Design Patterns: Introduction

What are design patterns?

In a nutshell, they describe generic solutions to software design problems. Originally defined in the 90s in the book “Design Patterns”, they have established themselves as commonly used ways of solving a problem. (see here for a description of the famous book: https://en.wikipedia.org/wiki/Design_Patterns)

There are different types of patterns

Patterns fall under one of three categories. They are either

- Creational : patterns which deal with object creation (using the keyword `new`)
- Structural: patterns which combine classes and objects to create larger structures (and choosing between inheritance or composition)
- Behavioural: patterns which show how objects can communicate

Common principles in design patterns

Experience has shown that some object-oriented approaches are more flexible than others. Here’s a summary of the main principles (taken from “Java Design Patterns Essentials”):

1. Program to an interface, not an implementation: here “interface” means the general concept of abstraction, so it can refer to a Java interface or an abstract class. Use the most general type (an interface) when declaring variables, constructor and method arguments, etc. Then your code becomes more flexible.

2. Prefer object composition over inheritance: Using inheritance isn’t always such a good idea, because you force your code to implement certain behaviour. Sometimes the “has a”-relationship is much more flexible than using a hierarchy.

3. Keep objects loosely-coupled: Classes should model only one thing. Don’t put too much into one class. Again, the idea is to keep the code flexible (ie. You can re-use it in another context and you are therefore faster in coding). The Observer pattern is a good example.

However, keep in mind, that design patterns are just another set of tools in your tool-kit. Sometimes the solution is easier than just implementing a design pattern….