**Topic: Object-oriented design**

- Relationships among classes: containment, inheritance, reference, friendship, local variable
- Analysis and design of a medical-appointment scheduler
- Design problem: elevator simulator
- Elements of a robust design
- Unified Modeling Language

**Designing for interaction**

- An object interacts with its environment (users, files, ports, other objects)
- Rather than execute an algorithm or single transaction, an object provides ongoing service
- Interactions with other objects occur through messages (member function calls)
- Classes are defined externally by interfaces

**Relationships between classes**

- Containment **has**
- Inheritance **is a kind of**
- Reference or pointer knows about
- Friendship **has access to**
- Local variable **uses**

**O-O analysis example: Scheduling medical appointments**

We wish to manage appointments that patients make with practitioners in a medical office through a receptionist/scheduler. Each practitioner has a daily appointment schedule with times and patient names.

(Adapted from Kip Irvine, C++ & OOP, p. 159)

**Analysis and design: Scheduling application**

Key
- class
- is-a (inheritance)
- has-a (containment)
- knows-about (link)

Script:
- Scheduler requests patient name
- Patient chooses doctor
- Scheduler displays doctor’s schedule
- Patient requests time slot
- Scheduler adds appt. to sched. and to patient recs
Scheduler class interfaces

**Scheduler**
- Schedule appointment
- Schedule all appointments
- Save appointments

**Practitioner**
- Add appointment
- Display appointments

**Daily schedule**
- Add appointment
- Tell whether slot available
- Display appointments

**Patient**
- Input name
- Choose practitioner
- Choose time
- Set appointment

**Appointment**
- Construct from time, practitioner, patient
- Tell whether a time has been set

---

Elevator simulation

**problem-domain description**

A two-story building has an elevator with one door. The elevator may carry one person. Each floor of the building has an up and a down button, used to call the elevator. The elevator has buttons too. When elevator stops at a floor, its door and floor’s doors open, light turns on, bell rings. A clock keeps track of time. At random times, a person arrives to wait for the elevator on random floors.

(Adapted from Deitel/Deitel C++)

---

Elevator simulation

**design exercise**

1. What classes of objects are found in the problem-domain description?
2. What are the attributes of each class?
3. What are the operations associated with each class?
4. What are the relationships among the classes?
5. *(Challenge:)* Begin coding class definitions

---

Elements of a robust design

- Modules (classes) may be developed, tested, deployed independently
- The design may be easily extended. *Example:* in elevator simulator, migrate design easily to support multiple persons, floors, elevators

---

Event-driven programming

- An event is normally the user’s input
- Examples of events: keypress, menu choice, mouse click

---

Model-view-controller architecture

<table>
<thead>
<tr>
<th>Kind of class</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Array of database records</td>
</tr>
<tr>
<td>View</td>
<td>Window Button</td>
</tr>
<tr>
<td>Controller</td>
<td>Menu Instance of application class</td>
</tr>
</tbody>
</table>
Model, view, and controller classes

Use cases and system specification
- Use case: A typical interaction between system and an actor in its environment
- Actor: A role, such as customer, manager, supplier, salesperson
- Actors initiate use cases
- Example of two uses cases (UML use-case diagram):

Software projects start with problem descriptions
- What service must the software provide? Example: Process customer orders from catalog
- What assumptions are made? Example: Some customers may find Web access convenient
- What risks are involved? Example: Some users are inexperienced with Web access

Unified Modeling Language
- A new standard graphical notation for system specification and design
- Motivated by need to depict interactions between systems and their environments, initiated by external actors
- Diagrams include use-case, activity, class, state, interaction
- Supports an object-oriented methodology
- Flowcharts and module hierarchy charts are sufficient to model algorithms. UML models interaction, message passing, interleaved inputs and outputs

UML class diagrams
- Rectangle with 3 horizontal compartments
- Class association diagram

UML statecharts
- After (Harel, 1986)
- Model changing state or mode of a system
- State includes a set of variable values or processes occurring
- Example: Microwave oven
**UML activity diagrams**
- A variation of statecharts
- Model the life of an object
- Similar to flowcharts, but exits from diamonds are labeled with conditions (guards) that may change due to input from environment

**Example:** microwave oven

![Activity Diagram Example](attachment:activity_diagram.png)

**Activity diagrams**
- Depict order in which steps occur
- **Examples:**
  - A decision point (branching)
  - A fork (2 forms A join (execution display at once) converges)

**UML sequence diagrams**
- Depict ongoing interactions (“collaborations”) over time between objects, via messages exchanged
- Microwave oven below has four objects; control object sends messages turn-on, tick, turn-off, ring to the others in response to certain guard conditions

![Sequence Diagram Example](attachment:sequence_diagram.png)