Week 7:
Object-Oriented Design

Nguyễn Thị Minh Tuyền

Adapted from slides of Ian Sommerville

Topics covered

1. Object-oriented design using the UML
2. Design patterns
3. Open source development

Design and implementation

- Software design and implementation is the stage in the software engineering process at which an executable software system is developed.
- Software design and implementation activities are invariably inter-leaved.
  - Software design is a creative activity in which you identify software components and their relationships, based on a customer's requirements.
  - Implementation is the process of realizing the design as a program.

Build or buy

- In a wide range of domains, it is now possible to buy off-the-shelf systems (COTS) that can be adapted and tailored to the users' requirements.
  - Example: if you want to implement a medical records system, you can buy a package that is already used in hospitals. It can be cheaper and faster to use this approach rather than developing a system in a conventional programming language.
- When you develop an application in this way, the design process becomes concerned with how to use the configuration features of that system to deliver the system requirements.

Topics covered

1. Object-oriented design using the UML
2. Design patterns
3. Open source development

Object-oriented development

- Object-oriented analysis (OOA), design (OOD) and programming (OOP) are related but distinct.
- OOA is concerned with developing an object model of the application domain.
- OOD is concerned with developing an object-oriented system model to implement requirements.
- OOP is concerned with realising an OOD using an OO programming language such as Java or C++.

Objects and object classes

- An **object** is an entity that has a state and a defined set of operations which operate on that state.
  - The state is represented as a set of object attributes.
  - The operations associated with the object provide services to other objects (clients) which request these services when some computation is required.
  - Objects are created according to some **object class** definition.
- An object class definition serves as a template for objects. It includes declarations of all the attributes and services which should be associated with an object of that class.

An OOD process

- Structured OOD processes involve designing object classes and relationship between these classes.
- Object-oriented systems are easier to change than systems developed using functional approaches.
  - Objects include both data and operations to manipulate that data.
  - They may therefore be understood and modified as stand-alone entities.
- Changing the implementation of an object or adding services should not affect other system objects.

Process stages

- To develop an OOD from concept to detailed, there are several things that you need to do:

- Define the context and modes of use of the system
- Design the system architecture
- Identify the principal system objects
- Develop design models
- Specify object interfaces

Process stages

To develop an OOD from concept to detailed, there are several things that you need to do:

- Define the context and modes of use of the system
- Design the system architecture
- Identify the principal system objects
- Develop design models
- Specify object interfaces

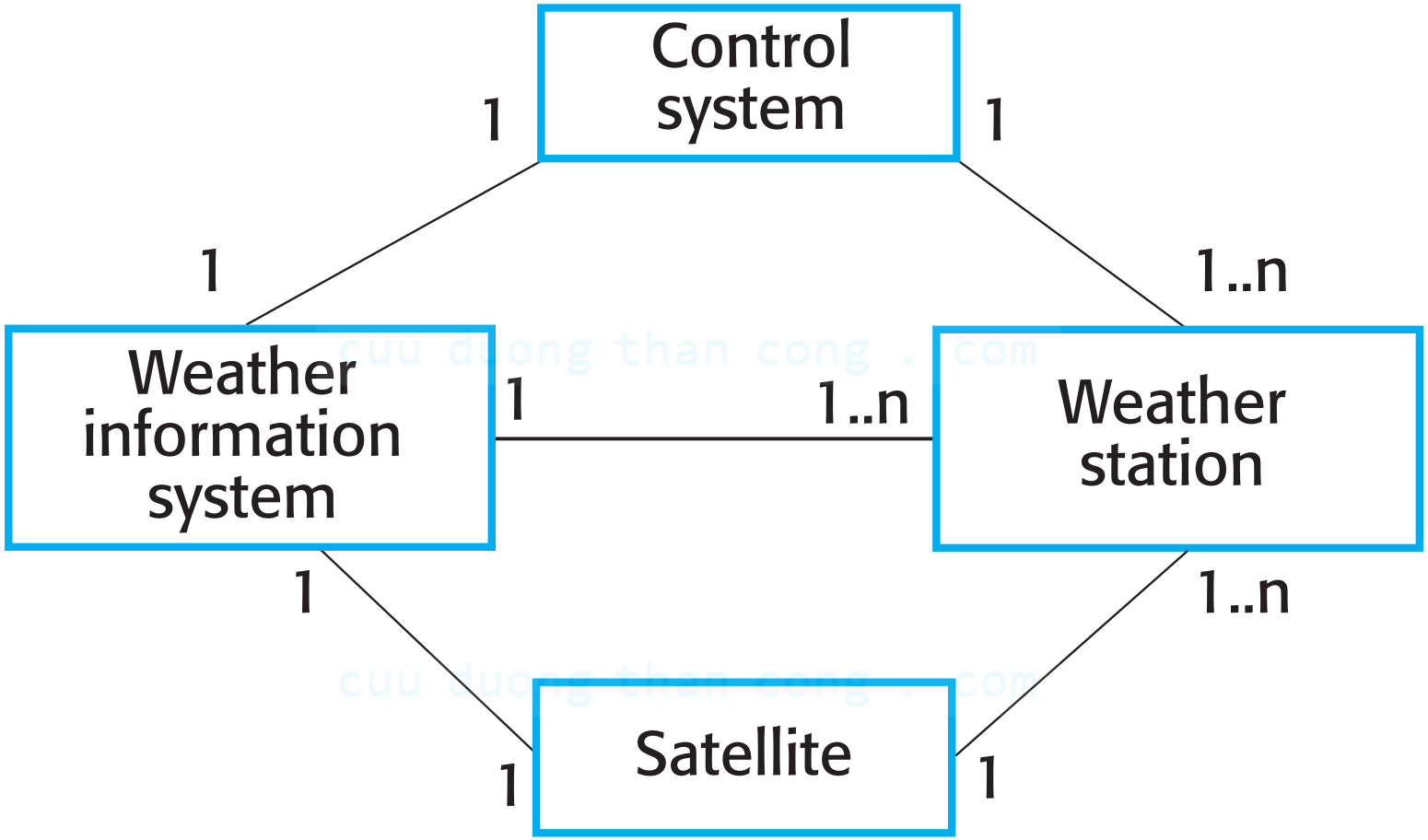
System context and interactions

- Understanding the relationships between the software that is being designed and its external environment is essential for deciding
  - how to provide the required system functionality and
  - how to structure the system to communicate with its environment.
- Understanding of the context also lets you establish the boundaries of the system. Setting the system boundaries helps you decide
  - what features are implemented in the system being designed and
  - what features are in other associated systems.

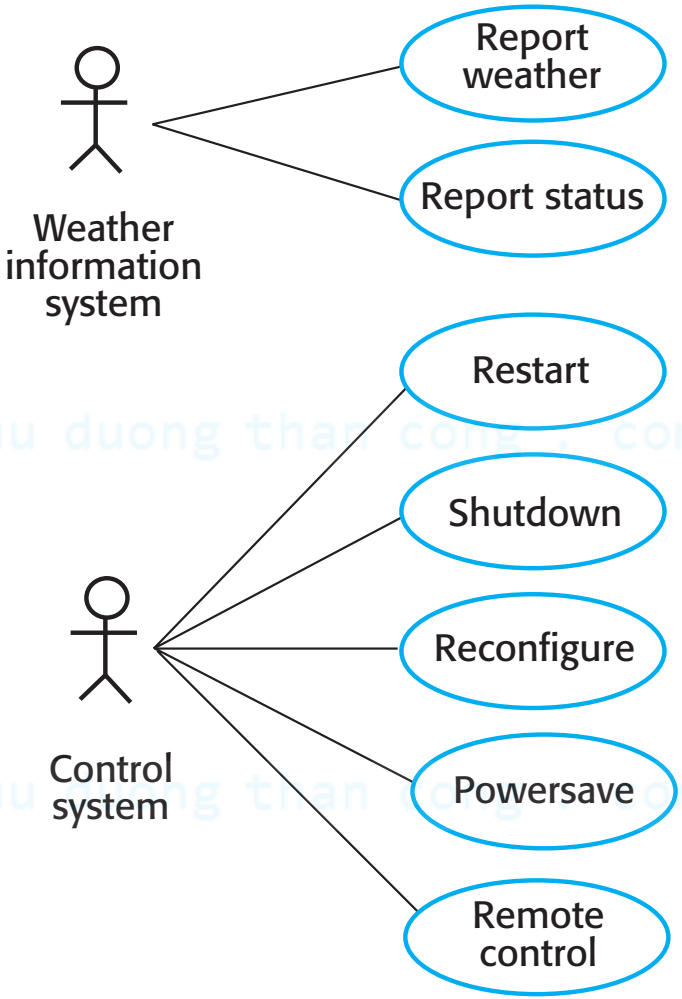
Context and interaction models

- System context
  - A **static model** that describes other systems in the environment.
  - Use a subsystem model to show other systems.
- Model of system use
  - A **dynamic model** that describes how the system interacts with its environment.
  - Use use-cases to show interactions

System context for the weather station



Weather station use cases



Use case description—Report weather

System	Weather station
Use case	Report weather
Actors	Weather information system, Weather station
Description	The weather station sends a summary of the weather data that has been collected from the instruments in the collection period to the weather information system. The data sent are the maximum, minimum, and average ground and air temperatures; the maximum, minimum, and average air pressures; the maximum, minimum, and average wind speeds; the total rainfall; and the wind direction as sampled at five-minute intervals.
Stimulus	The weather information system establishes a satellite communication link with the weather station and requests transmission of the data.
Response	The summarized data is sent to the weather information system.
Comments	Weather stations are usually asked to report once per hour but this frequency may differ from one station to another and may be modified in the future.

Process stages

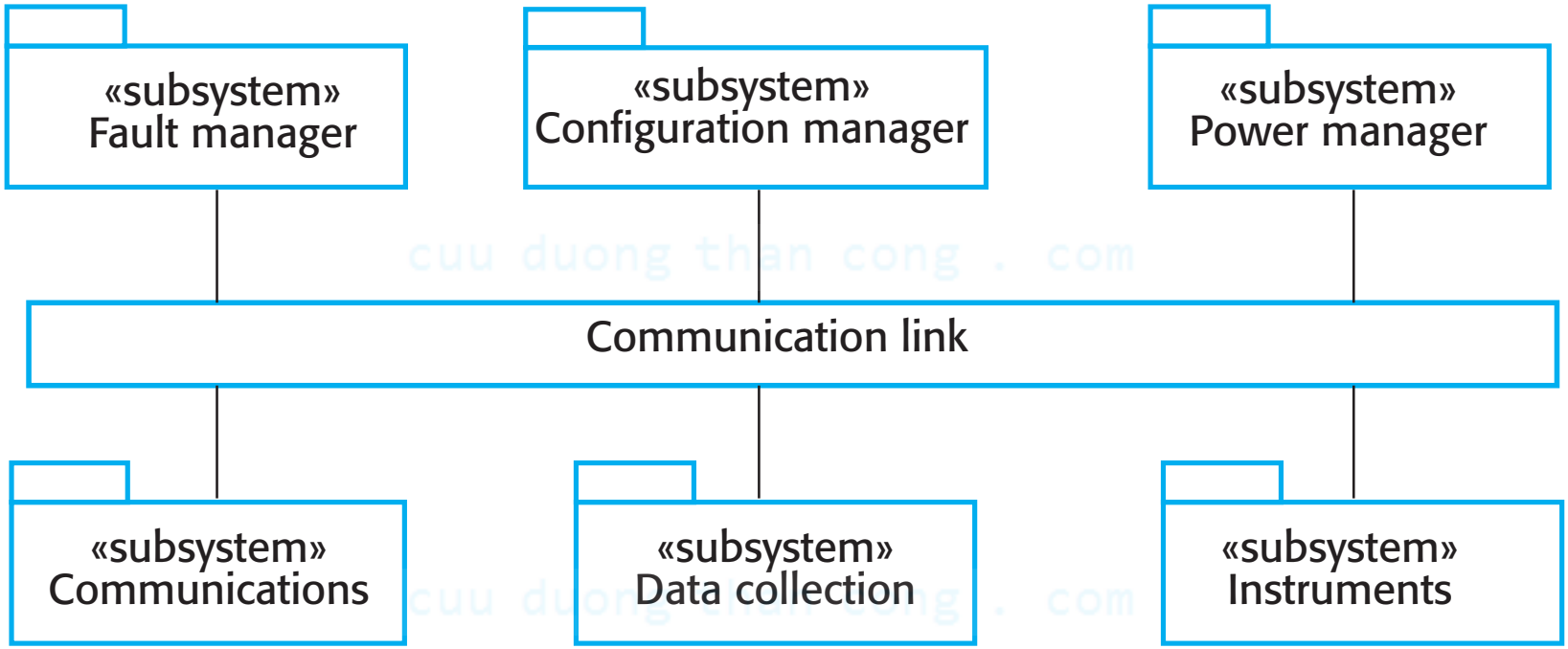
- To develop an OOD from concept to detailed, there are several things that you need to do:

- Define the context and modes of use of the system
- Design the system architecture
- Identify the principal system objects
- Develop design models
- Specify object interfaces

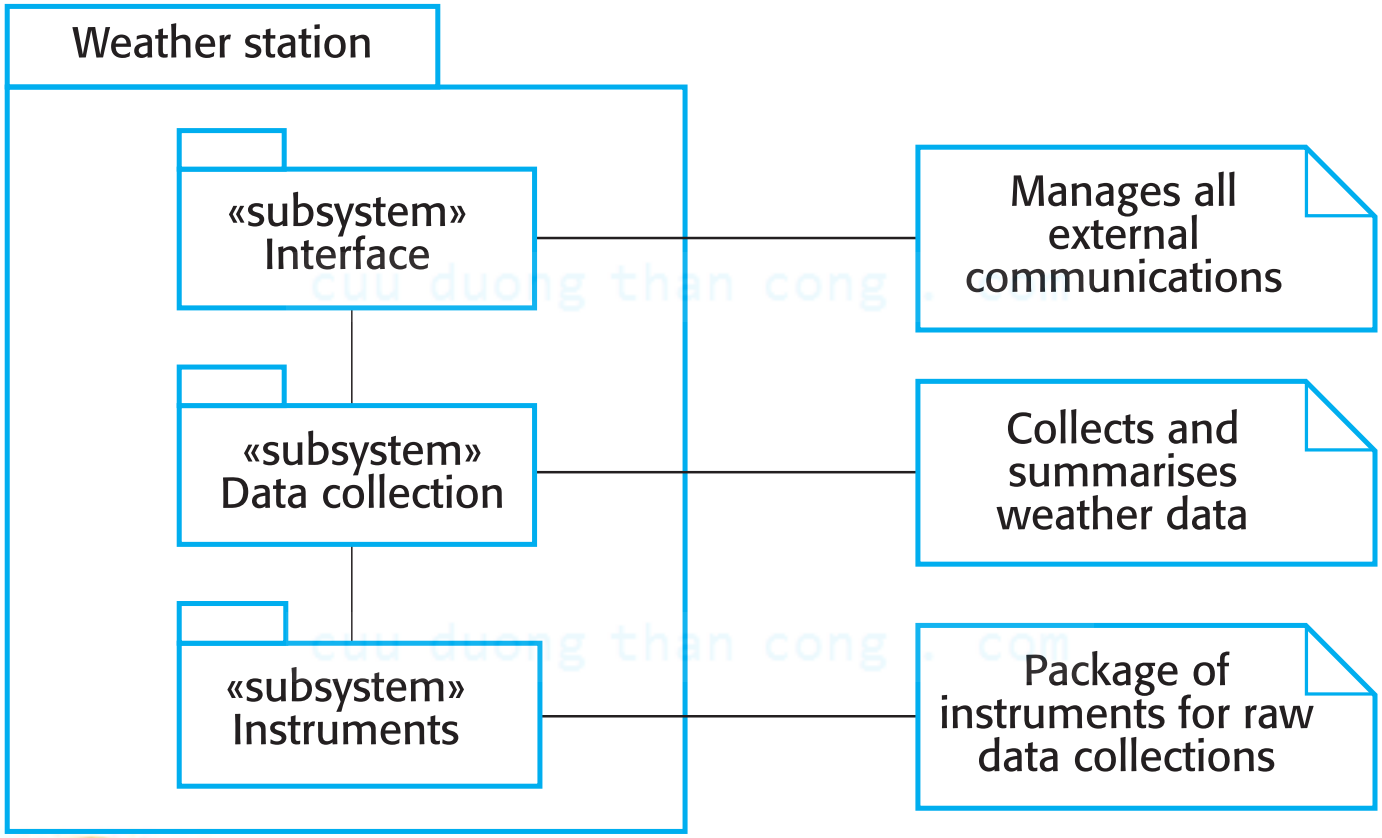
Architectural design

- Once interactions between the system and its environment have been understood, you use this information for designing the system architecture.
  - identify the major components that make up the system and their interactions, and
  - then organize the components using an architectural pattern such as a layered or client-server model.

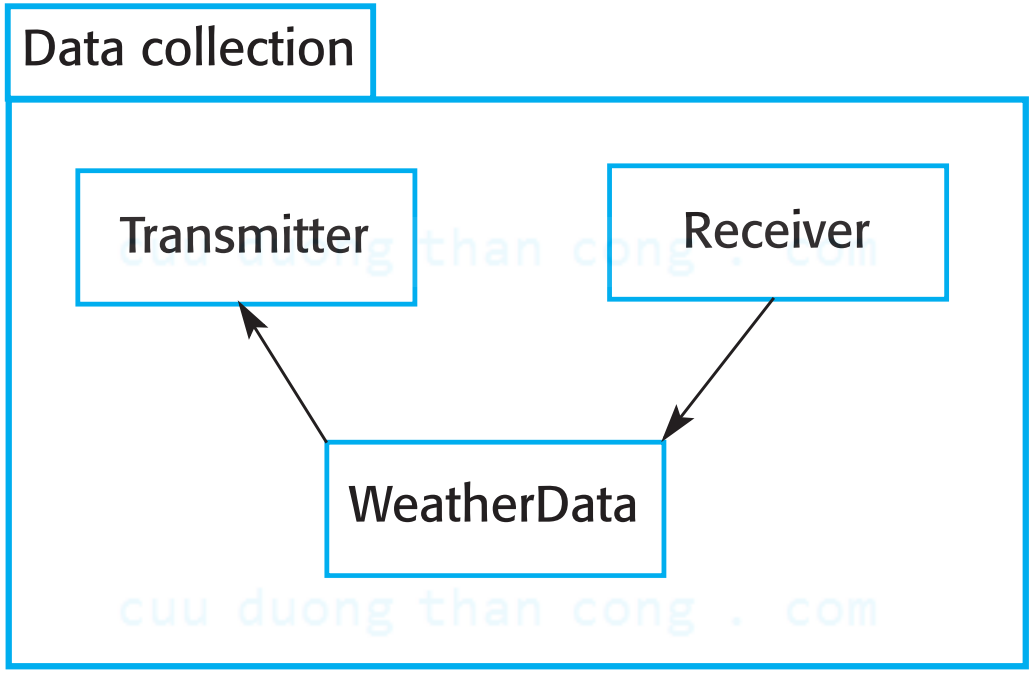
High-level architecture of the weather station



Weather station architecture



Architecture of data collection system



Process stages

- To develop an OOD from concept to detailed, there are several things that you need to do:

- Define the context and modes of use of the system
- Design the system architecture
- Identify the principal system objects
- Develop design models
- Specify object interfaces

Object class identification

- Identifying object classes is often a difficult part of object oriented design.
- There is no 'magic formula' for object identification.
  - It relies on the skill, experience and domain knowledge of system designers.
- Object identification is an iterative process. You are unlikely to get it right first time.

Use case description—Report weather

System	Weather station
Use case	Report weather
Actors	Weather information system, Weather station
Description	The weather station sends a summary of the **weather data** that has been collected from the instruments in the collection period to the weather information system. The data sent are the maximum, minimum, and average ground and air temperatures; the maximum, minimum, and average air pressures; the maximum, minimum, and average wind speeds; the total rainfall; and the wind direction as sampled at five-minute intervals.
Stimulus	The weather information system establishes a satellite communication link with the weather station and requests transmission of the data.
Response	The summarized data is sent to the weather information system.
Comments	Weather stations are usually asked to report once per hour but this frequency may differ from one station to another and may be modified in the future.

Approaches to identification

- Use a grammatical approach based on a natural language description of the system.
- Base the identification on tangible things in the application domain.
- Use a behavioural approach and identify objects based on what participates in what behaviour.
- Use a scenario-based analysis. The objects, attributes and methods in each scenario are identified.

Weather station description

A weather station is a package of software controlled instruments which collects data, performs some data processing and transmits this data for further processing. The instruments include air and ground thermometers, an anemometer, a wind vane, a barometer and a rain gauge. Data is collected periodically.

When a command is issued to transmit the weather data, the weather station processes and summarises the collected data. The summarised data is transmitted to the mapping computer when a request is received.

Weather station description

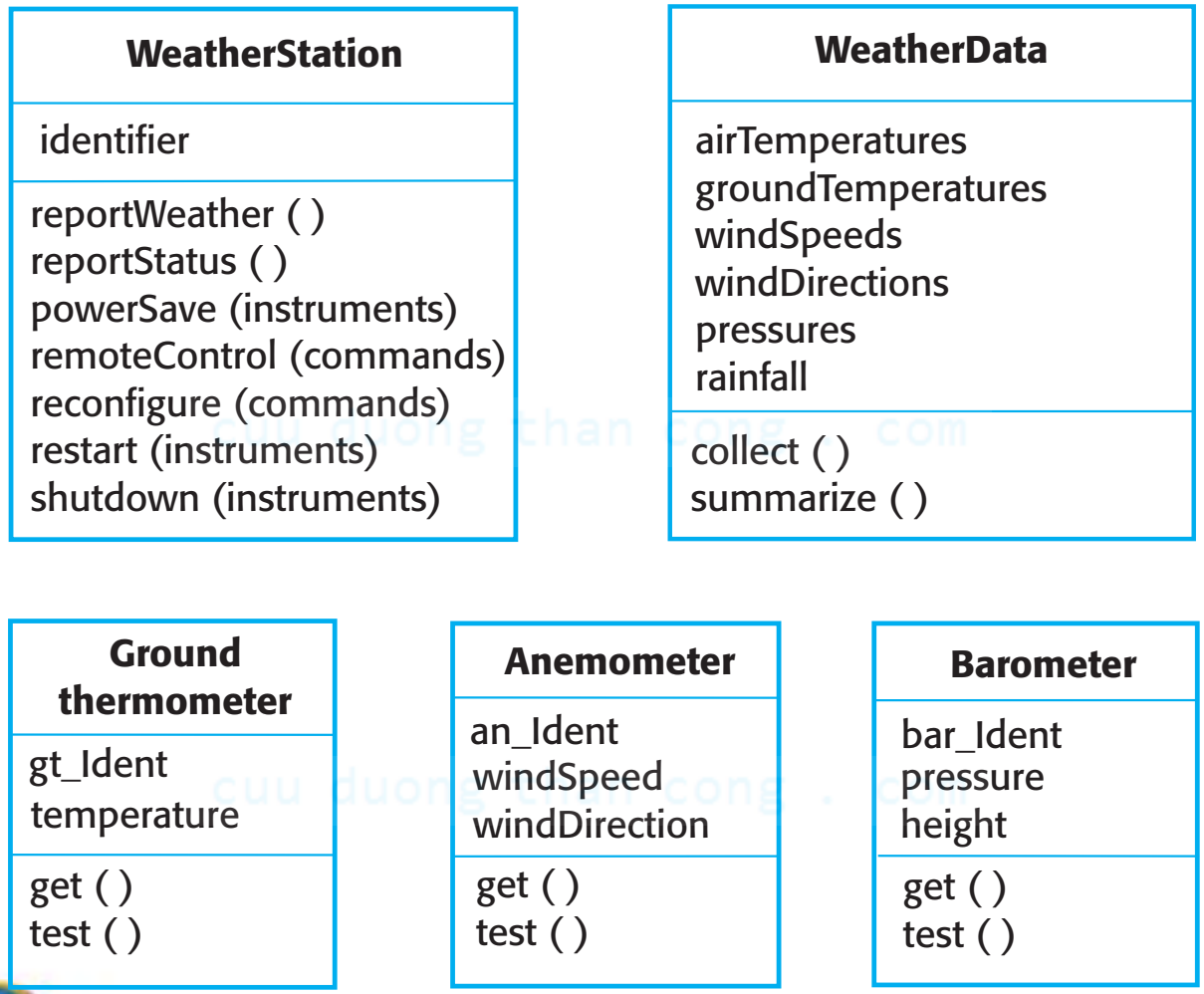
A weather station is a package of software controlled instruments which collects data, performs some data processing and transmits this data for further processing. The instruments include air and ground thermometers, an anemometer, a wind vane, a barometer and a rain gauge. Data is collected periodically.

When a command is issued to transmit the weather data, the weather station processes and summarises the collected data. The summarised data is transmitted to the mapping computer when a request is received.

Weather station object classes

- Object class identification in the weather station system may be based on the tangible hardware and data in the system:
  - Ground thermometer, Anemometer, Barometer, etc.
    - Application domain objects that are 'hardware' objects related to the instruments in the system.
  - Weather station
    - The basic interface of the weather station to its environment. It therefore reflects the interactions identified in the use-case model.
  - Weather data
    - Encapsulates the summarized data from the instruments.

Weather station object classes



Process stages

- To develop an OOD from concept to detailed, there are several things that you need to do:

- Define the context and modes of use of the system
- Design the system architecture
- Identify the principal system objects
- Develop design models
- Specify object interfaces

Design models

- Design models show
  - the objects or object classes in a system and
  - the relationships between these entities.
- Static models describe the static structure of the system in terms of object classes and relationships.
- Dynamic models describe the dynamic interactions between objects.

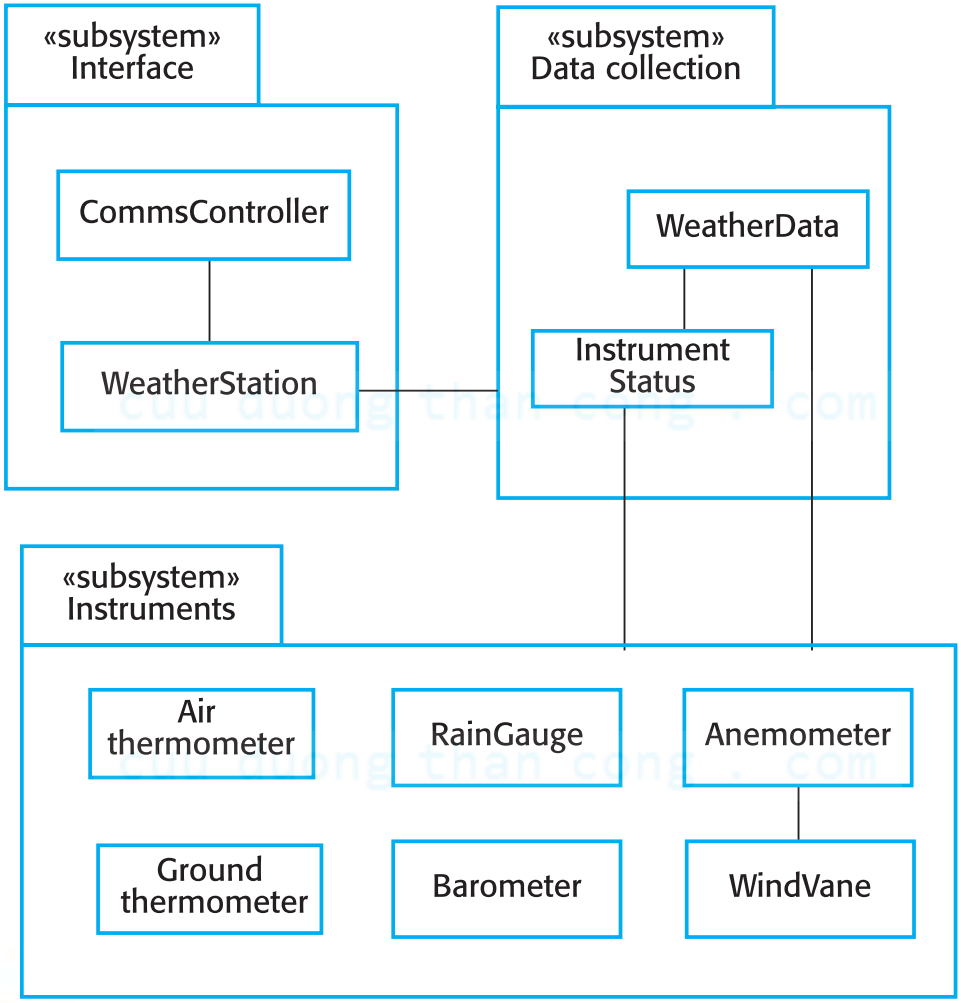
Examples of design models

- Subsystem models
  - show logical groupings of objects into coherent subsystems.
- Sequence models
  - show the sequence of object interactions.
- State machine models
  - show how individual objects change their state in response to events.
- Other models include use-case models, aggregation models, generalisation models, etc.

Subsystem models

- Are static models.
- Shows how the design is organised into logically related groups of objects.
- In the UML, these are shown using packages - an encapsulation construct.
  - This is a logical model.
  - The actual organisation of objects in the system may be different.

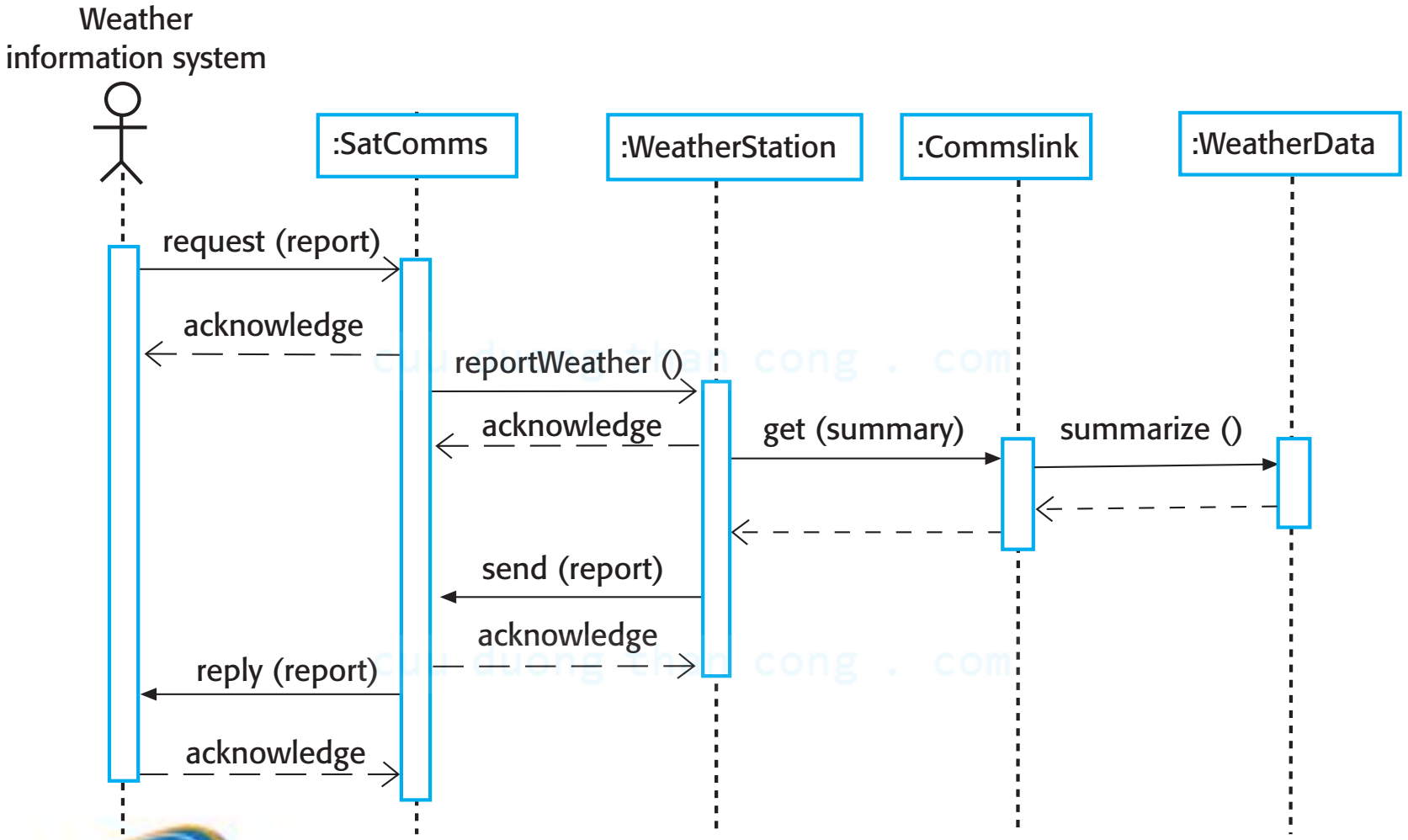
Weather station subsystems



Sequence models

- Are dynamic models.
- Sequence models show the sequence of object interactions that take place
  - Objects are arranged horizontally across the top;
  - Time is represented vertically so models are read top to bottom;
  - Interactions are represented by labelled arrows, Different styles of arrow represent different types of interaction;
  - A thin rectangle in an object lifeline represents the time when the object is the controlling object in the system.

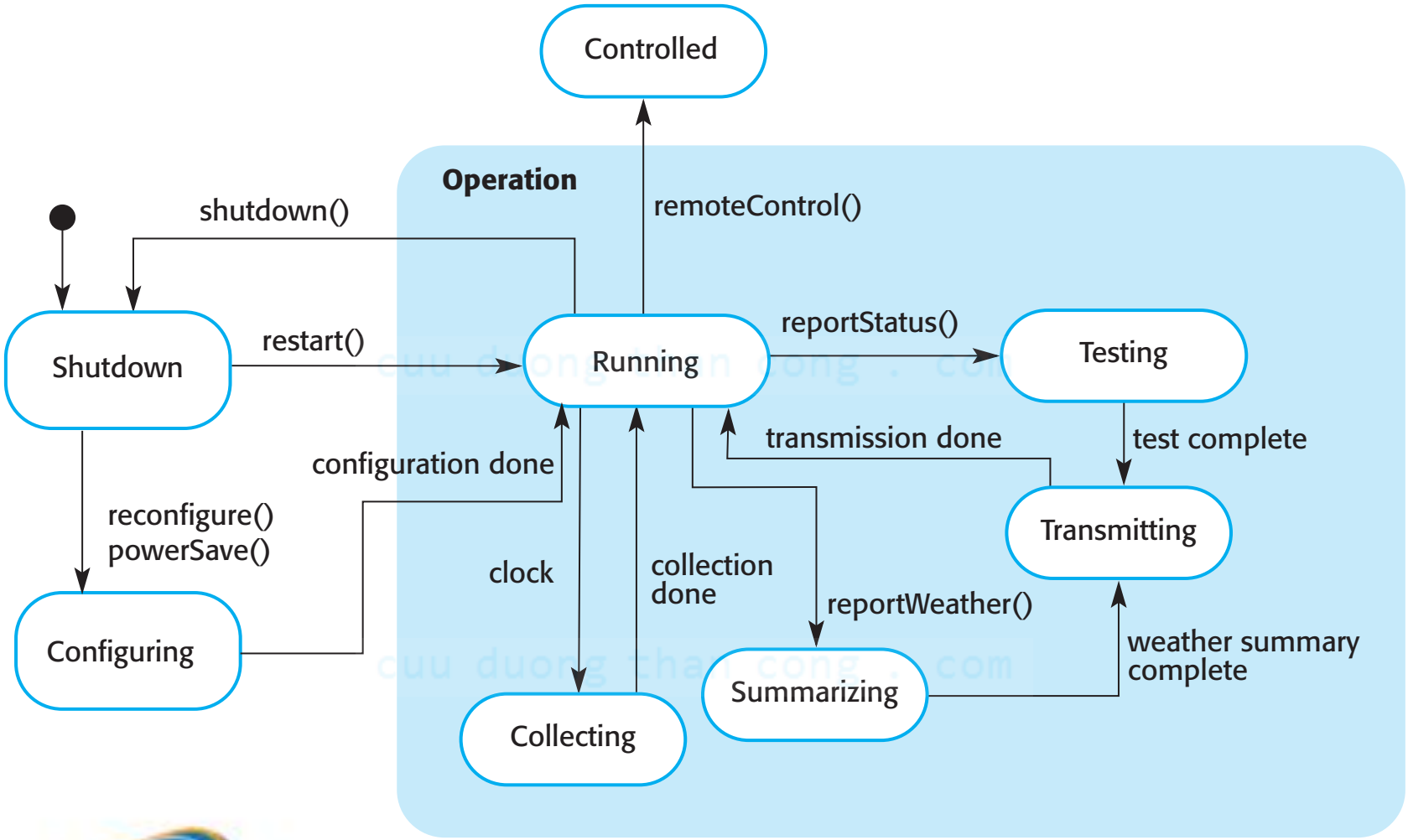
Sequence diagram describing data collection



State diagrams

- Are dynamic models.
- Are used to show
  - how objects respond to different service requests and
  - the state transitions triggered by these requests.
- Are useful high-level models of a system or an object's run-time behavior.
- You don't usually need a state diagram for all of the objects in the system.
  - Many of the objects in a system are relatively simple and a state model adds unnecessary detail to the design.

Weather station state diagram



Process stages

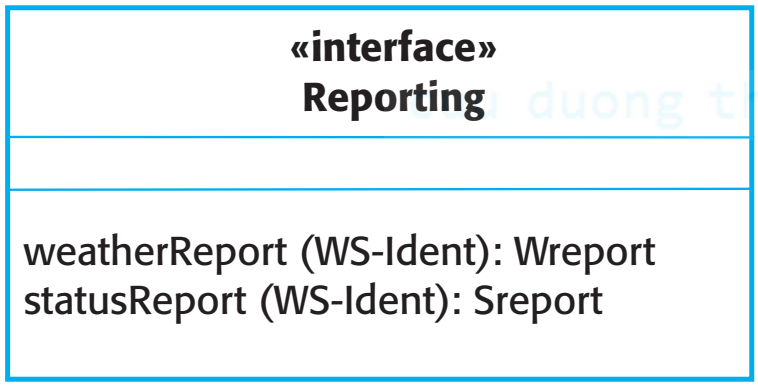
- To develop an OOD from concept to detailed, there are several things that you need to do:

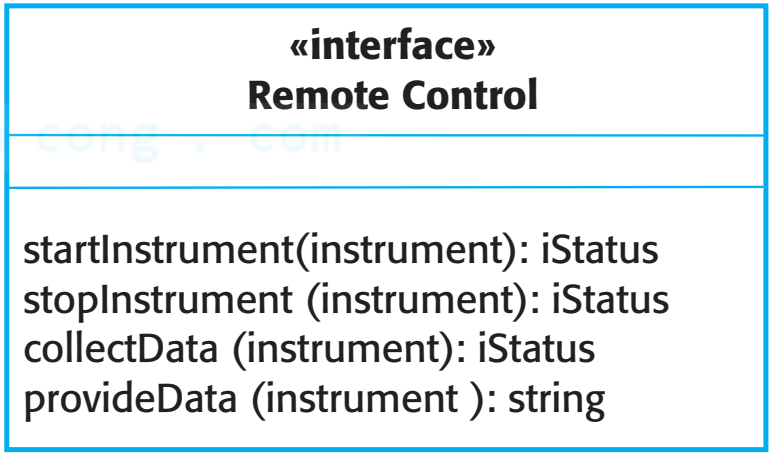
- Define the context and modes of use of the system
- Design the system architecture
- Identify the principal system objects
- Develop design models
- Specify object interfaces

Interface specification

- Object interfaces have to be specified so that the objects and other components can be designed in parallel.
- Designers should avoid designing the interface representation but should hide this in the object itself.
- Objects may have several interfaces which are viewpoints on the methods provided.
- The UML uses class diagrams for interface specification but Java may also be used.

Weather station interfaces





Weather station interface

```
interface WeatherStation {

        public void WeatherStation () ;

        public void startup () ;
        public void startup (Instrument i) ;

        public void shutdown () ;
        public void shutdown (Instrument i) ;

        public void reportWeather ( ) ;

        public void test () ;
        public void test ( Instrument i ) ;

        public void calibrate ( Instrument i) ;

        public int getID () ;

} //WeatherStation
```

Topics covered

1. Object-oriented design using the UML
2. Design patterns
3. Open source development

Design patterns

- 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
- Consequences
  - Optimisations to enhance display performance are impractical.

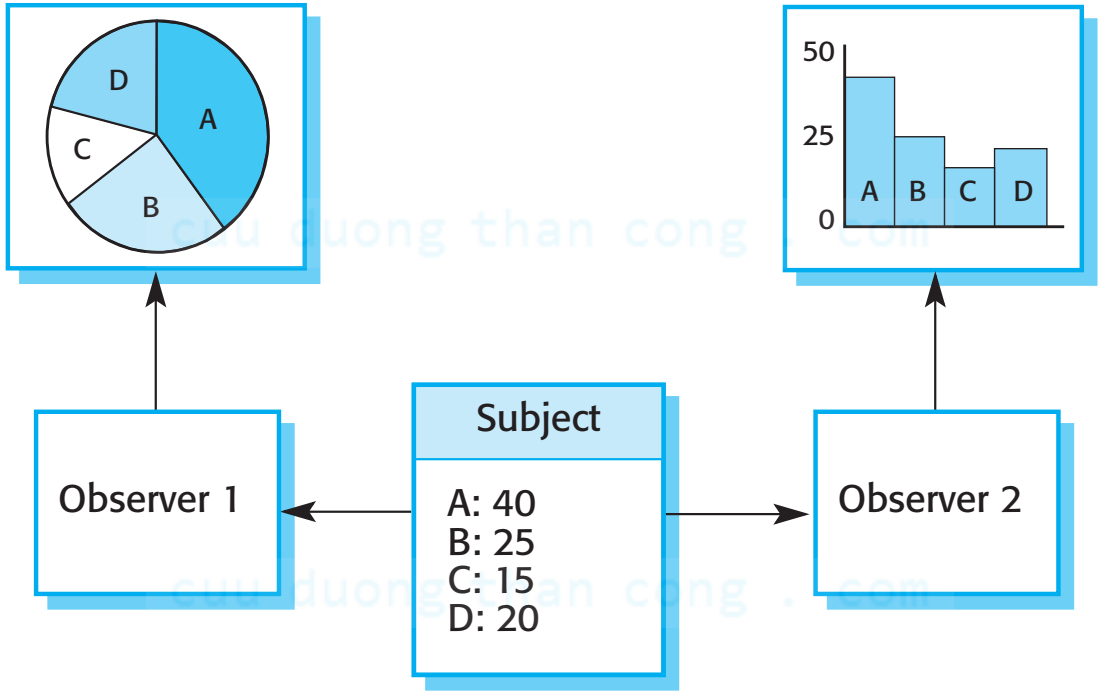
The Observer pattern (1)

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.

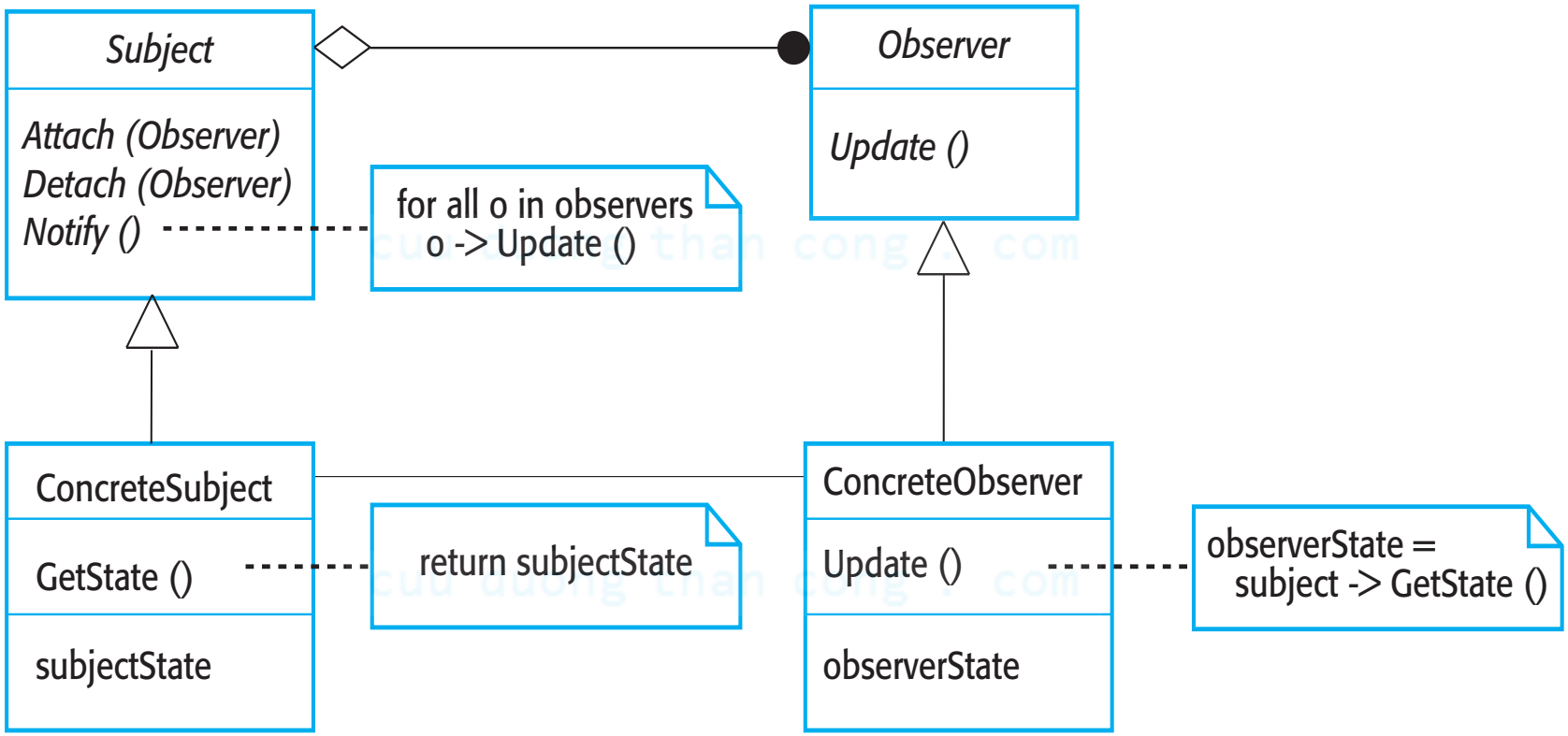
The Observer pattern (2)

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



A UML model of 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).

Topics covered

1. Object-oriented design using the UML
2. Design patterns
3. Open source development

Open source development

- Open source development is an approach to software development
  - the source code of a software system is published and volunteers are invited to participate in the development process.
- Free Software Foundation (www.fsf.org)
  - "The Free Software Foundation (FSF) is a nonprofit with a worldwide mission to promote computer user freedom. We defend the rights of all software users."
- 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 the Linux operating system
  - 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 issues

- For a company:
  - Should the product that is being developed make use of open source components?
  - Should an open source approach be used for the software's development?

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: 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 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
  - 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)
  - A variant of the GPL license
  - 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
  - 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.

Questions?