

# Chapter 14

## Chapter 14 Object-Oriented Design

### Learning Objective

... Designing systems using *self-contained objects* and *object classes*.

Frederick T Sheldon  
Assistant Professor of Computer Science  
Washington State University

---

---

---

---

---

---

---

---

## Objectives

- To explain how a software design may be represented as a set of interacting objects
- To illustrate, with a simple example, the object-oriented design process
- To introduce various models which describe an object-oriented design
- To explain how objects may be represented as concurrent processes

---

---

---

---

---

---

---

---

## Topics covered

- Objects, object classes and inheritance
- Object identification
- An object-oriented design example
- Concurrent objects

---

---

---

---

---

---

---

---

## 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

---

---

---

---

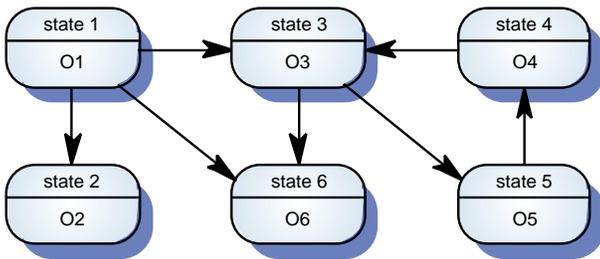
---

---

---

---

## OOD structure



---

---

---

---

---

---

---

---

## 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 realizing an OOD using an OO programming language such as C++

---

---

---

---

---

---

---

---

## Object-oriented design methods

Some methods which were originally based on functions (such as the Yourdon method) have been adapted to object-oriented design. Other methods such as the Booch method have been developed specifically for OOD

HOOD is an object-oriented design method developed to support Ada programming.

JSD has an object-oriented flavor but does not conceal entity state information.

---

---

---

---

---

---

---

---

## OO Design method commonality

The identification of objects, their attributes and services

The organization of objects into an aggregation hierarchy

The construction of dynamic object-use descriptions which show how services are used

The specification of object interfaces

---

---

---

---

---

---

---

---

## Objects, classes and inheritance

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.

---

---

---

---

---

---

---

---

## 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 the printing service associated with lists to print the list L1

*List.Print (L1)*

Call the service associated with integer arrays which finds the maximum value of array XX. Return the result in Max\_value

*IntArray.Max ( XX, Max\_value)*

---

---

---

---

---

---

---

---

## A mail message object class



---

---

---

---

---

---

---

---

## Interface design of mail message

```
package Mailis
type MESSAGE is private ;
-- Object operations
procedure Send (M: MESSAGE; Dest: DESTINATION) ;
procedure Present (M: MESSAGE; D: DEVICE) ;
procedure File (M: MESSAGE; File: FILENAME) ;
procedure Print (M: MESSAGE; D: DEVICE) ;

-- Sender attribute
function Sender (M: MESSAGE) return MAIL_USER ;
procedure Put_sender (M: in out MESSAGE; Sender: MAIL_USER) ;
-- Receiver attribute
function Receiver (M: MESSAGE) return MAIL_USER ;
procedure Put_receiver (M: in out MESSAGE; Receiver: MAIL_USER) ;
-- Access functions and Put operations for other attributes here
...
private
-- The representation of the attributes is concealed by
-- representing it as an access type. Details are inside the package body
type MAIL_MESSAGE_RECORD ;
type MESSAGE is access MAIL_MESSAGE_RECORD ;
end Mail ;
```

---

---

---

---

---

---

---

---

## Interface design of mail message

```
class Mail_message {
public:
    Mail_message ();
    ~Mail_message ();
    void Send ();
    void File (char* filename);
    void Print (char* printer_name);
    void Present (char* device_name);
    char* Sender ();
    void Put_sender (char* S);
    char* Receiver ();
    void Put_receiver (char* R);
    // Other access and inspection functions here
private:
    char* sender, receiver, senderaddr, receiveraddr;
    char* title, text;
    date datesent, datereceived;
};
```

---

---

---

---

---

---

---

---

---

---

## Object definition

### Ada

```
with Mail ;
-- define an object of type mail message by declaring a
-- variable of the specified abstract data type
Office_memo: Mail.MESSAGE ;
-- Call an operation on mail message
Mail.Print (Office_memo, Laser_printer);
```

### C++

```
-- define an object of type Mail_message
Mail_message Office_memo ;

// Call an operation on mail message
Office_memo.Print ("Laser_printer");
```

---

---

---

---

---

---

---

---

---

---

## Inheritance

Objects are members of classes which define attribute types and operations

Classes may be arranged in a class hierarchy where one class is derived from an existing class (super-class)

A sub-class inherits the attributes and operations from its super class and may add new methods or attributes of its own

---

---

---

---

---

---

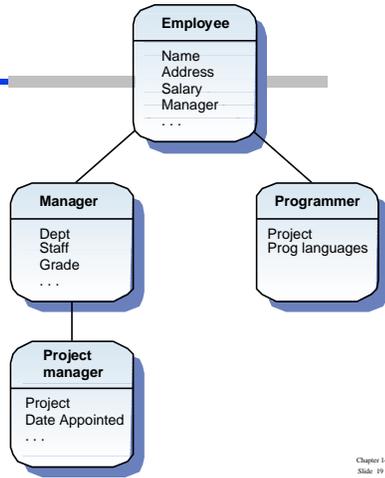
---

---

---

---

## A class or type hierarchy



---

---

---

---

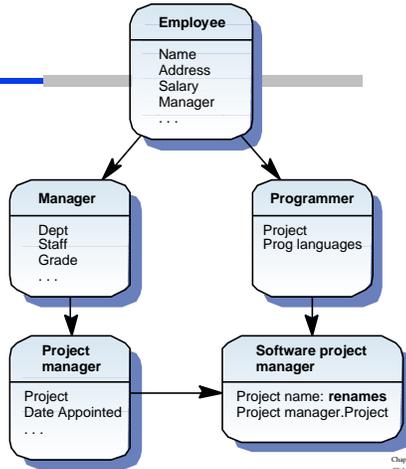
---

---

---

---

## Multiple inheritance



---

---

---

---

---

---

---

---

## 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 organizational 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.

---

---

---

---

---

---

---

---

## Object identification

Identifying objects 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 behavioral approach and identify objects based on what participates in what behavior

Use a scenario-based analysis. Used in the ObjectOry method

---

---

---

---

---

---

---

---

## An office information system

The Office Information **Retrieval System** (OIRS) is an automatic file clerk which can *file documents* under some name in one or more **indexes**, retrieve **documents**, *display* and *maintain document indexes*, archive documents and *destroy documents*. The **system** is activated by a request from the **user** and always *returns* a message to the **user** indicating the success or failure of the request.

---

---

---

---

---

---

---

---

## Objects and operations

Nouns in the description give pointers to objects in the system

Verbs give pointers to operations associated with objects

Approach assumes that the designer has a common sense knowledge of the application domain as not all objects and services are likely to be mentioned in the description

---

---

---

---

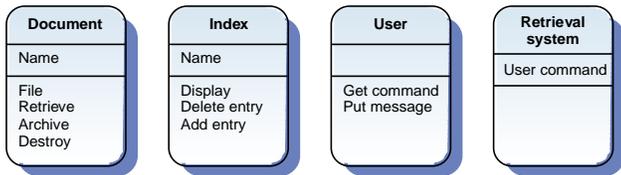
---

---

---

---

## Preliminary object identification



---

---

---

---

---

---

---

---

## A weather mapping system

Takes data from several remote weather stations which perform local data processing

The data is transmitted to an area computer for further processing and integration with other weather reports

Weather maps are generated by the area computer by combining the weather data with a map database

---

---

---

---

---

---

---

---

## 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. Each weather station collects meteorological data over a period and produces summaries of that data. On request, it sends the collected, processed information to an area computer for further processing. Data on the air temperature, the ground temperature, the wind speed and direction, the barometric pressure and the amount of rainfall is collected by each weather station.

Weather stations transmit their data to the area computer in response to a request from that machine. The area computer collates the collected data and integrates it with reports from other sources such as satellites and ships. Using a digitized map database it then generates a set of local weather maps.

---

---

---

---

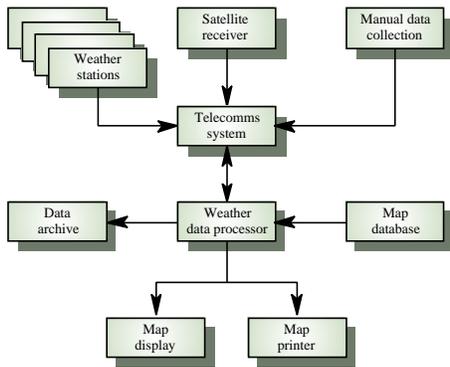
---

---

---

---

## System architecture



---

---

---

---

---

---

---

---

## Principal abstract objects

### Weather station

Package of instruments which collects data, performs some processing and transmits the data for further processing

### Map database

Database of survey information which allows maps to be generated at different scales

### Weather map

A representation of an area with superimposed, summarized weather information

### Weather data

Used to produce the map and is archived for future processing

---

---

---

---

---

---

---

---

## 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 summarizes the collected data. The summarized data is transmitted to the mapping computer when a request is received.

---

---

---

---

---

---

---

---

## Weather station objects

### Identified objects

Air and ground thermometers, anemometer, wind vane, barometer, rain gauge. The package of instruments may also be an object

### Identified operations

Collect data, Perform data processing and Transmit Data

### Identified attributes

Summarized data

This description is refined using domain knowledge  
e.g. a weather station must have a unique identifier

---

---

---

---

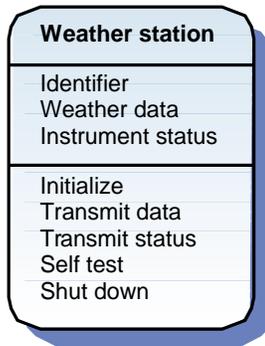
---

---

---

---

## Weather station class



---

---

---

---

---

---

---

---

## Hardware object design

Hardware objects correspond directly to sensors or actuators connected to the system

They conceal the details of the hardware control e.g. buffer address, masking bit pattern etc.

Hardware changes can often be introduced by hardware object re-implementation

---

---

---

---

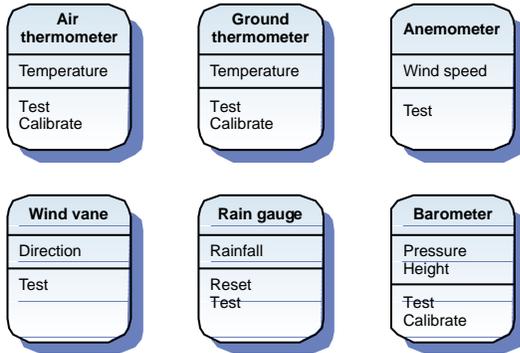
---

---

---

---

## Hardware control objects




---

---

---

---

---

---

---

---

## Data collected by weather station

### Air and ground temperature

Maximum, minimum and average

### Wind speed

Average speed, maximum gust speed

### Wind direction

Every 5 minutes during collection period

### Pressure

Average barometric pressure

### Rainfall

Cumulative rainfall

---

---

---

---

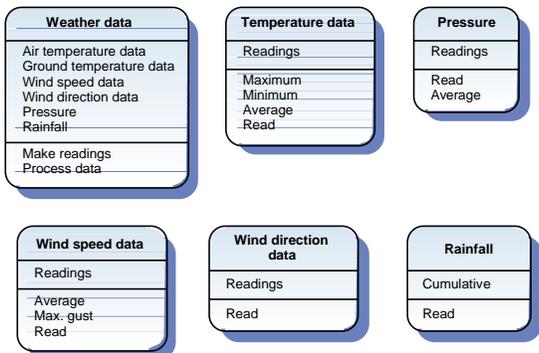
---

---

---

---

## Weather data objects




---

---

---

---

---

---

---

---

## Weather data

All weather data can be encapsulated in a single object. Logically, the weather station transmits a single object to the area computer

The attributes of the weather data object are themselves objects

The Process\_data operation is initiated when weather information is to be transmitted. It computes the information required using raw collected data

---

---

---

---

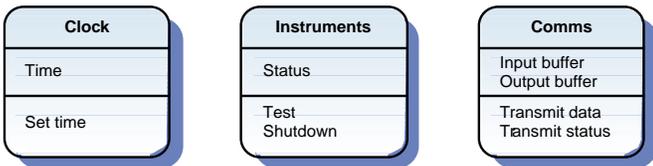
---

---

---

---

## Other weather station objects



---

---

---

---

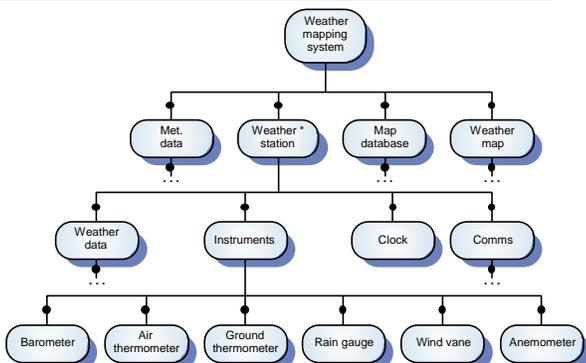
---

---

---

---

## Object aggregation hierarchy



---

---

---

---

---

---

---

---

# Static and dynamic system structure

Object aggregation hierarchy diagrams show the static system structure. They illustrate objects and sub-objects. This is NOT the same as an inheritance hierarchy

Object-service usage diagrams illustrate how objects use other objects. They show the messages passed (procedures called) between objects

---

---

---

---

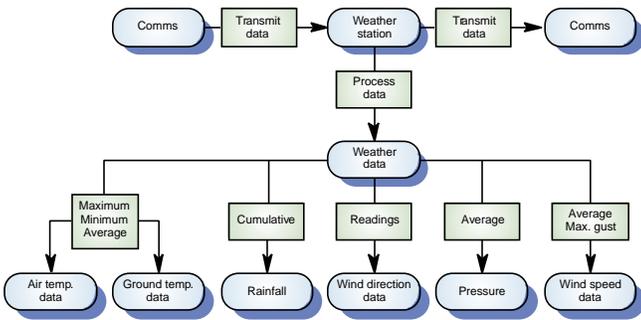
---

---

---

---

# Object interactions



---

---

---

---

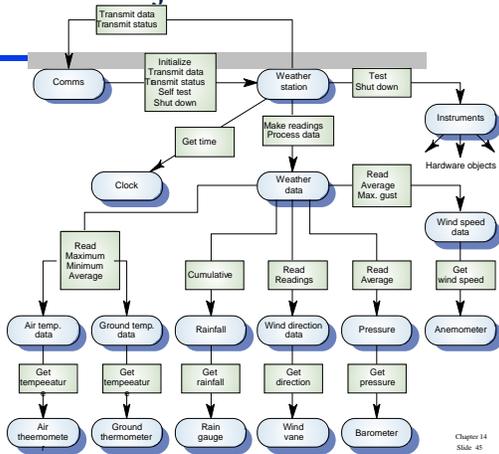
---

---

---

---

# Weather station object interactions



---

---

---

---

---

---

---

---

## Object interface design

Concerned with specifying the detail of the object interfaces. This means defining attribute types and the signatures and semantics of object operations

Representation information should be avoided

Precise specification is essential so a programming language description should be used

---

---

---

---

---

---

---

---

## Ada interface design 1

```
with Weather_data, Instrument_status, Mapping_computer ;
package Weather_station is
  type T is private ;
  type STATION_IDENTIFIER is STRING (1..6) ;
  procedure Initialize (WS: T) ;
  procedure Transmit_data ( Id: STATION_IDENTIFIER ;
                           WR: Weather_data.REC ;
                           Dest: Mapping_computer.ID ) ;
  procedure Transmit_status ( Id: STATION_IDENTIFIER ;
                              IS: Instrument_status.REC ;
                              Dest: Mapping_computer.ID ) ;
  procedure Self_test (WS: T) ;
  procedure Shut_down (WS: T) ;
end Weather_station ;
```

---

---

---

---

---

---

---

---

## Ada interface design 2

```
-- Access and constructor procedures for object attributes
-- Attribute: Station identifier
function Station_identifier (WS: T) return STATION_IDENTIFIER ;
procedure Put_identifier (WS: in out T ; Id: STATION_IDENTIFIER) ;
-- Attribute: Weather data record
function Weather_data (WS: T) return Weather_data.REC ;
procedure Put_weather_data (WS: in out T ; WR: Weather_data.REC) ;
-- Attribute: Instrument status
procedure Put_instrument_status (WS: in out T ; IS: Instrument_status.REC) ;
function Instrument_status (WS: T) return Instrument_status.REC ;
private
  type T is record
    Id: STATION_IDENTIFIER ;
    Weather_data: Weather_data.REC ;
    Instrument_status: Instrument_status.REC ;
  end record ;
end Weather_station ;
```

---

---

---

---

---

---

---

---

## C++ interface design

```
class Weather_station {
public:
    Weather_station ();
    ~Weather_station ();
    void Transmit_data (computer_id dest);
    void Transmit_status (computer_id dest);
    void Self_test ();
    void Shut_down ();

    // Access and constructor functions
    char* Identifier ();
    void Put_identifier (char* Id);
    instrument_status Inst_status ();
    void Put_instrument_status (Instrument_status ISD);
    weather_data_rec Weather_data ();
    void Put_weather_data (weather_data_rec WDR);
private:
    char* station_identifier;
    weather_data_rec Weather_data;
    instrument_status inst_status;
};
```

---

---

---

---

---

---

---

---

---

---

## 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 a Pollution record object.

Add an operation Transmit pollution data to Weather station. Modify control software to collect pollution readings

Add an Air quality sub-object to Pollution record at the same level as Pressure, Rainfall, etc.

Add a hardware object Air quality meter

Adding pollution data collection does NOT affect weather data collection in any way

---

---

---

---

---

---

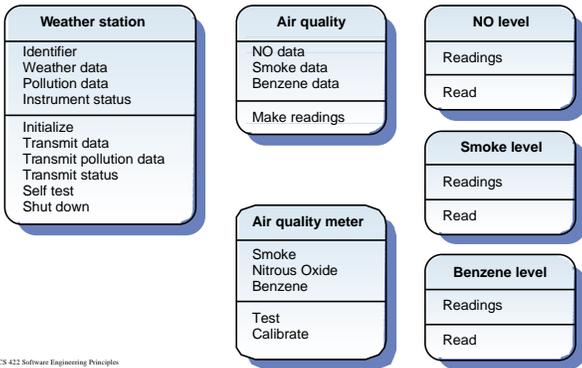
---

---

---

---

## Pollution monitoring objects



---

---

---

---

---

---

---

---

## 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

---

---

---

---

---

---

---

---

## Object implementation

C++ has no built-in concurrency constructs  
Ada's concurrency constructs (tasks) may be used to implement concurrent objects  
Task types represent object classes, tasks represent object instances, task entries represent object operations.  
Task entries are called like procedures

---

---

---

---

---

---

---

---

## Active and passive objects

### Passive objects.

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

### 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

---

---

---

---

---

---

---

---

## Concurrent counter object

Passive object which may be used in a real-time system, associated with system sensors

Operations are Get, Add and Initialize

Counters are created using language declaration mechanisms

Geiger1, Geiger2: Concurrent\_counter

Operations are accessed via entry calls

Geiger1.Get (The\_value); Geiger2.Add (1)

---

---

---

---

---

---

---

---

## Ada counter object

```
task type Concurrent_counter is
  entry Add (N: NATURAL);
  entry Initialize (N: NATURAL);
  entry Get (N: out NATURAL);
end Concurrent_counter;
task body Concurrent_counter is
  Value: NATURAL := 0;
begin
  loop
    select
      accept Add (N: NATURAL) do
        Value := Value + N;
      end Add;
    or
      accept Initialize (N: NATURAL) do
        Value := N;
      end Initialize;
    or
      accept Get (N: out NATURAL) do
        N := Value;
      end Get;
    end select;
  end loop;
end Counter;
```

---

---

---

---

---

---

---

---

## 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

---

---

---

---

---

---

---

---

## An active transponder object

```
task Transponder is
  entry Give_position (Pos: POSITION) ;
end Transponder ;
task body Transponder is
  Current_position: POSITION ;
  C1, C2: Satellite.COORDS ;
  loop
    select
      accept Give_position (Pos: out POSITION) do Pos:= Current_position ;
      end Give_position ;
    else
      C1 := Satellite1.Position ;
      C2 := Satellite2.Position ;
      Current_position := Navigator.Compute (C1, C2) ;
    end select ;
  end loop ;
end Transponder ;
```

---

---

---

---

---

---

---

---

## Key points

OOD is design with *information hiding*.

Representations may be changed without extensive system modifications *...a definite advantage*

An object has a private state with associated *constructor* and *access* operations. Objects provide services (operations) to other objects.

Object identification is a difficult process. *Identifying nouns and verbs in a natural language description can be a useful starting point for object identification.*

---

---

---

---

---

---

---

---

## Key points

---

Object interfaces must be precisely defined. A programming language such as Ada or C++ may be used for this

Useful documentation of an OOD include object hierarchy charts and object interaction diagrams

Objects may be implemented as either sequential or concurrent entities. Concurrent objects may be active or passive

---

---

---

---

---

---

---

---