Object-Oriented Design

Object-oriented design using the UML

An object-oriented design process

- Structured object-oriented design processes involve developing a number of different system models.
- They require a lot of effort for development and maintenance of these models and, for small systems, this may not be cost-effective.
- However, for large systems developed by different groups design models are an important communication mechanism.

Process stages

- There are a variety of different object-oriented design processes that depend on the organization using the process.
- Common activities in these processes include:
  - 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**.
- Example used in the book is a wilderness weather station.

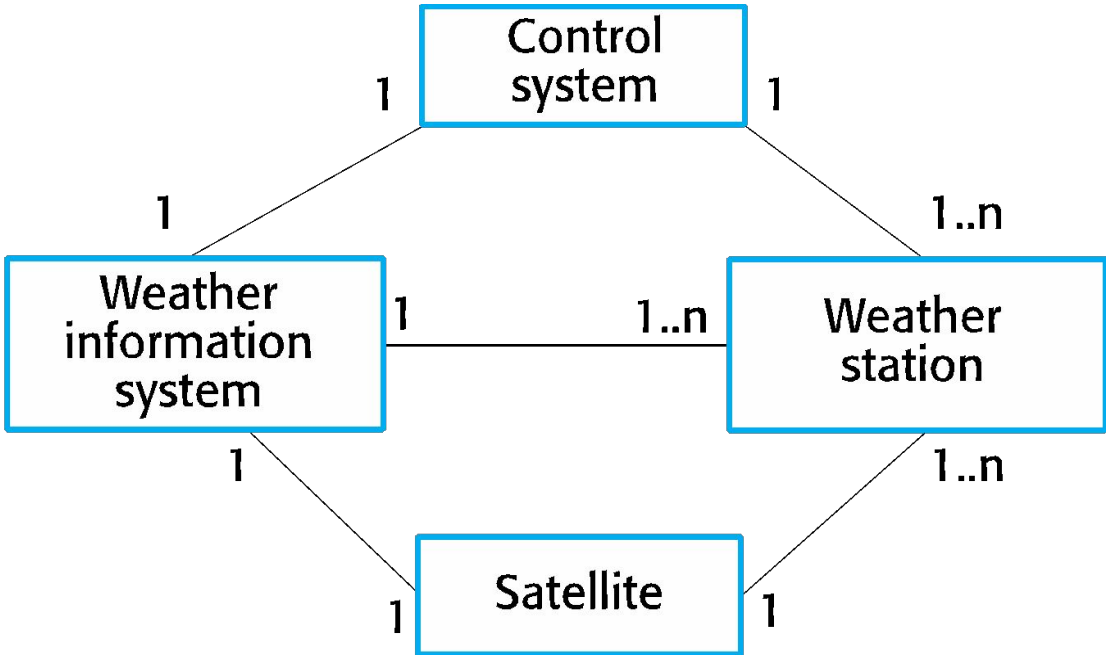
System context and interactions

- 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
  - for deciding 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 us decide what features are implemented in the system being designed and what features are in other associated systems.

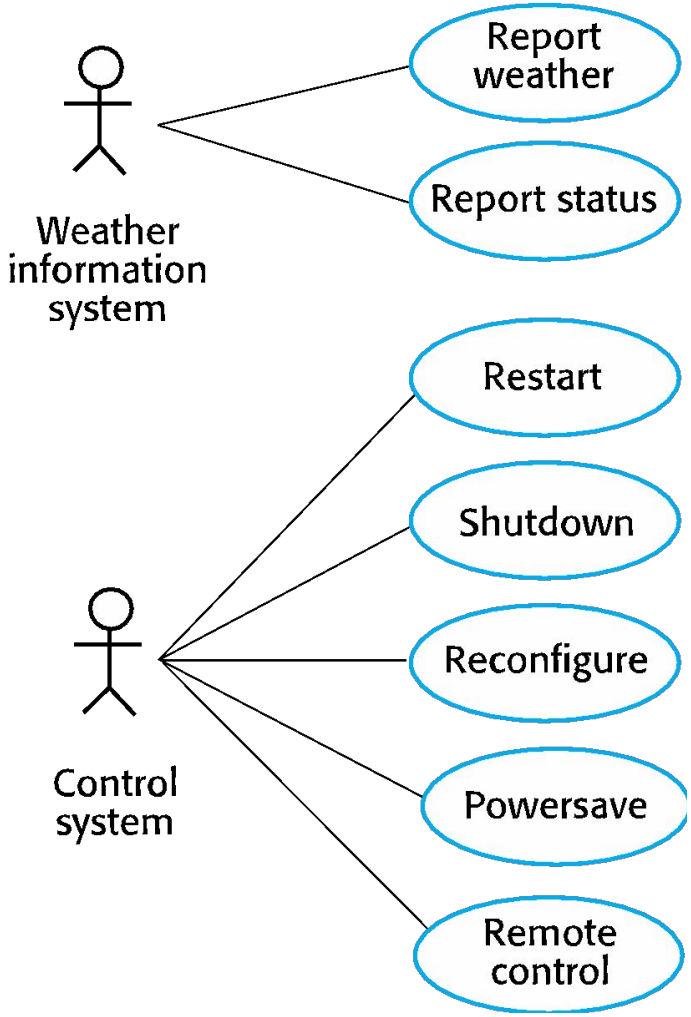
Context and interaction models

- A system context model is a structural model that demonstrates the other systems in the environment of the system being developed.
- An interaction model is a dynamic model that shows how the system interacts with its environment as it is used.

System context for the weather station



Weather station use cases



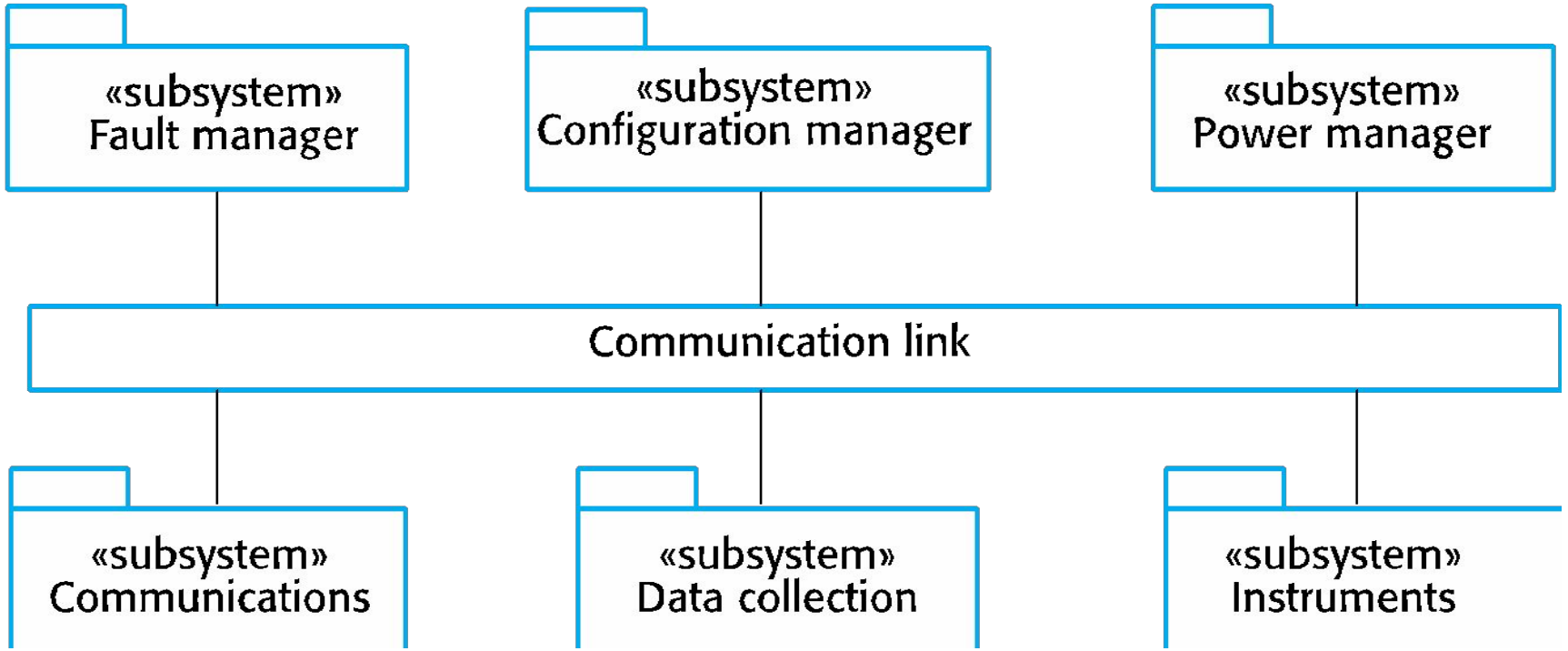
Use case description—Report weather

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.

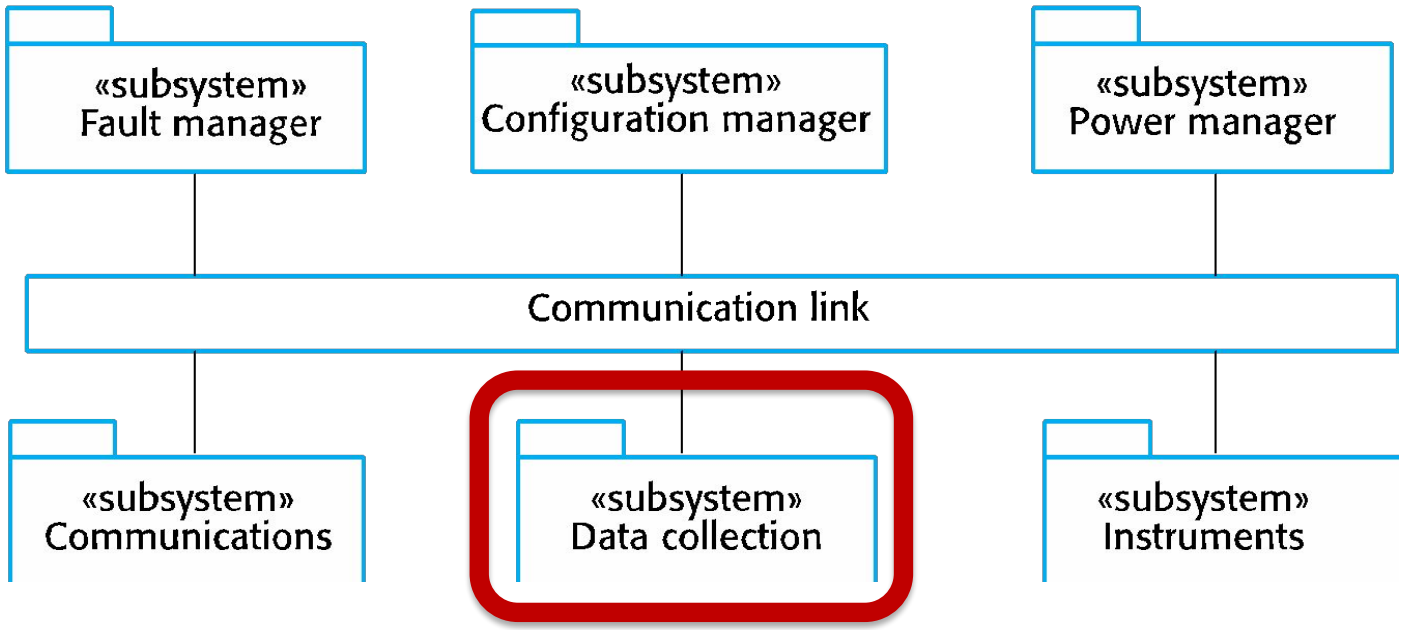
Architectural design

- Once interactions between the system and its environment have been understood, you use this information for designing the system architecture.
- You identify the major components that make up the system and their interactions, and then may organize the components using an architectural pattern and/or style
  - E.g., layered, client-server model, dataflow, event-based.
- The weather station is composed of independent subsystems that communicate by broadcasting messages on a common infrastructure.

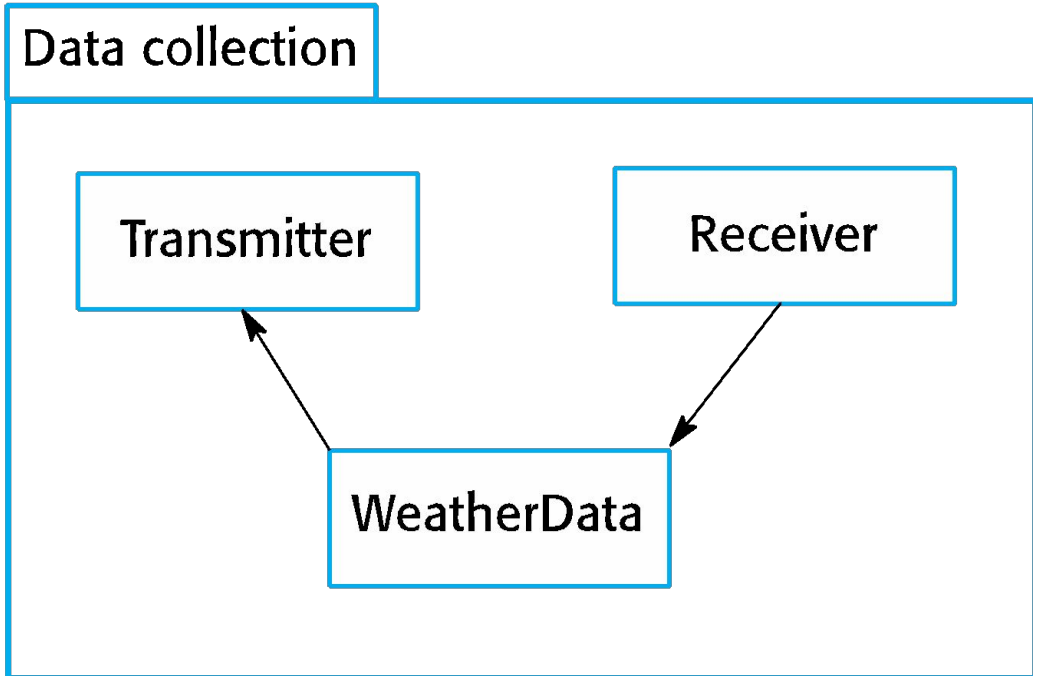
High-level architecture of the weather station



High-level architecture of the weather station



Architecture of data collection system



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 designers’
  - skill
  - experience
  - domain knowledge.
- Object identification is an iterative process.
- You are unlikely to get it right first time.

Approaches to object 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 behavioral approach and identify objects based on what participates in what behavior.
- Use a scenario-based analysis.
  - Identify objects, attributes and methods in each scenario.

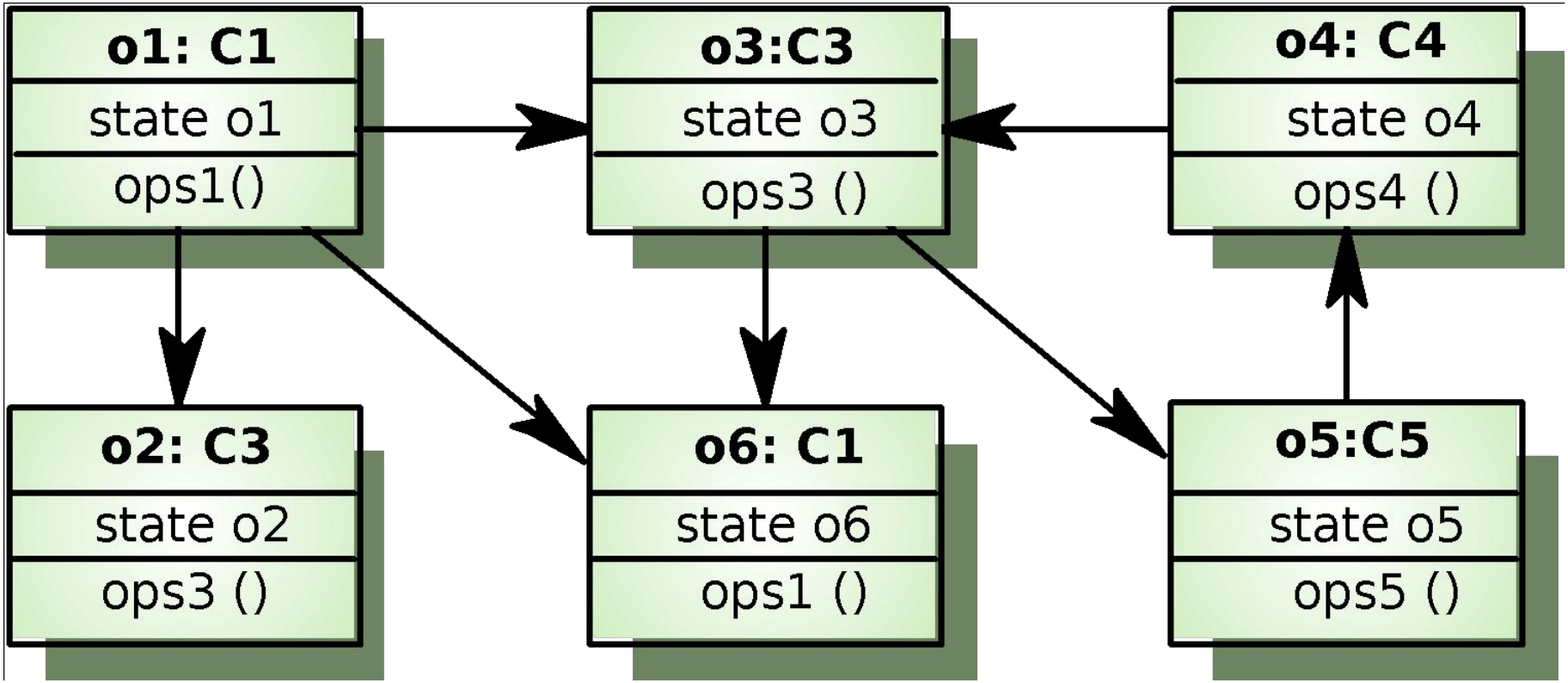
Characteristics of OOD

- Objects are abstractions of real-world or system entities and manage themselves
- Objects are independent and encapsulate state and representation information.
- System functionality is expressed in terms of object services
- Shared data areas are eliminated
  - Objects communicate by message passing
- Objects may be distributed
- Objects may execute sequentially or in parallel

Advantages of OOD

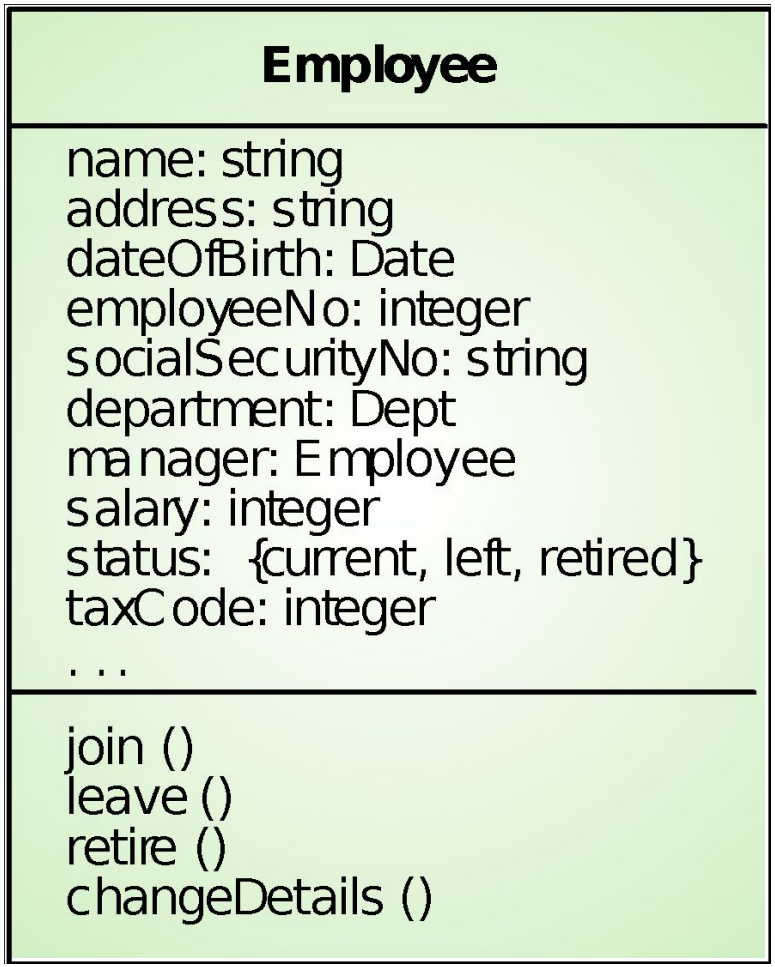
- Easier maintenance. Objects may be understood as stand-alone entities
- Objects are appropriate reusable components
  - *Is this entirely true?*
- For some systems, there may be an obvious mapping from real world entities to system objects

Interacting objects



Objects and object classes

- Objects are entities in a software system which represent instances of real-world and system entities
- Object classes are templates for objects
  - Classes may be used to create objects
- Object classes may inherit attributes and services from other object classes



Object communication

- Conceptually, objects communicate by message passing
- Messages
  - The name of the service requested by the calling object.
  - Copies of the information required to execute the service and the name of a holder for the result of the service.
- In practice, messages are often implemented by procedure (a.k.a. method) calls
  - Name = method name
  - Information = parameter list
  - Result holder = method return value

Message examples

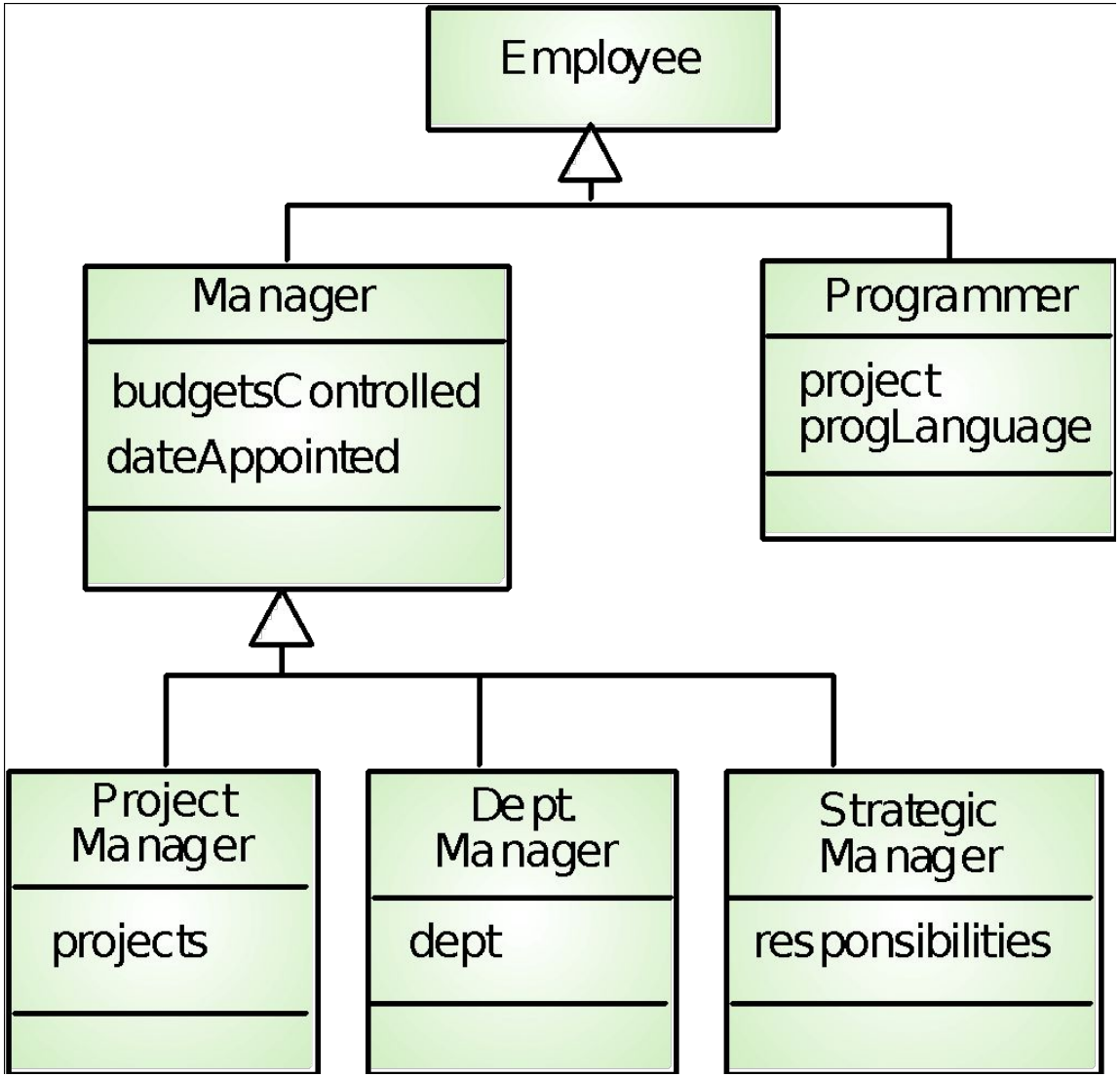
```
// Call a method associated with a buffer
// object that returns the next value
// in the buffer
    v = circularBuffer.Get () ;

// Call the method associated with a
// thermostat object that sets the
// temperature to be maintained
    thermostat.setTemp (20) ;
```

Generalization and inheritance

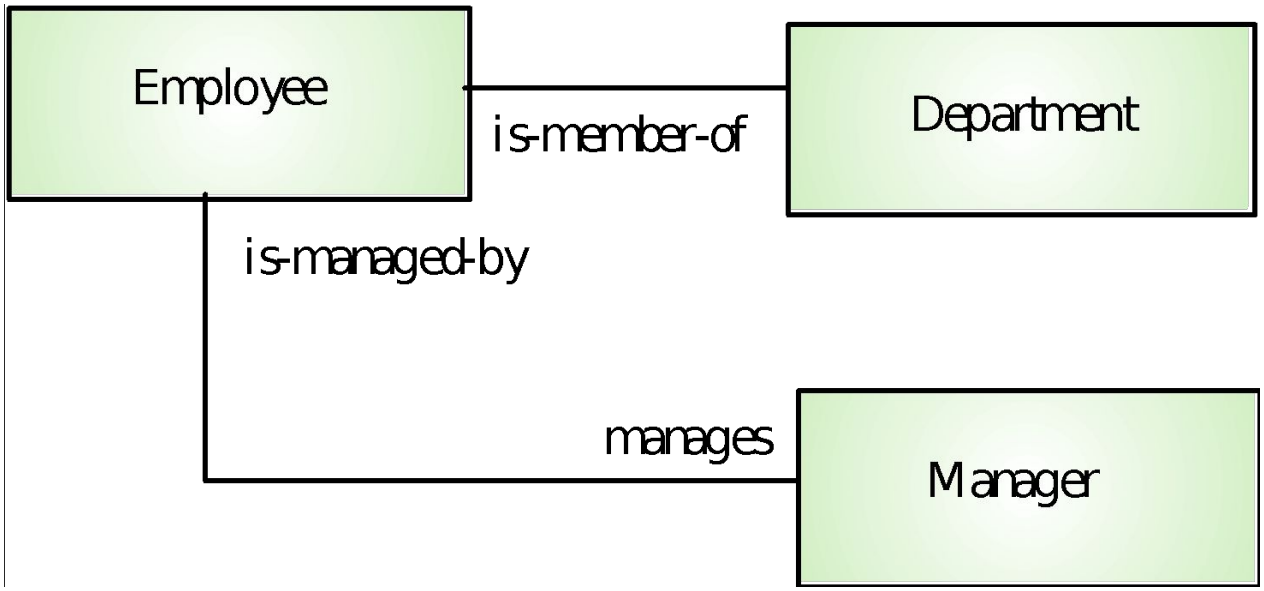
- Objects are members of classes which define attribute types and operations
- Classes may be arranged in a class hierarchy where one class (a super-class) is a generalization of one or more other classes (sub-classes)
- A sub-class inherits the attributes and operations from its super class and may add new methods or attributes of its own
- It is a reuse mechanism at both the design and the programming level
- Inheritance introduces complexity and this is undesirable, especially in critical systems

A generalisation hierarchy



Object Relationships

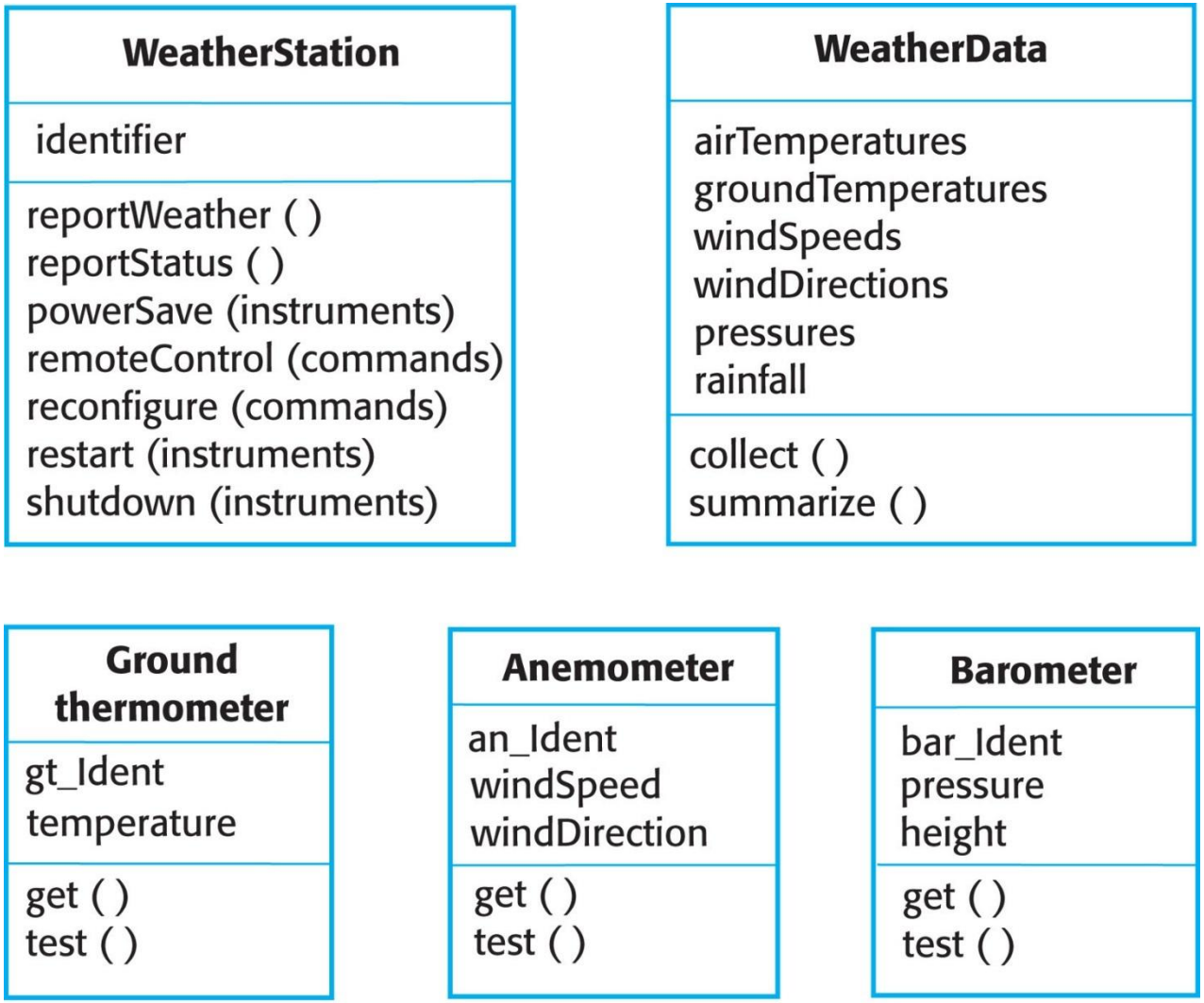
- Objects and object classes participate in relationships with other objects and object classes
  - In UML, such a relationship is indicated by an association
- Associations may be annotated with information that describes the association



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
    - 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



Design models

- Design models show the objects and object classes and relationships between these entities.
- There are two kinds of design model:
  - **Structural 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 that show logical groupings of objects into coherent subsystems.
- **Sequence** models that show the sequence of object interactions.
- **State machine** models that show how individual objects change their state in response to events.
- Other models include use-case models, aggregation models, generalization models, etc.

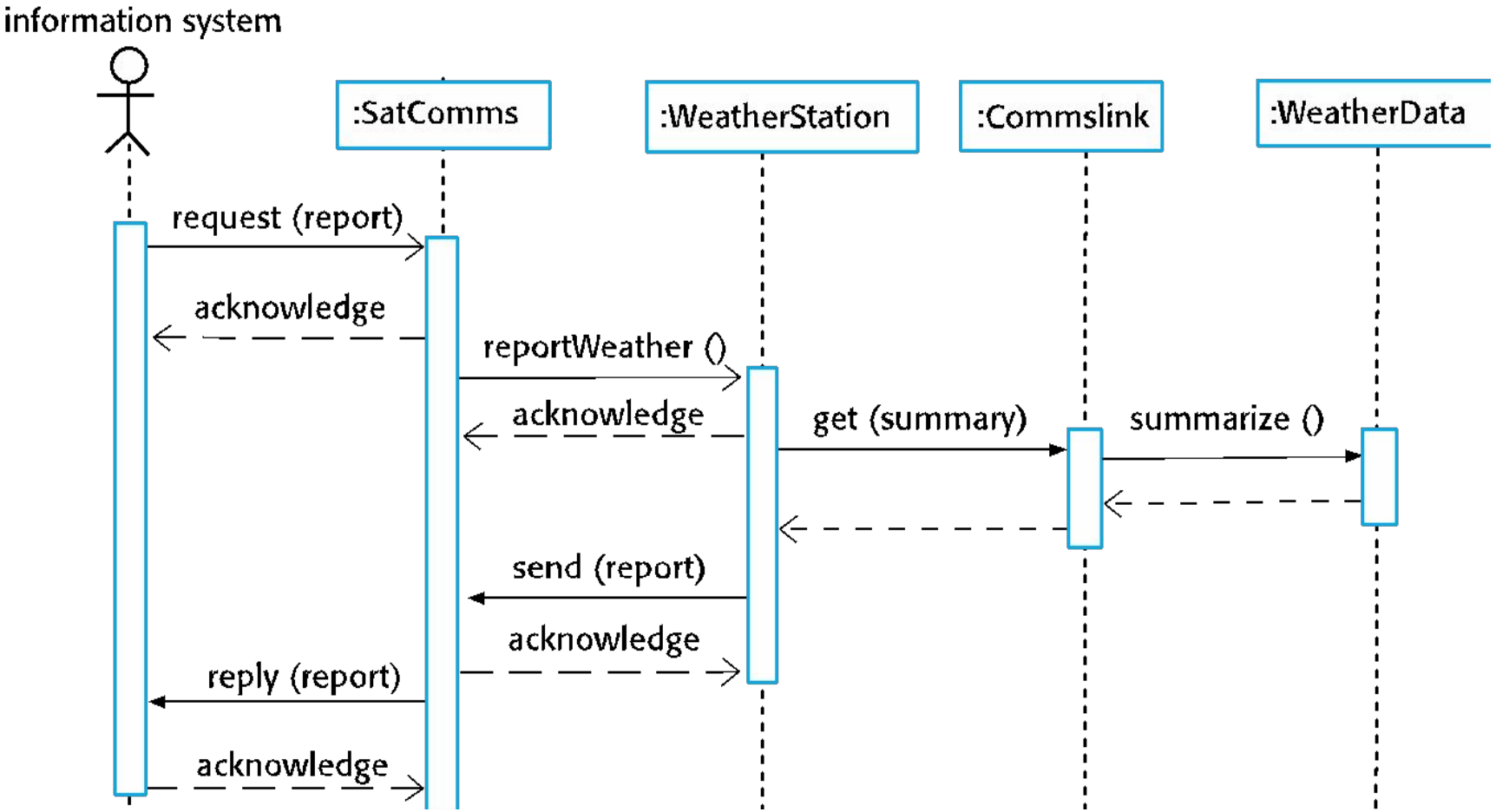
Subsystem models

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

Sequence models

- Sequence models show orderings 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.

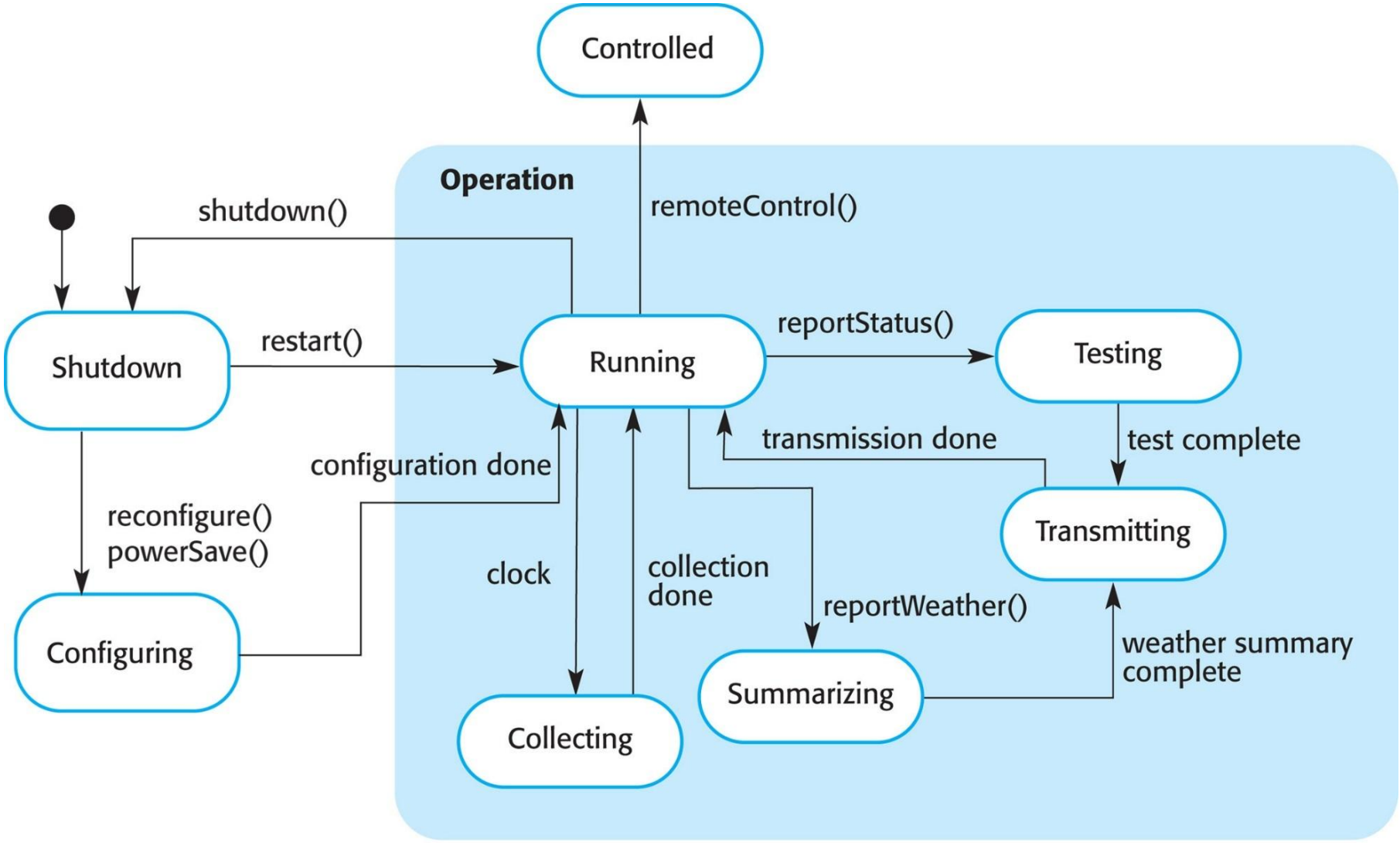
Sequence diagram describing data collection



State diagrams

- State diagrams are used to show
  - how objects respond to different service requests
  - the state transitions triggered by these requests.
- State diagrams are high-level models of a system's or object's run-time behavior.
- You won't usually need a state diagram for all of the objects in the system.
  - Many of the objects in a system are relatively simple – a state model may add unnecessary detail to the design.

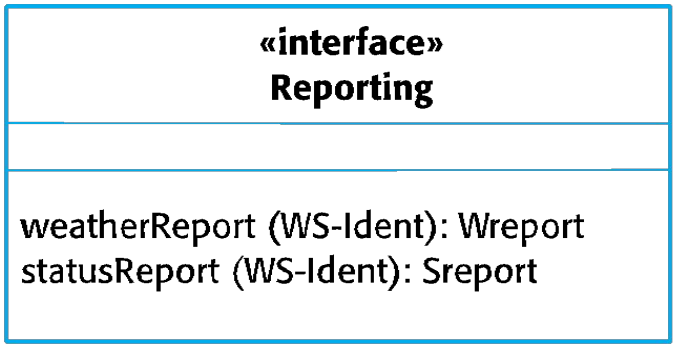
Weather station state diagram

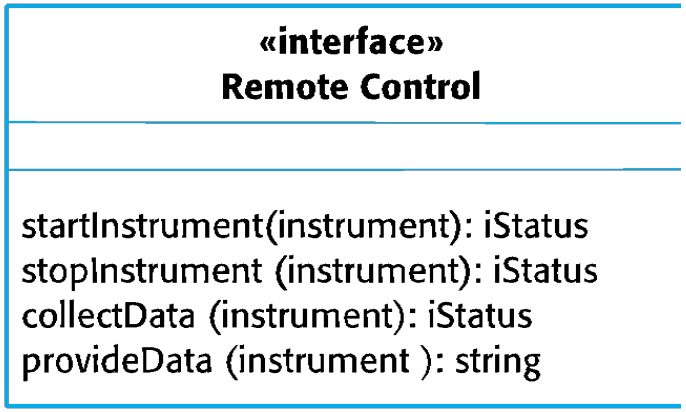


Interface specification

- Object interfaces have to be specified so that the objects and other components can be designed in parallel.
- 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





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.

Patterns

- *Patterns and Pattern Languages are ways to describe best practices, good designs, and capture experience in a way that it is possible for others to reuse this experience.*

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
  - Optimizations to enhance display performance are impractical.

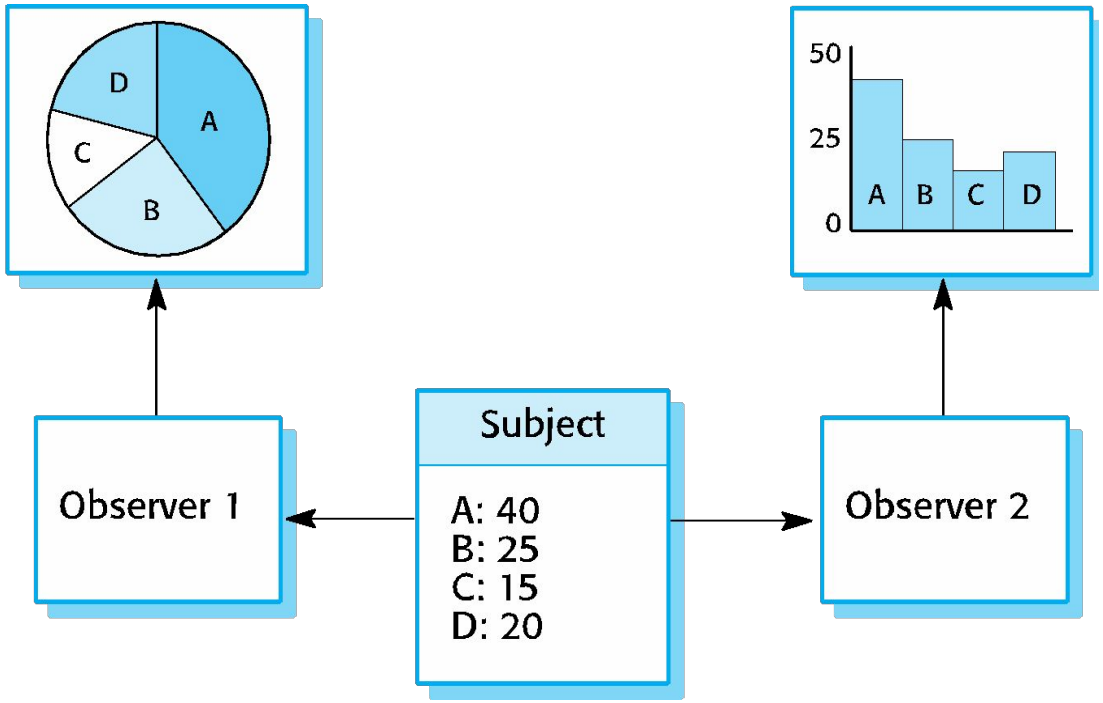
The Observer pattern (1)

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.

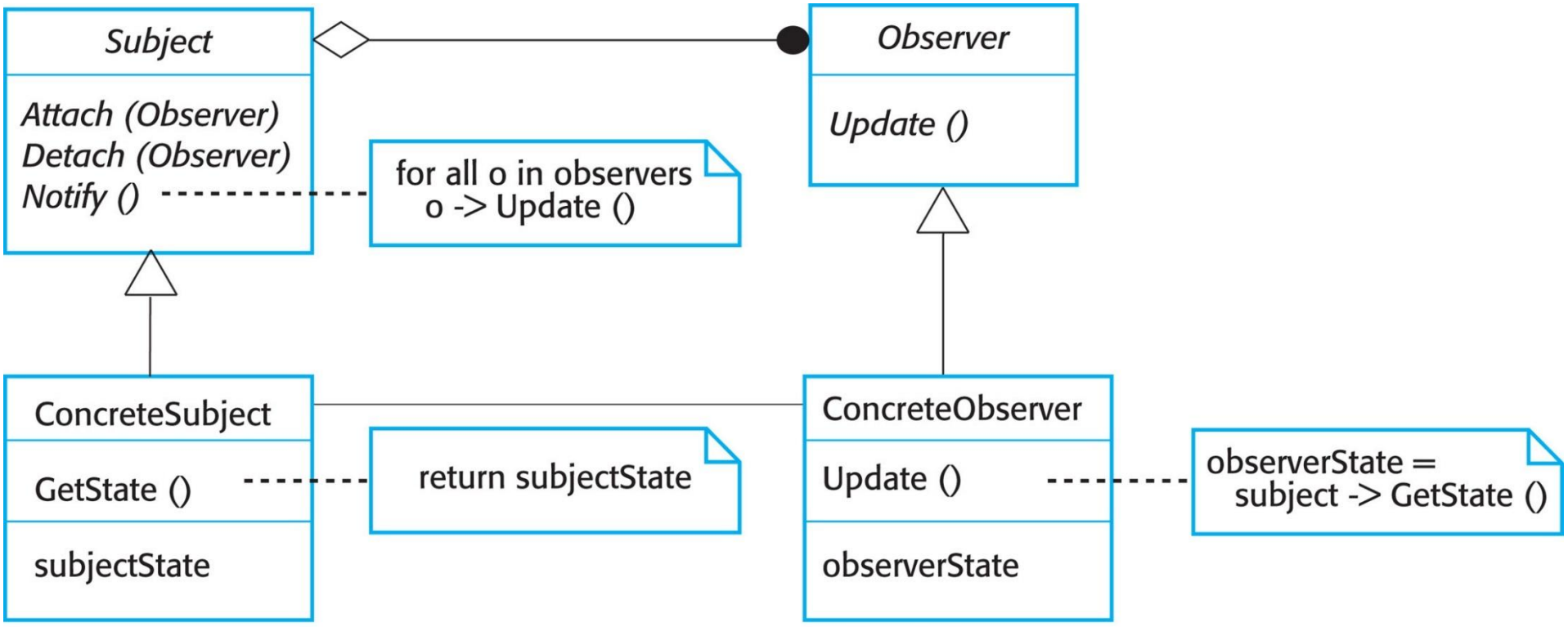
The Observer pattern (2)

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).