Design patterns

◇ A design pattern is a way of reusing abstract knowledge about a problem and its solution.

◇ A pattern is a description of the problem and the essence of its solution.

◇ It should be sufficiently abstract to be reused in different settings.

◇ Pattern descriptions usually make use of object-oriented characteristics such as inheritance and polymorphism.

Pattern elements

◇ Name
  ▪ A meaningful pattern identifier.

◇ Problem description.

◇ Solution description.
  ▪ Not a concrete design but a template for a design solution that can be instantiated in different ways.

◇ Consequences
  ▪ The results and trade-offs of applying the pattern.

The Observer pattern

◇ Name
  ▪ Observer.

◇ Description
  ▪ Separates the display of object state from the object itself.

◇ Problem description
  ▪ Used when multiple displays of state are needed.

◇ Solution description
  ▪ See slide with UML description.

◇ Consequences
  ▪ Optimisations to enhance display performance are impractical.
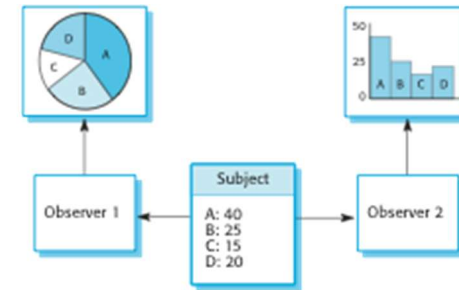
The Observer pattern (1)

Pattern name	Observer
Description	Separates the display of the state of an object from the object itself and allows alternative displays to be provided. When the object state changes, all displays are automatically notified and updated to reflect the change.
Problem description	In many situations, you have to provide multiple displays of state information, such as a graphical display and a tabular display. Not all of these may be known when the information is specified. All alternative presentations should support interaction and, when the state is changed, all displays must be updated. This pattern may be used in all situations where more than one display format for state information is required and where it is not necessary for the object that maintains the state information to know about the specific display formats used.

The Observer pattern (2)

Pattern name	Observer
Solution description	This involves two abstract objects, Subject and Observer, and two concrete objects, ConcreteSubject and ConcreteObject, which inherit the attributes of the related abstract objects. The abstract objects include general operations that are applicable in all situations. The state to be displayed is maintained in ConcreteSubject, which inherits operations from Subject allowing it to add and remove Observers (each observer corresponds to a display) and to issue a notification when the state has changed. The ConcreteObserver maintains a copy of the state of ConcreteSubject and implements the Update() interface of Observer that allows these copies to be kept in step. The ConcreteObserver automatically displays the state and reflects changes whenever the state is updated.
Consequences	The subject only knows the abstract Observer and does not know details of the concrete class. Therefore there is minimal coupling between these objects. Because of this lack of knowledge, optimizations that enhance display performance are impractical. Changes to the subject may cause a set of linked updates to observers to be generated, some of which may not be necessary.

Multiple displays using the Observer pattern

Design problems

✧ To use patterns in your design, you need to recognize that any design problem you are facing may have an associated pattern that can be applied.

- Tell several objects that the state of some other object has changed (Observer pattern).
- Tidy up the interfaces to a number of related objects that have often been developed incrementally (Façade pattern).
- Provide a standard way of accessing the elements in a collection, irrespective of how that collection is implemented (Iterator pattern).
- Allow for the possibility of extending the functionality of an existing class at run-time (Decorator pattern).

Reuse

✧ From the 1960s to the 1990s, most new software was developed from scratch, by writing all code in a high-level programming language.

- The only significant reuse or software was the reuse of functions and objects in programming language libraries.

✧ Costs and schedule pressure mean that this approach became increasingly unviable, especially for commercial and Internet-based systems.

✧ An approach to development based around the reuse of existing software emerged and is now generally used for business and scientific software.

Reuse levels

✧ The abstraction level
  ▪ At this level, you don't reuse software directly but use knowledge of successful abstractions in the design of your software.

✧ The object level
  ▪ At this level, you directly reuse objects from a library rather than writing the code yourself.

✧ The component level
  ▪ Components are collections of objects and object classes that you reuse in application systems.

✧ The system level
  ▪ At this level, you reuse entire application systems.

Configuration management

✧ Configuration management is the name given to the general process of managing a changing software system.

✧ The aim of configuration management is to support the system integration process so that all developers can access the project code and documents in a controlled way, find out what changes have been made, and compile and link components to create a system.

✧ See also Chapter 25.

Configuration management activities

✧ Version management, where support is provided to keep track of the different versions of software components. Version management systems include facilities to coordinate development by several programmers.

✧ System integration, where support is provided to help developers define what versions of components are used to create each version of a system. This description is then used to build a system automatically by compiling and linking the required components.

✧ Problem tracking, where support is provided to allow users to report bugs and other problems, and to allow all developers to see who is working on these problems and when they are fixed.

Host-target development

✧ Most software is developed on one computer (the host), but runs on a separate machine (the target).

✧ More generally, we can talk about a development platform and an execution platform.
  ▪ A platform is more than just hardware.
  ▪ It includes the installed operating system plus other supporting software such as a database management system or, for development platforms, an interactive development environment.

Development platform tools

✧ An integrated compiler and syntax-directed editing system that allows you to create, edit and compile code.

✧ A language debugging system.

✧ Graphical editing tools, such as tools to edit UML models.

✧ Testing tools, such as Junit that can automatically run a set of tests on a new version of a program.

✧ Project support tools that help you organize the code for different development projects.

Integrated development environments (IDEs)

✧ Software development tools are often grouped to create an integrated development environment (IDE).

✧ An IDE is a set of software tools that supports different aspects of software development, within some common framework and user interface.

✧ IDEs are created to support development in a specific programming language such as Java. The language IDE may be developed specially, or may be an instantiation of a general-purpose IDE, with specific language-support tools.

Open source development

✧ Open source development is an approach to software development in which the source code of a software system is published and volunteers are invited to participate in the development process

✧ Its roots are in the Free Software Foundation (www.fsf.org), which advocates that source code should not be proprietary but rather should always be available for users to examine and modify as they wish.

✧ Open source software extended this idea by using the Internet to recruit a much larger population of volunteer developers. Many of them are also users of the code.

Open source systems

✧ The best-known open source product is, of course, the Linux operating system which is widely used as a server system and, increasingly, as a desktop environment.

✧ Other important open source products are Java, the Apache web server and the mySQL database management system.

Open source business

✧ More and more product companies are using an open source approach to development.

✧ Their business model is not reliant on selling a software product but on selling support for that product.

✧ They believe that involving the open source community will allow software to be developed more cheaply, more quickly and will create a community of users for the software.

Open source licensing

✧ A fundamental principle of open-source development is that source code should be freely available, this does not mean that anyone can do as they wish with that code.

- ▪ Legally, the developer of the code (either a company or an individual) still owns the code. They can place restrictions on how it is used by including legally binding conditions in an open source software license.
- ▪ Some open source developers believe that if an open source component is used to develop a new system, then that system should also be open source.
- ▪ Others are willing to allow their code to be used without this restriction. The developed systems may be proprietary and sold as closed source systems.

License models

✧ The GNU General Public License (GPL). This is a so-called 'reciprocal' license that means that if you use open source software that is licensed under the GPL license, then you must make that software open source.

✧ The GNU Lesser General Public License (LGPL) is a variant of the GPL license where you can write components that link to open source code without having to publish the source of these components.

✧ The Berkley Standard Distribution (BSD) License. This is a non-reciprocal license, which means you are not obliged to re-publish any changes or modifications made to open source code. You can include the code in proprietary systems that are sold.

License management

✧ Establish a system for maintaining information about open-source components that are downloaded and used.

✧ Be aware of the different types of licenses and understand how a component is licensed before it is used.

✧ Be aware of evolution pathways for components.

✧ Educate people about open source.

✧ Have auditing systems in place.

✧ Participate in the open source community.