Chapter 12

Object-Oriented Design

Object-oriented Design

- Designing systems using self-contained objects and object classes

Objectives

- To explain how a software design may be represented as a set of interacting objects that manage their own state and operations
- To describe the activities in the object-oriented design process
- To introduce various models that describe an object-oriented design
- To show how the UML may be used to represent these models

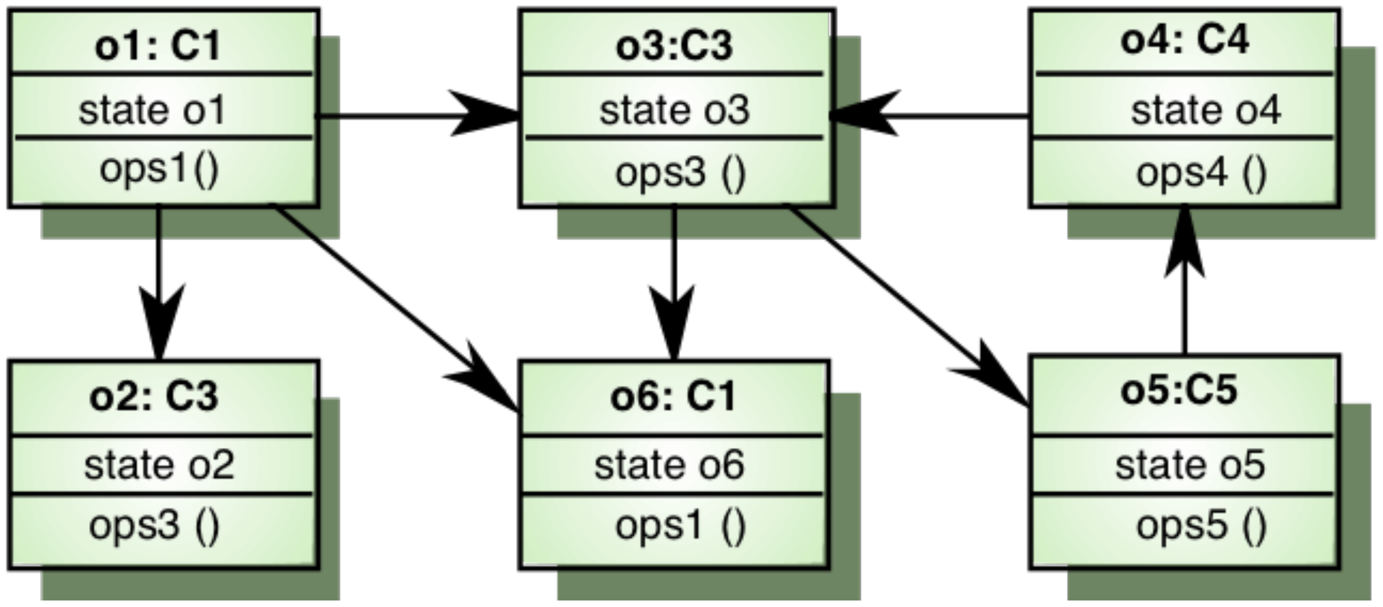
Topics covered

- Objects and object classes
- An object-oriented design process
- Design evolution

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 and may execute sequentially or in parallel

Interacting objects



Advantages of OOD

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

Object-oriented development

- Object-oriented analysis, design and programming 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

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

Objects

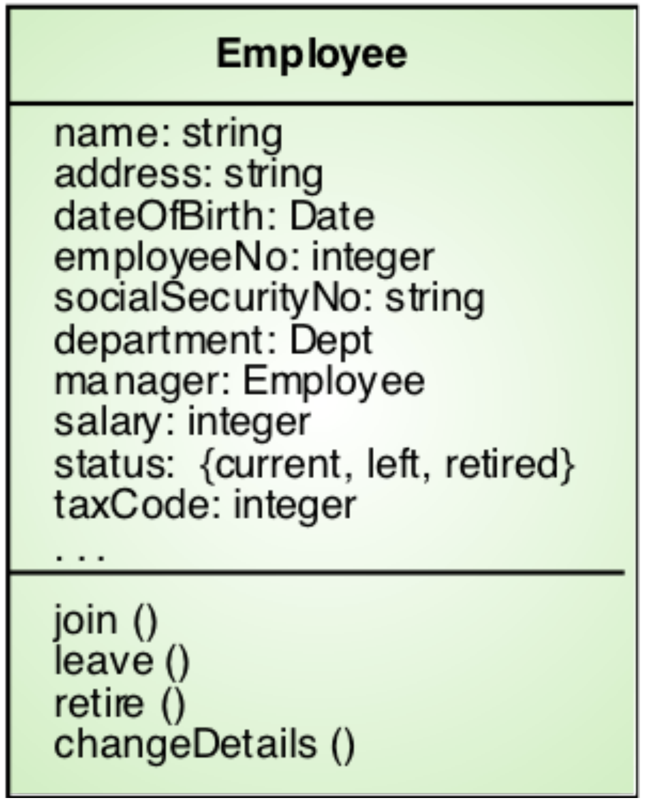
An **object** is an entity which has a state and a defined set of operations which operate on that state. The state is represented as a set of object attributes. The operations associated with the object provide services to other objects (clients) which request these services when some computation is 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.

The Unified Modeling Language

- Several different notations for describing object-oriented designs were proposed in the 1980s and 1990s
- The Unified Modeling Language is an integration of these notations
- It describes notations for a number of different models that may be produced during OO analysis and design
- It is now a *de facto* standard for OO modelling

Employee object class (UML)



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 calls
  - Name = procedure name.
  - Information = parameter list.

Message examples

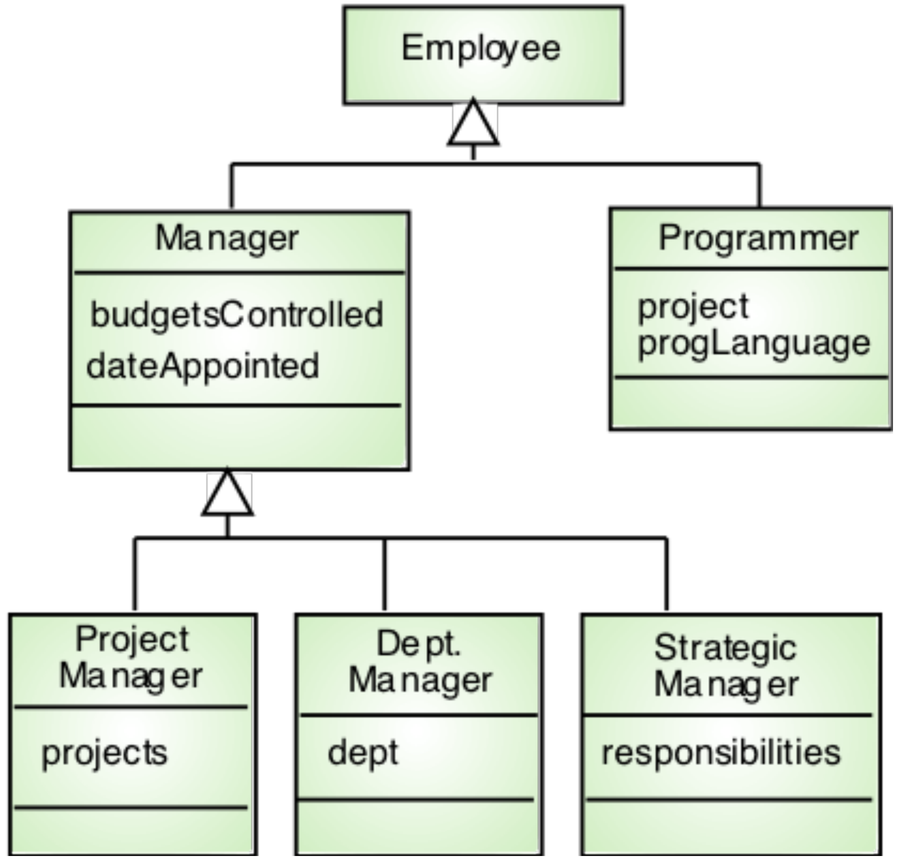
```
// Call a method associated with a buffer
  // object that returns the next value
  //   in the buffer
          v = circularBuffer.Get () ;

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

Generalisation 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 generalisation 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
- Generalisation in the UML is implemented as inheritance in OO programming languages

A generalisation hierarchy



Advantages of inheritance

- It is an abstraction mechanism which may be used to classify entities
- It is a reuse mechanism at both the design and the programming level
- The inheritance graph is a source of organisational knowledge about domains and systems

Problems with inheritance

- Object classes are not self-contained. they cannot be understood without reference to their super-classes
- Designers have a tendency to reuse the inheritance graph created during analysis. Can lead to significant inefficiency
- The inheritance graphs of analysis, design and implementation have different functions and should be separately maintained

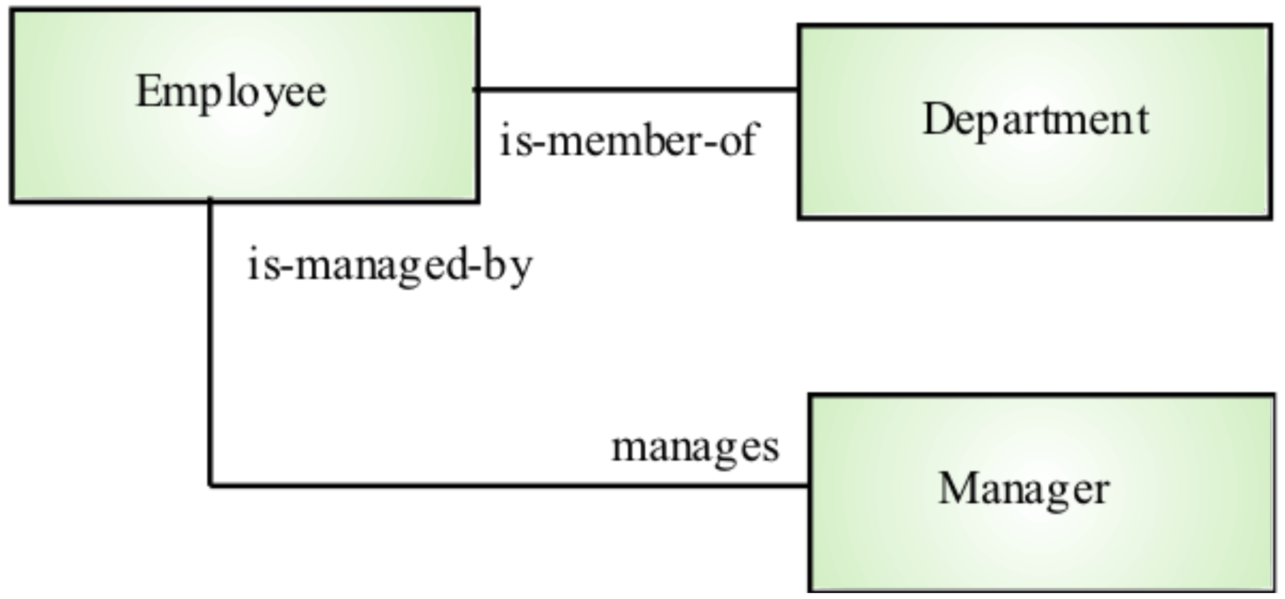
Inheritance and OOD

- There are differing views as to whether inheritance is fundamental to OOD.
  - View 1. Identifying the inheritance hierarchy or network is a fundamental part of object-oriented design. Obviously this can only be implemented using an OOPL.
  - View 2. Inheritance is a useful implementation concept which allows reuse of attribute and operation definitions. Identifying an inheritance hierarchy at the design stage places unnecessary restrictions on the implementation
- Inheritance introduces complexity and this is undesirable, especially in critical systems

UML associations

- Objects and object classes participate in relationships with other objects and object classes
- In the UML, a generalised relationship is indicated by an association
- Associations may be annotated with information that describes the association
- Associations are general but may indicate that an attribute of an object is an associated object or that a method relies on an associated object

An association model



Concurrent objects

- The nature of objects as self-contained entities make them suitable for concurrent implementation
- The message-passing model of object communication can be implemented directly if objects are running on separate processors in a distributed system

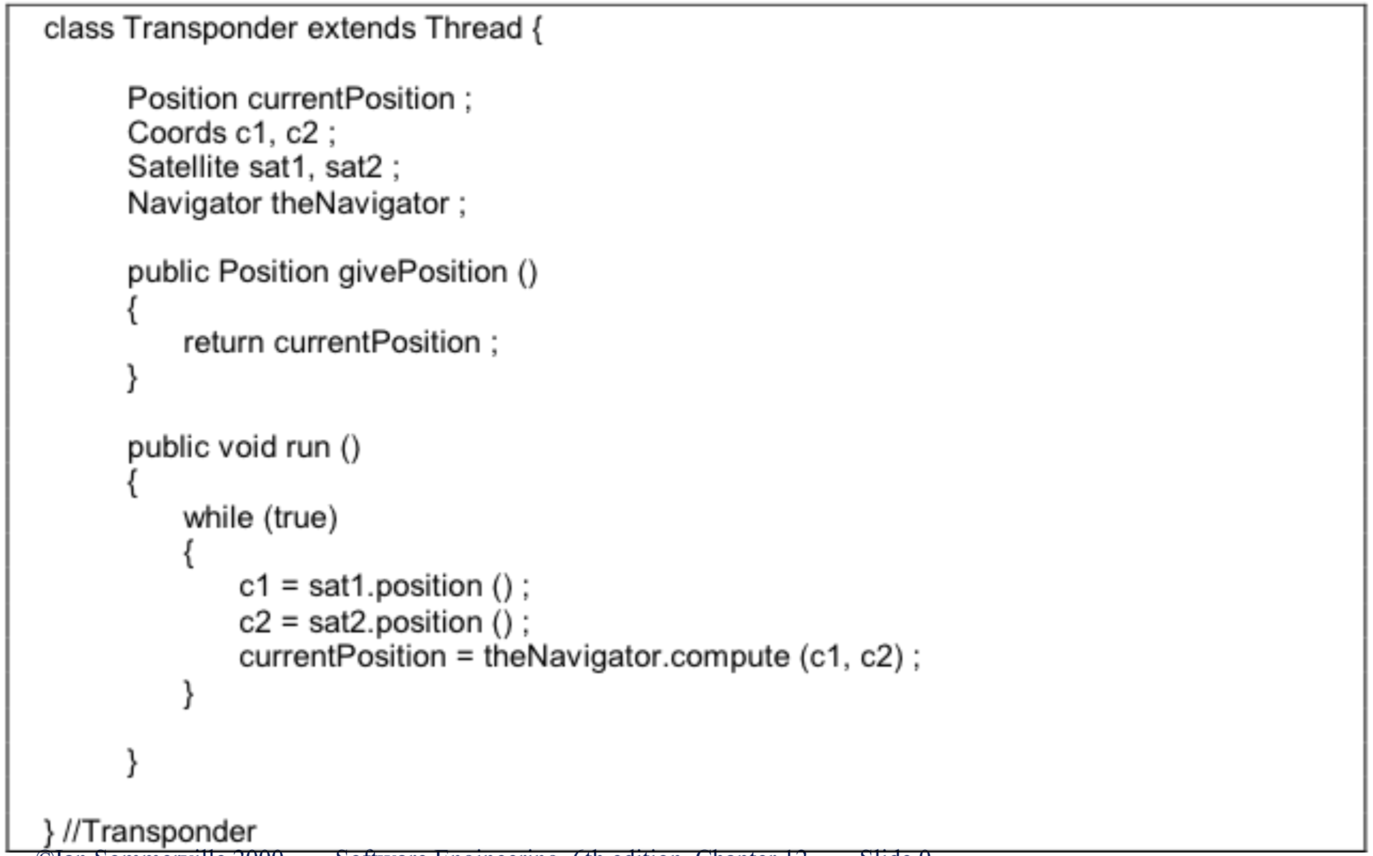
Servers and active objects

- Servers.
  - The object is implemented as a parallel process (server) with entry points corresponding to object operations. If no calls are made to it, the object suspends itself and waits for further requests for service
- Active objects
  - Objects are implemented as parallel processes and the internal object state may be changed by the object itself and not simply by external calls

Active transponder object

- Active objects may have their attributes modified by operations but may also update them autonomously using internal operations
- Transponder object broadcasts an aircraft’s position. The position may be updated using a satellite positioning system. The object periodically update the position by triangulation from satellites

An active transponder object



```
class Transponder extends Thread {

        Position currentPosition ;
        Coords c1, c2 ;
        Satellite sat1, sat2 ;
        Navigator theNavigator ;

        public Position givePosition ()
        {
            return currentPosition ;
        }

        public void run ()
        {
            while (true)
            {
                c1 = sat1.position () ;
                c2 = sat2.position () ;
                currentPosition = theNavigator.compute (c1, c2) ;
            }

        }

} //Transponder
```

Java threads

- Threads in Java are a simple construct for implementing concurrent objects
- Threads must include a method called run() and this is started up by the Java run-time system
- Active objects typically include an infinite loop so that they are always carrying out the computation

An object-oriented design process

- 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

Weather system description

A weather data collection system is required to generate weather maps on a regular basis using data collected from remote, unattended weather stations and other data sources such as weather observers, balloons and satellites. Weather stations transmit their data to the area computer in response to a request from that machine.

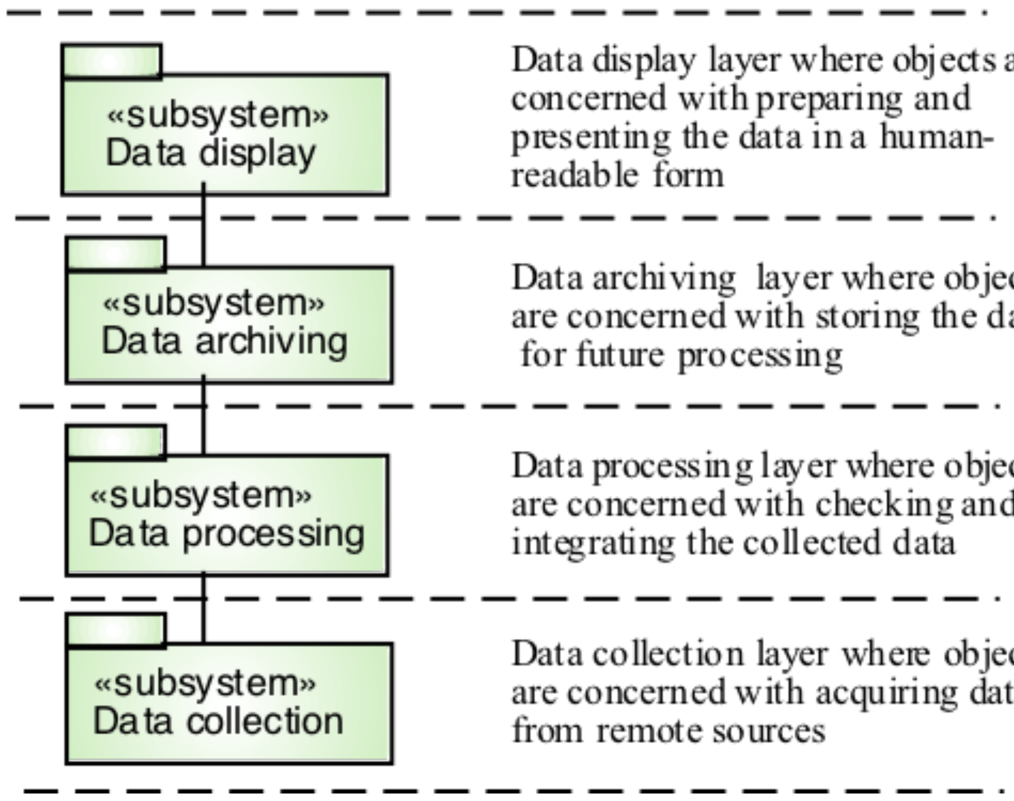
The area computer validates the collected data and integrates it with the data from different sources. The integrated data is archived and, using data from this archive and a digitised map database a set of local weather maps is created. Maps may be printed for distribution on a special-purpose map printer or may be displayed in a number of different formats.

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 every five minutes.

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.

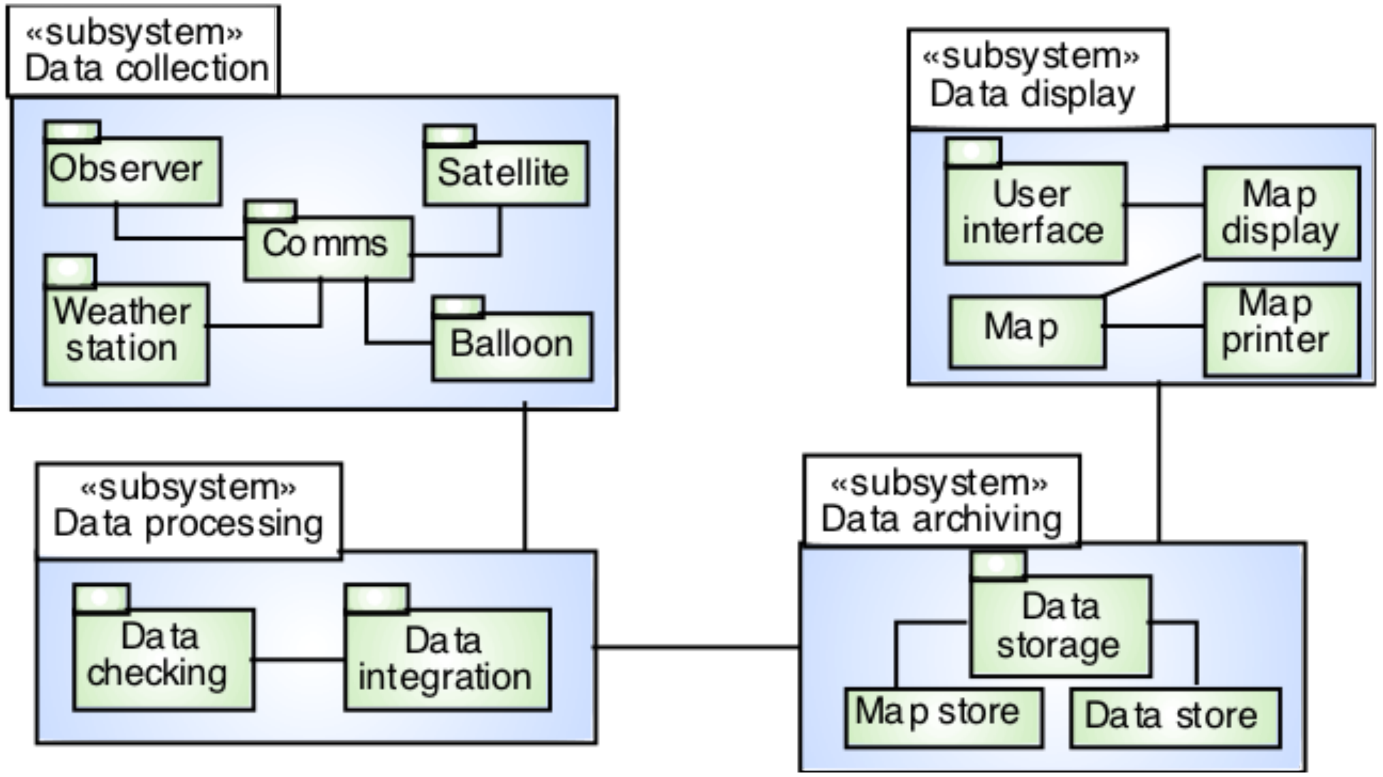
Layered architecture


«subsystem» Data display
Data display layer where objects are concerned with preparing and presenting the data in a human-readable form
«subsystem» Data archiving
Data archiving layer where objects are concerned with storing the data for future processing
«subsystem» Data processing
Data processing layer where objects are concerned with checking and integrating the collected data
«subsystem» Data collection
Data collection layer where objects are concerned with acquiring data from remote sources

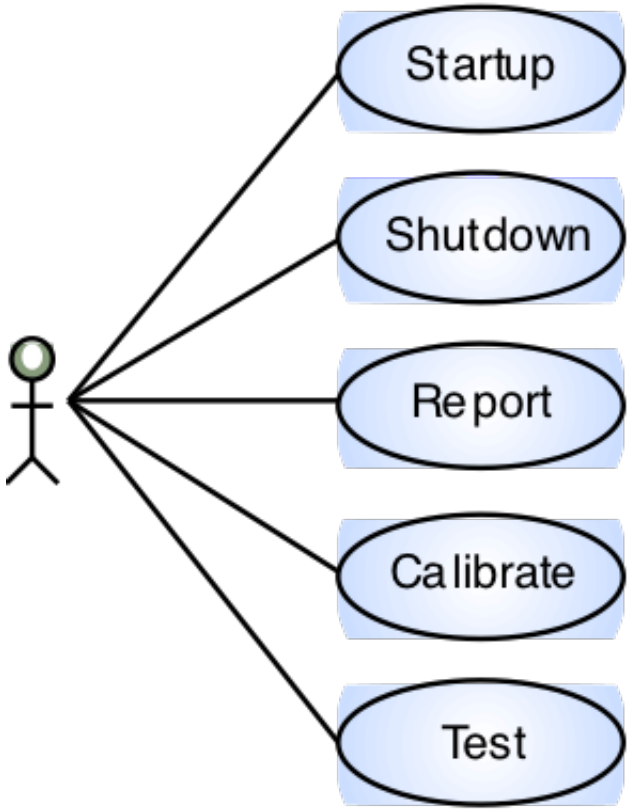
System context and models of use

- Develop an understanding of the relationships between the software being designed and its external environment
- System context
  - A static model that describes other systems in the environment. Use a subsystem model to show other systems. Following slide shows the systems around the weather station system.
- Model of system use
  - A dynamic model that describes how the system interacts with its environment. Use use-cases to show interactions

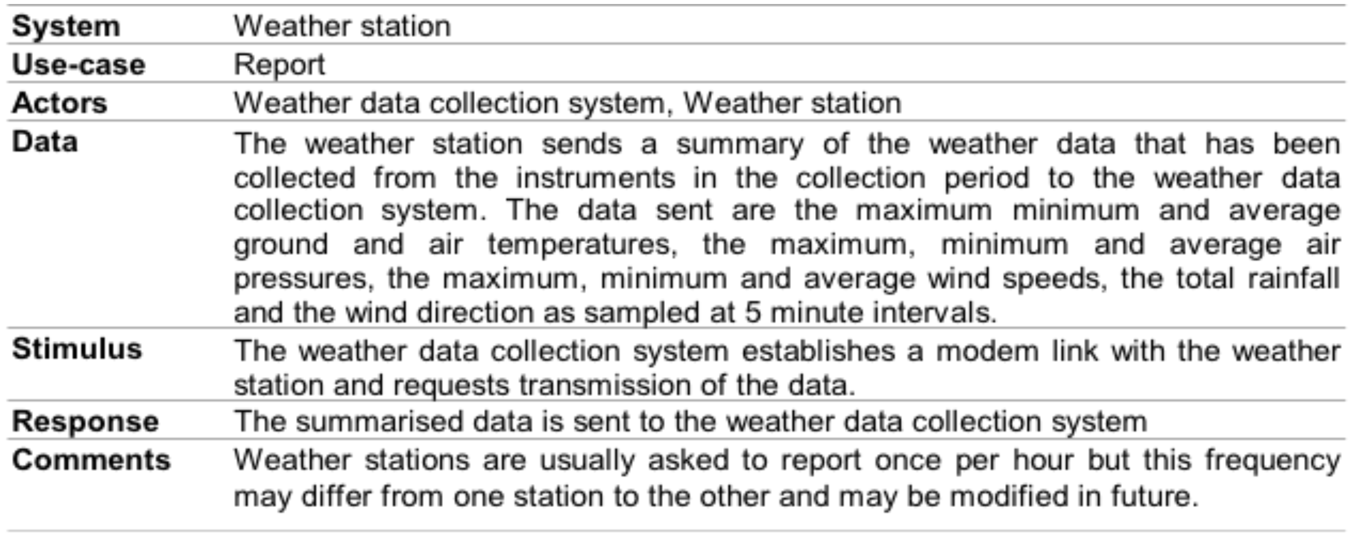
Subsystems in the weather mapping system



Use-cases for the weather station



Use-case description

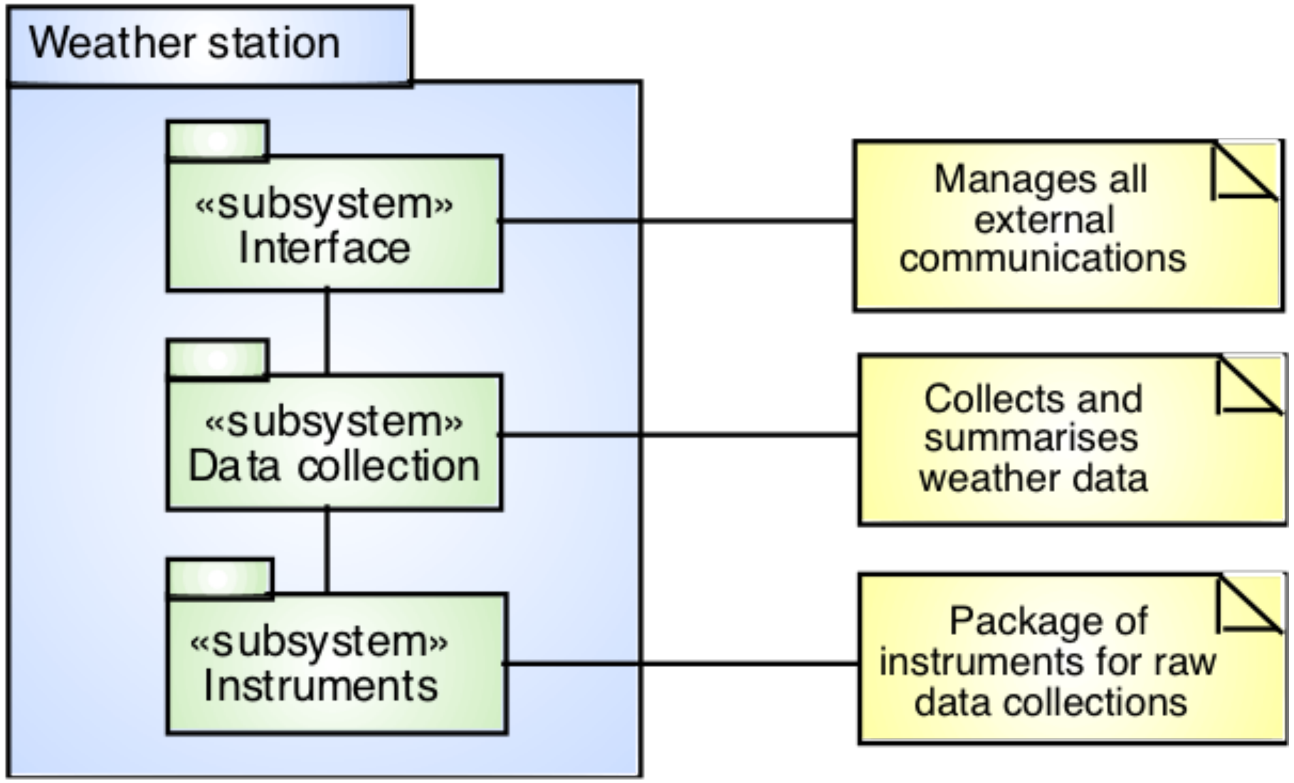


System	Weather station
Use-case	Report
Actors	Weather data collection system, Weather station
Data	The weather station sends a summary of the weather data that has been collected from the instruments in the collection period to the weather data collection system. The data sent are the maximum minimum 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 5 minute intervals.
Stimulus	The weather data collection system establishes a modem link with the weather station and requests transmission of the data.
Response	The summarised data is sent to the weather data collection system
Comments	Weather stations are usually asked to report once per hour but this frequency may differ from one station to the other and may be modified in future.

Architectural design

- Once interactions between the system and its environment have been understood, you use this information for designing the system architecture
- Layered architecture is appropriate for the weather station
  - Interface layer for handling communications
  - Data collection layer for managing instruments
  - Instruments layer for collecting data
- There should be no more than 7 entities in an architectural model

Weather station architecture


Weather station
«subsystem» Interface
«subsystem» Data collection
«subsystem» Instruments
Manages all external communications
Collects and summarises weather data
Package of instruments for raw data collections

Object identification

- Identifying objects (or object classes) is the most 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

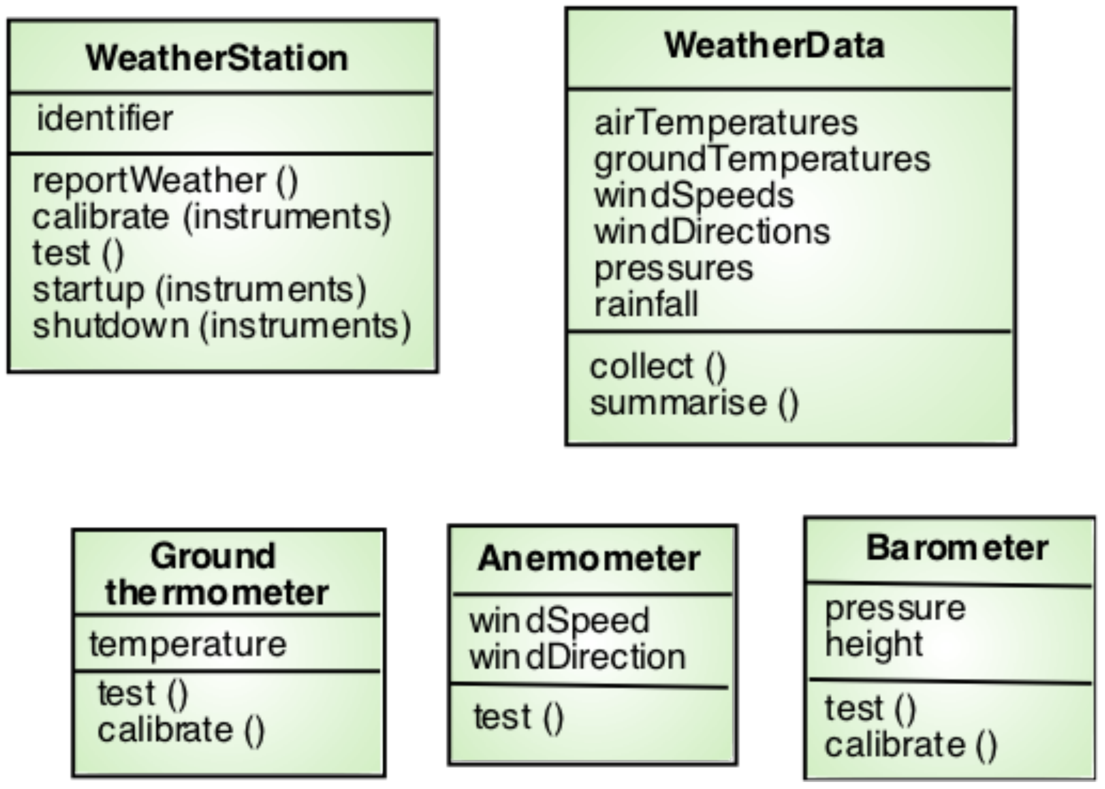
Approaches to identification

- Use a grammatical approach based on a natural language description of the system (used in Hood method)
- 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 object classes

- 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 summarised data from the instruments

Weather station object classes



Further objects and object refinement

- Use domain knowledge to identify more objects and operations
  - Weather stations should have a unique identifier
  - Weather stations are remotely situated so instrument failures have to be reported automatically. Therefore attributes and operations for self-checking are required
- Active or passive objects
  - In this case, objects are passive and collect data on request rather than autonomously. This introduces flexibility at the expense of controller processing time

Design models

- Design models show the objects and object classes and 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.

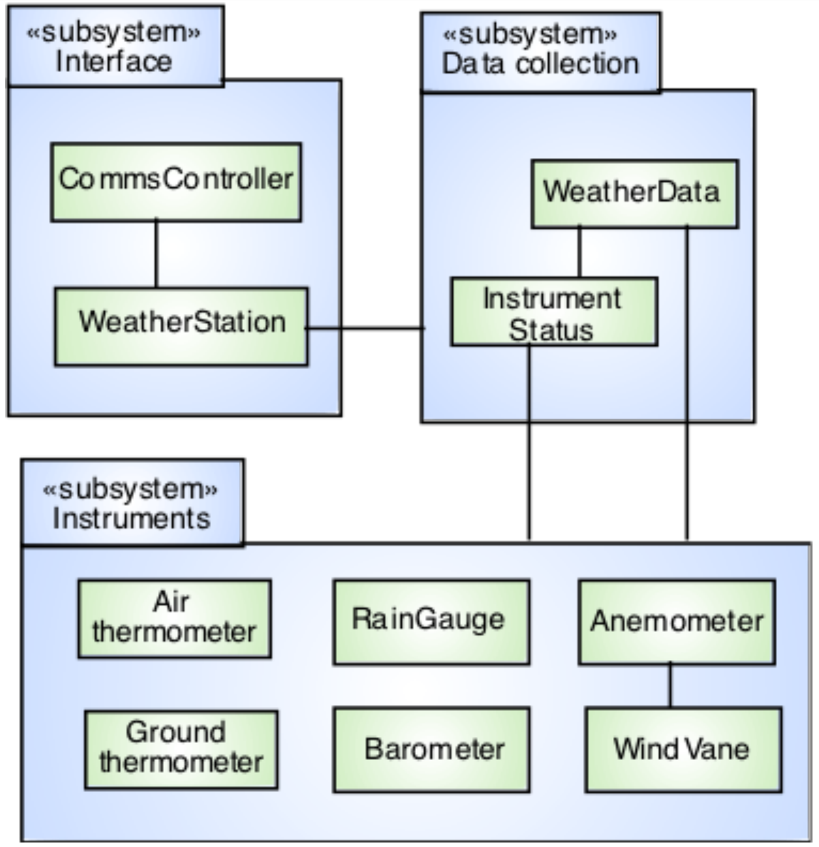
Examples of design models

- Sub-system 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, generalisation models,etc.

Subsystem 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.

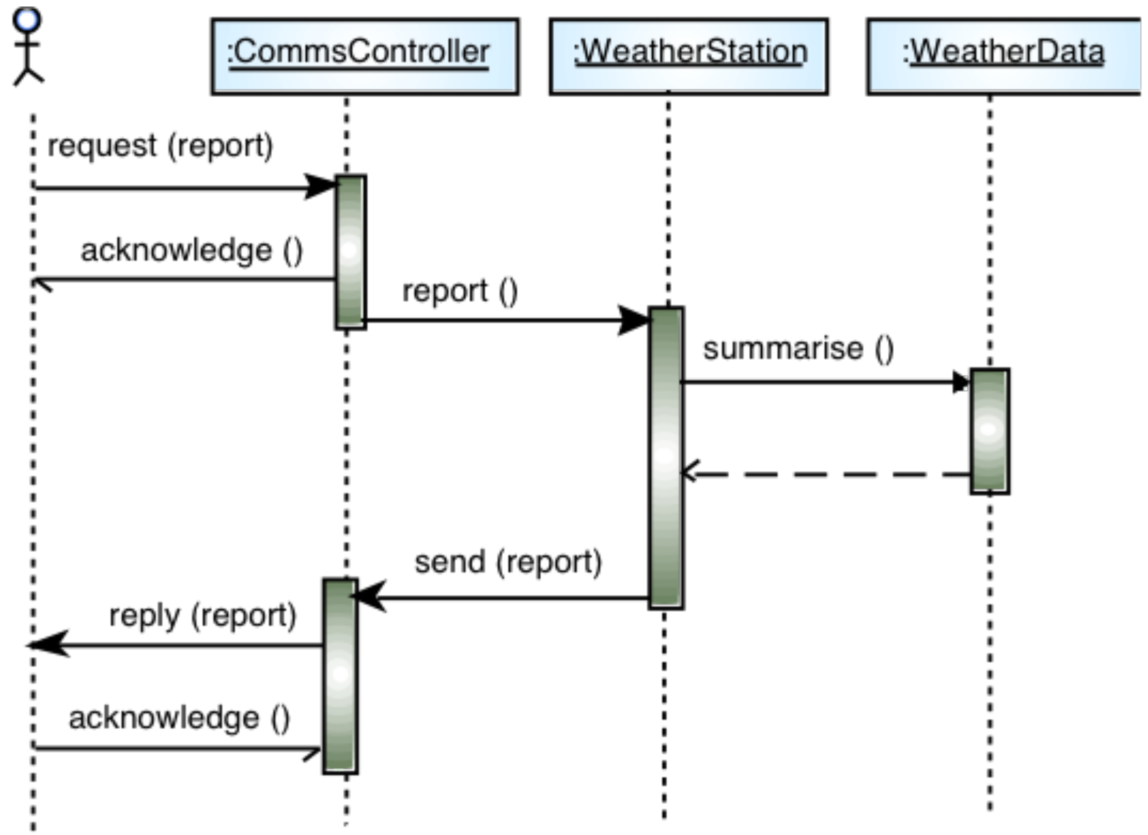
Weather station subsystems



Sequence models

- 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

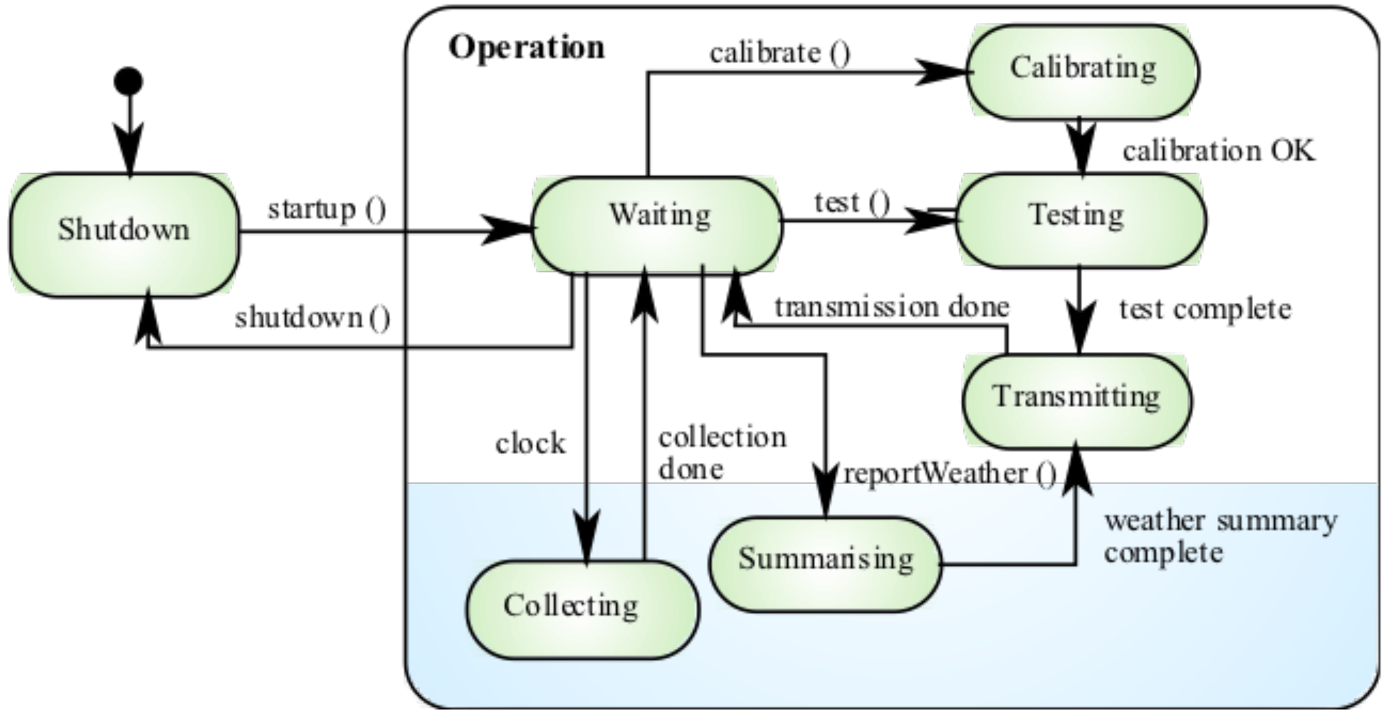
Data collection sequence



Statecharts

- Show how objects respond to different service requests and the state transitions triggered by these requests
- If object state is Shutdown then it responds to a Startup() message
- In the waiting state the object is waiting for further messages
- If reportWeather () then system moves to summarising state
- If calibrate () the system moves to a calibrating state
- A collecting state is entered when a clock signal is received

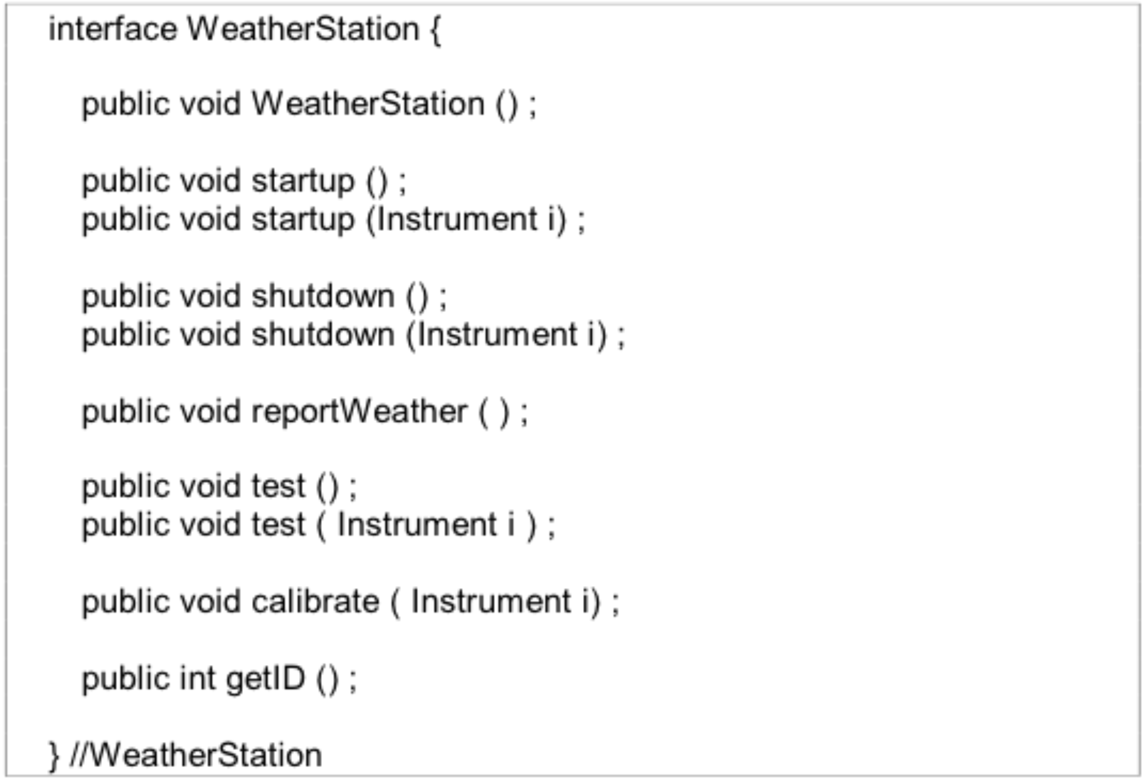
Weather station state diagram



Object 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 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
```

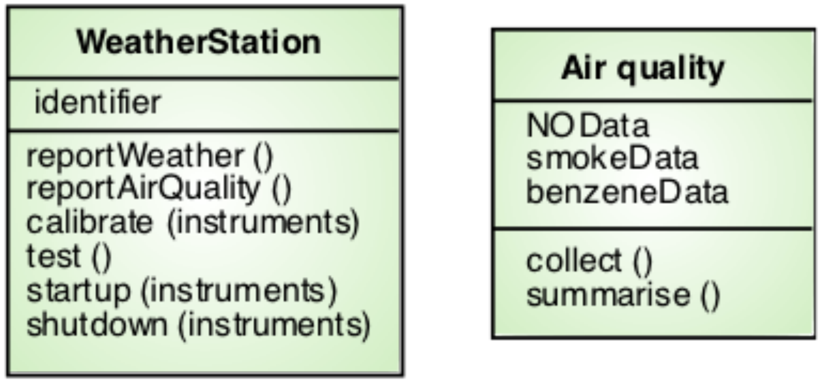
Design evolution

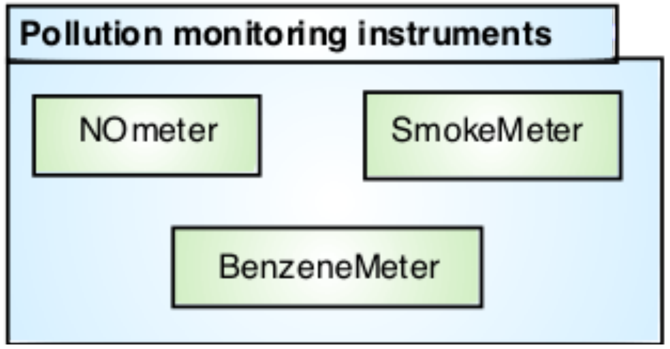
- Hiding information inside objects means that changes made to an object do not affect other objects in an unpredictable way
- Assume pollution monitoring facilities are to be added to weather stations. These sample the air and compute the amount of different pollutants in the atmosphere
- Pollution readings are transmitted with weather data

Changes required

- Add an object class called ‘Air quality’ as part of WeatherStation
- Add an operation reportAirQuality to WeatherStation. Modify the control software to collect pollution readings
- Add objects representing pollution monitoring instruments

Pollution monitoring





Key points

- OOD is an approach to design so that design components have their own private state and operations
- Objects should have constructor and inspection operations. They provide services to other objects
- Objects may be implemented sequentially or concurrently
- The Unified Modeling Language provides different notations for defining different object models

Key points

- A range of different models may be produced during an object-oriented design process. These include static and dynamic system models
- Object interfaces should be defined precisely using e.g. a programming language like Java
- Object-oriented design simplifies system evolution