the high level class (i.e. `GUIConference`) has not yet appeared as a sub-item underneath `Java Application`, double-click on the item `Java Application`. This will add the previously selected class to the list of runnable Java applications known by `eclipse`.
4. Select the required runnable Java application (i.e. `GUIConference`).
5. `GUIConference` requires the name of a seminar data file as its runtime argument. To specify the runtime argument:
   (a) On the right hand side of the "Run Configurations" window, select the tab titled `(x)= Arguments`.
   (b) This Java program takes one program argument, which is the name of the data files for seminar details.
   Select the `Arguments` tab. In the `Program arguments:` textarea, enter the desirable program argument, e.g.: `seminars.txt` or `badData.txt`.
   (c) Press the button "Apply", when finished entering the program argument(s).
6. Press the button "Run", when you are ready to run the Java application.

- When a dog gives a seminar, does its speech contain a "Woof!" after every *five* words?
- When a philosopher gives a seminar, does the speech contain "Ah-Hem!" at random points? Thus, if you re-run the conference and display the same speech by the philosopher again, does the throat-clearing sound now appear at different points in the speech?
- When inputting a data file (e.g. the data file `badData.txt`) which does not conform to the expected data format, does the application pop up a message window to alert the bad data format error?

4 Further Challenges

After you have finished all of the above tasks, if you would like to take on further challenges, do the following:

1. Design and implement a different speaker type that has similar capabilities to a `Dog` or a `Philosopher`, but speaks with a different verbal quirk.
2. Modify the application to enable a speech to be given by two speakers, with each speaker having different characteristics.