Inf2C Software Engineering 2017-18

Tutorial 3 (Week 8)

Notes on Answers

1 Introduction

2 The Observer pattern

See Figure 1 for a class diagram, illustrating the use of the lollipop notation for interfaces. This notation is good for reducing clutter when several classes implement multiple interfaces. Here its use does not make that much difference to the readability of the diagram. If it were not used, there would be a solid association arrow from the JButton box to the ActionListener box and a dashed arrow with an open triangle arrowhead from ButtonDemo box to the ActionListener box, indicating that the ButtonDemo class *realises* (UML terminology) or *implements* (Java terminology) the ActionListener interface.

See Figure 2 for a sequence diagram, just showing the constructor and method calls relevant to the Observer pattern.

Answers to questions raised on the tutorial sheet:

1. There is a directed association from the JButton object to the ActionListener interface.

2.
   - The Wikipedia page confusingly currently gives 2 different UML class diagrams. The first has a static association called subject. The second shows no way for the observers to learn anything about subjects' state or the messages they send to the observers.
   - The Design Patterns book shows a static association called model.
   - In the JButton demo, the actionPerformed() method of the ActionListener interface carries an ActionEvent object. Also the b1, b2 and b3 fields provide a way for a ButtonDemo object to directly interrogate the states of JButton objects.

3. Yes. The association from the ButtonDemo class to the JButton class. More generally, there might be method in some third class that is responsible for registering action listeners such as ButtonDemo with the JButton objects.

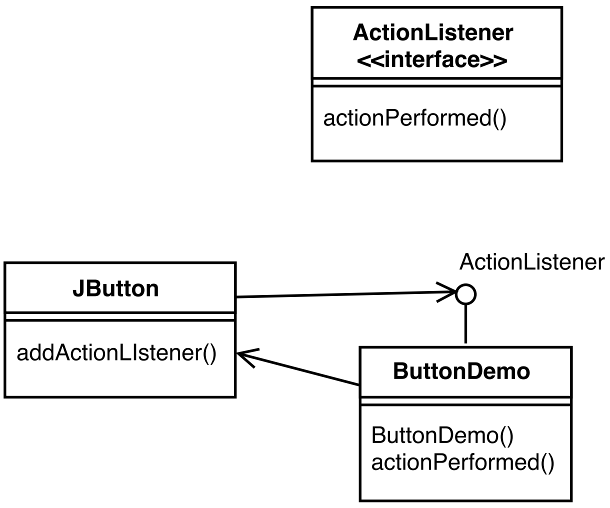

Figure 1: Button demo class diagram

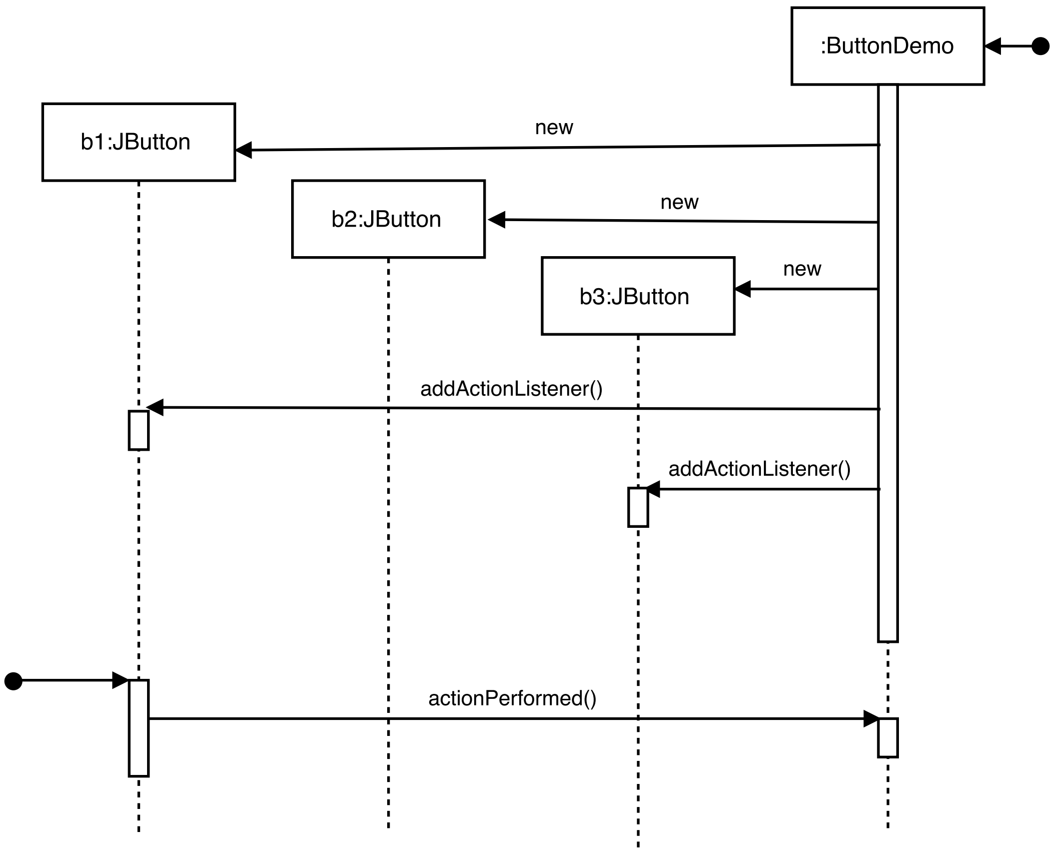

Figure 2: Button demo sequence diagram

4. Yes. The obvious one is that, in both Wikipedia and the book, the Observer is an abstract class, whereas in the implementation it is an interface. Conceptually, even using an interface for the Observer concept is somewhat heavy weight, as at heart what is registered with the Subject is a function (often called a *call-back* function because of the way in which it is used).

3 Design patterns in general

1. Yes we try to avoid duplication, and yes the application is of a design pattern is necessarily the duplication of some logic. It may not be a direct copy-and-paste, the duplication may be of the more abstract logic than of the actual concrete syntax used, but it is duplication nevertheless. However, the design pattern has arisen because the authors see no obvious way to write down the solution in a resuable/parameterised way. That is why the problem is a problem in the first place.

2. One obvious benefit is that you are able to draw on the thoughts of other expert software developers regarding the general advantages and disadvantages to the use of each particular design pattern use and apply those to your current situation.

3. Well, it may well be that a particular design pattern is subsumed by some language feature in another language. However, if the language you are currently using does not support that feature, then you need a plan for what you are doing *now*.

   The use of a particular design pattern may well be the best approach. You might consider switching implementation languages if that is an option, or you might consider proactively contributing some effort towards the introduction of that particular feature for the programming language that you are using. Though that will likely not affect your current course of action. It may be that such a feature is simply infeasible to be included into the particular language you have chosen.

   Finally, even if some particular design patterns are indeed subsumed by existing language features, this does not invalidate the *concept* of a design pattern. The concept can be thought of as capturing a recurring solution *for which* there is no current language feature. Whatever your opinion of the existing one, you would need a very strong opinion of the expressivity of abstraction of some programming to claim that it could never benefit from such a concept.

Paul Jackson. 10 Nov 2017.