Object-Oriented System Development

- Overview of object-oriented design (OOD)
- Overview of object-oriented programming (OOP)
- Unified Modeling Language UML
- Object-oriented design process

Object-oriented development

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

Advantages of OOD

- For some systems, there may be an obvious mapping from real world entities to system objects, and so the design is easily understood
- Objects are loosely coupled and can be built-up independently
- Easier maintenance. Objects may be understood as stand-alone entities
- Objects are appropriate reusable components

Disadvantages of OOD

- An object must be explicitly referenced in order to access its services
- If an object's interface is changed, you may need to evaluate the effect on all users of that object
- More complex systems and entities are difficult to represent as objects

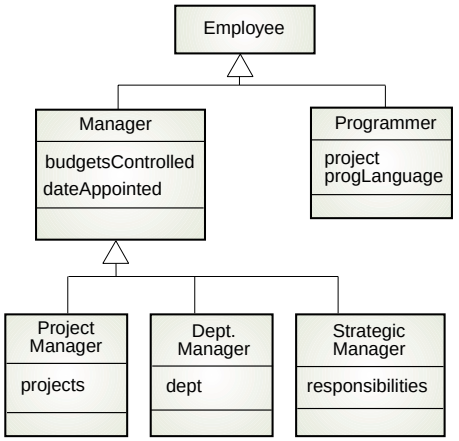
Key contributions of OOP

- Data abstraction
  - Hide the irrelevant details of the data structures. For example, the programmer does need not need to know the data structures used inside the *stack* class, as long as *push* and *pop* work correctly.
- Encapsulation (information hiding)
  - The implementation details are hidden inside. Classes communicate with well-defined interfaces. It is easier to reuse your work.
  - Declare all member variables as private, and instead use accessor functions (constructors, readers, and writers).
- Polymorphism
  - An object responds appropriately to a message based on its type and position in a class hierarchy. The same message (such as *draw*) sent to different classes takes on different forms. Methods are "overloaded."
- Inheritance...

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

An inheritance hierarchy



Advantages of inheritance

- It is an abstraction mechanism which may be used to classify entities
- The inheritance graph is a source of organisational knowledge about domains and systems
- It is a reuse mechanism at both the design and the programming level
- The important contribution to OOP: Well-defined, fully-functioning, debugged source and object code can be instantly re-used

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 graphs created during analysis. Can lead to significant inefficiency (???)
- The inheritance graphs of analysis, design and implementation have different functions and should be separately maintained (???)

The Unified Modeling Language

- A unification of OOA&D methods that appeared in the late 80's and early 90's.
  - It mostly unifies the methods of the three amigos: Booch, Jacobson, and Rumbaugh (object modeling technique, OMT). The Unified Modeling Language is an integration of these notations
- A set of (mostly graphical) notation used to express the analysis and design of OO systems.
- A key component for communicating design ideas.
- There is also a UML process, but the language can be used without the process.
  - The diagrams and notations are meaningful and useful regardless of whether you used the UML process.

What are the major diagrams used in UML?

- Use case diagrams
  - Depict how a user could interact with a system (scenarios) in the process of attempting to achieve a goal.
- Class diagrams
  - Descriptions of the types of objects in the system, and the various kinds of static relationships that exist among them.
- State-transition diagrams
  - Describe the behavior of a system. Show all possible states that an object can get into as a result of events that reach that object.
- Interaction diagrams
  - Describe how groups of objects collaborate in some behavior. Show the sequence of object interactions

An object-oriented design process

- Figure out and define the context and modes of use of the system
- Design the system architecture
- Identify the principal system objects
- Develop interaction and state-transition models
- Specify object interfaces

- (but not necessarily in that order)

The Unified Software Development Process

- A process framework for transforming user requirements into a software system.
  - Use-case driven: All analyses are derived from specific user-system interactions. The requirements analysis asks “What is the system supposed to do for each user?”
  - Architecture-centric: Emphasis on the entire design of the system, with important characteristics made visible, and details hidden inside.
  - Iterative and incremental: In every iteration: Identify and specify relevant use cases, create a design using the chosen architecture as a guide, implement the design in components, and verify that the components satisfy the use cases.
  - Object-oriented, Uses the Unified Modeling Language (UML)

Start with the problem statement: 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.

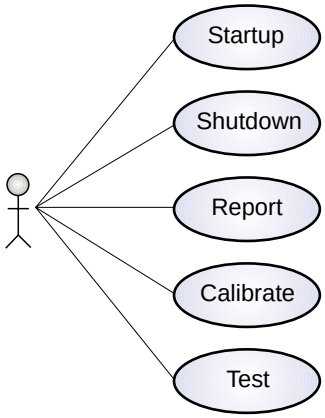
When a command is issued to transmit the weather data, the weather station processes and summarises the collected data. The summarised data is transmitted to the mapping computer when a request is received.

(Why is it important to have a good problem statement?)

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.

Use-cases for the weather station



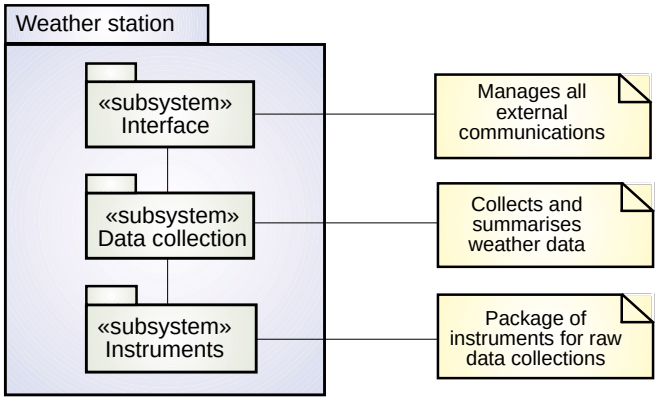
Architectural design

- 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

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 architecture



Question

- How do you figure out the names of the objects and attributes that you will use in a system?

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

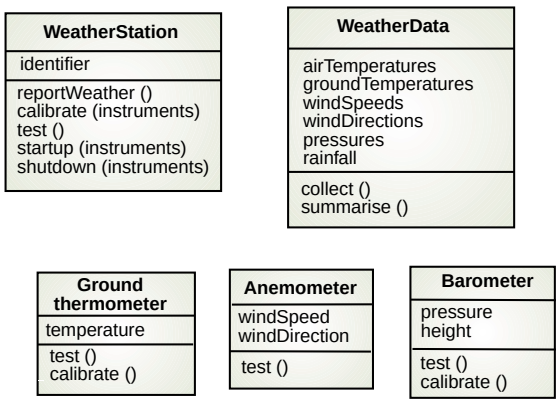
Approaches to object identification

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

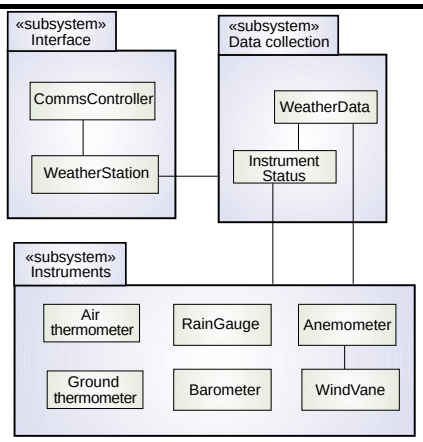
Weather station object classes

- 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



Weather station subsystems



Question

- How do you figure out the names of the methods that you will use in a system?

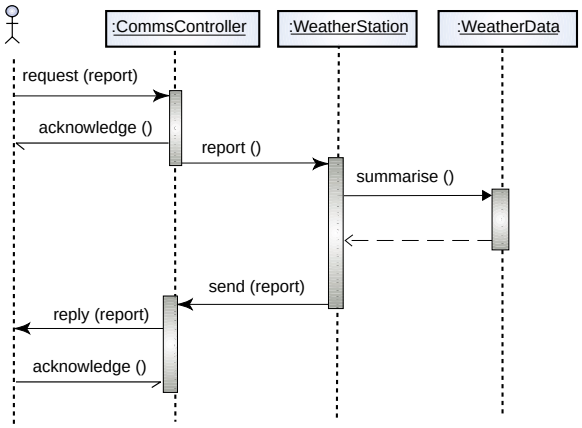
Dynamic models

- Describe the dynamic interactions between objects (as opposed to the static class diagrams)
- State-transition diagrams
  - Describe the behavior of a system. Show all possible states that an object can get into as a result of events that reach that object.
- Interaction diagrams
  - Describe how groups of objects collaborate in some behavior.
  - Show the sequence of object interactions

Interaction Diagrams

- UML calls them “interaction diagrams.” Sommerville calls them “sequence models”
- Interaction diagrams show the sequence of events in which objects in the system interact
  - Objects and users are arranged horizontally across the top.
  - Time is represented vertically so models are read top to bottom
  - Interactions are represented by labeled 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

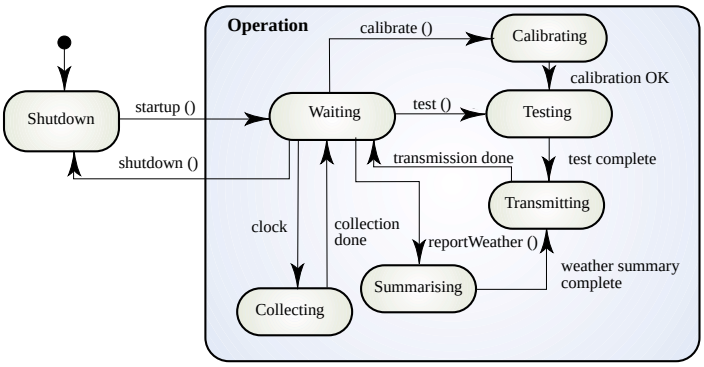
Interaction Diagrams



State-transition diagrams

- Show the events that cause a transition from one state to another, and the actions that result
  - 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-transition 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
- Use class diagrams or just write the code.

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
```


An object-oriented design process

- Figure out and define the context and modes of use of the system
- Design the system architecture
- Identify the principal system objects
- Develop interaction and state-transition models
- Specify object interfaces

- (but not necessarily in that order)

Conclusion

- OOA is a means of capturing, refining, structuring, and describing the requirements.
- OOD is an approach to laying out the system
- OOP is a style of programming that makes key contributions to the re-use of code
- UML provides diagrammatic models and a structured process (set of activities) that can be used in OOA, OOD, and OOP, to transform a user's requirements into a software system.