

Inf2C tutorial SE3: Design

1 Introduction

This tutorial encourages you to look in depth at a simple design pattern, considering the design issues that arise both in deciding whether or not to use the pattern, and in deciding what variant to use.

First, recall the design principles discussed in the course so far. (List the ones you can remember.)

Next, consult the Wikipedia entry on the Singleton design pattern. (Wikipedia articles can change without notice. There is a copy of the way it was when I wrote this on the Schedule page: consult this if you spot a mismatch between what this tutorial sheet expects the article to say and what it actually does say when you look.)

Read the opening section, down to **Common uses**, and discuss it. Specifically, discuss:

- Why might the designer wish to *ensure* that there could only be one instance of a given class, rather than simply creating the class as usual and only creating one instance?
- Conversely, what disadvantages might there be to restricting the system to have only one instance where this is not really required?
- What do you think an “anti-pattern” is, and what do you think of the idea? Is it useful to identify anti-patterns? (This is a question with no unique right answer: at least one book of anti-patterns has been published and has sold well and been praised, but equally, many people think one shouldn’t study them.)

Next read the **Common uses** section, and check that you understand the point about why Singletons are often preferred to global variables. (Later, if time permits, come back and discuss the other bullet points here: you will need to have someone in the group who has looked up the other patterns mentioned.)

2 Singletons in Java and UML

Before you look at the Java code in the article: write the Java code for a class Catalogue which will only ever be able to have one instance. Make sure you provide some means for any client to access the unique instance (since a client cannot be allowed just to create its own instance...).

You may find it helpful to consult the UML class diagram in the article, although it uses some UML notation we did not cover in the lecture. What do you think the underlining of an attribute and a method in the class means? Once you’ve worked it out, do you think this notation is consistent with the uses of underlining that you have seen in the class? Why?

Next, read the Java implementations in the article. Compare them with each other and with the simple Java code you wrote above. Check that you understand the Java keywords `private`, `protected`, `static`, `final`, `enum` and their roles here; also that you understand the use of inner classes.

Extensions for after the tutorial

After the tutorial, follow some of the links in the Wikipedia article to explore some of these issues further. See how many of the code examples in different languages you can understand, looking things up as necessary. Have a look at the discussions on the Talk page.

3 Sequence diagrams

Suppose that there is a singleton class `Catalogue` in your library application, whose instance methods include an operation `void addCopy(b:Book)`. This creates a new `Copy` object and inserts it into a collection of references to `Copy` objects held by the `Book b`. (It probably also updates its own internal state, but you need not model this.)

Draw a sequence diagram which shows an actor sending a message `addCopy(b)` to the instance of class `Catalogue`, and shows what this object will do in response. Take care over activity bars, return arrows, and the placement of objects. Notice where you have to make design decisions.

Extensions, possibly for after the tutorial

What are you abstracting away or over-simplifying? In other words, what else would you need to know or decide before you could program this behaviour, or how does this application differ from a real object-oriented library catalogue system?

Suppose that, instead of assuming that our actor already knew how to address the singleton `Catalogue` object, we had wanted to model another object getting access to the singleton `Catalogue` object in the first place. What new issues would that have brought up? Try it.