CS 351
Design of Large Programs
Object-Oriented Design Principles

Brooke Chenoweth

University of New Mexico

Spring 2024

A Starting Point

Simplifying assumptions:

- the program execution is sequential
- the program executes on a single machine

The program is hierarchically structured in terms of three levels:

- main program
- subordinate objects
- external devices

Relevant Concepts

Main program

- an *active procedure*
- controls the execution logic
- invokes methods on subordinate objects

Subordinate objects

- are *objects* in the programming sense
- offer public methods to the main program
- do not interact with each other
- have no public fields
- may be instances of some class

Key Relations

- The relation between the main program and the subordinate objects is *reference* relation
    - the entity above may invoke services provided by the entity below i.e., the procedure may call methods on the objects below
- The relation between objects and external devices is *encapsulates* relation
    - an external device is encapsulated by a single object (for now)
    - access to the external device below is controlled by the object above

Where Did Classes Go?

OOD may be used even in the absence of an OOL

Understanding the fundamentals of OOD does not require the class concept.

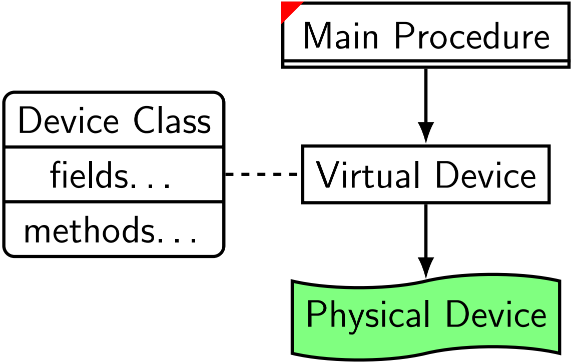
Language support:

- enhances programming productivity
- enriches the design vocabulary
- fosters code reuse

The relation between class definitions and the design is reflected:

- by the fact that objects in the design are instances of classes
- by the mechanics of class definition captured in class diagrams

Notation


Main Procedure
Device Class
fields. . .
methods. . .
Virtual Device
Physical Device

Design Principles

1. Separation of Concerns
2. Information Hiding
3. Data Encapsulation
4. Device Encapsulation
5. Balanced Levels of Abstraction
6. Protection Against Change

1. Separation of Concerns

- The principle of separation of concerns demands that:
  - unrelated concerns should be associated with distinct entities in the design
  - related concerns should be associated with a relevant entity in the design
- This principle impacts design decisions relating to modularity
- Object-oriented design enables the application of this principle
- Strict application of the principle is not always straightforward
- Changes to requirements may have a major impact on the design

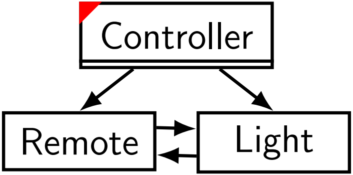
Illustration: Remote Light Control

Consider a light fixture controlled by a remote.

- the light can be turned on and off
- the remote sends a request to turn the light on and off

Remote and Light are natural objects to consider in the design.

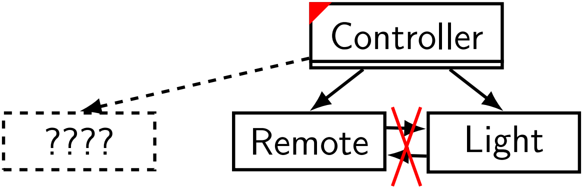
How should these two objects interact with each other?



Did We Get It Right?

What is the impact of:

- adding a light switch on the wall?
- adding a motion sensor for night time use?
- turning off the lights in the morning?



2. Information Hiding

Limit knowledge about design decisions as much as possible.

- fundamental to encapsulation

Postpone design decisions for as long as possible.

- fundamental to top-down design

This relates strongly to the scope of program changes. . . How we can minimize them?

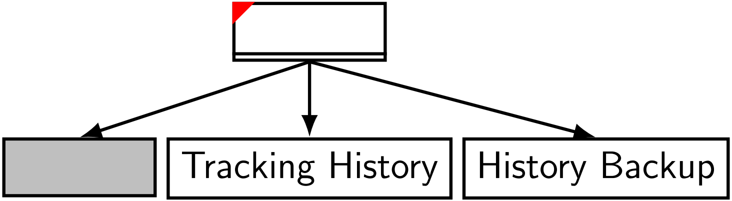
Illustration: Animal Tracking

Consider a system that uses infrared sensing to count animals at night

- an infrared camera takes snapshots at regular intervals
- hot spots in the infrared image are treated as potential distinct specimens
- the number of animals and the time of detection are recorded

Illustration: Animal Tracking

Is the following design employing information hiding?



3. Data Encapsulation

The introduction of the abstract data type accomplished two important objectives:

1. Decoupled implementation details from operations on the data.
   This protects against changes in data storage design.
2. Enabled the definition of programmer-defined data types
   This simplifies programming.

Illustration: Custom Dictionary

Consider an object called MyDictionary:

- initially contains an empty set W of words
- at most N words can be stored
- addWord(w) – adds one word to the set W, if there is room for it
- removeWord(w) – removes one word from the set W
- containsWord(w) — returns true iff the word is in the set W

Simple Implementation: array of strings

Illustration: Custom Dictionary Revisited

Consider the following change in requirements:

- ~~at most N words can be stored~~
- the number of words is very large

This new implemenation requires a tree structure.

4. Device Encapsulation

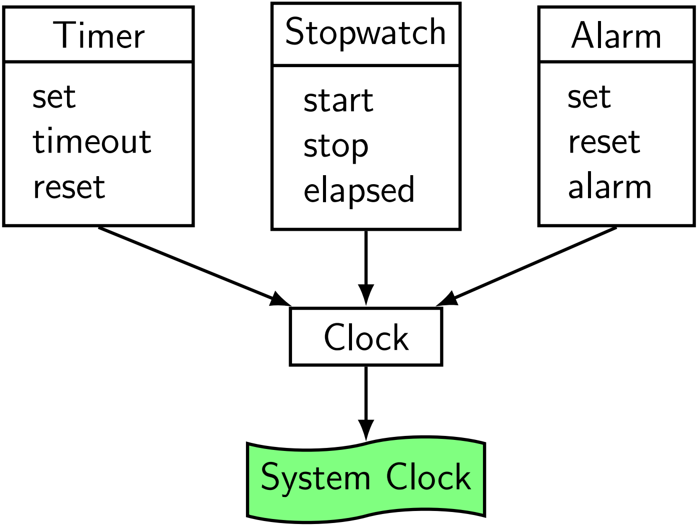
Devices are a volatile element of most designs. Protect the system against device/protocol substitutions:

- microcontroller reassignment of pins
- communication interface (USB connection vs. Ethernet)
- memory mapped I/O vs. interrupt controls

Layers of encapsulation:

- application-specific virtualization
- virtual device
- device driver

Illustration: Timers



Timer

set

timeout

reset

Stopwatch

start

stop

elapsed

Alarm

set

reset

alarm

Clock

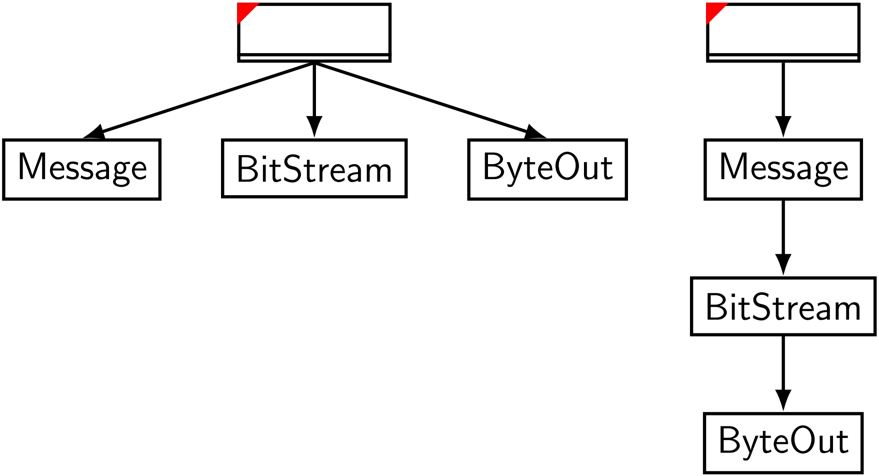
System Clock

5. Balanced Levels of Abstraction

In a hierarchically designed program or system:

- when moving up in the structure the level of abstraction should increase
- when moving down in the structure the level of abstraction should decrease
- entities at the same level in the structure should exhibit comparable degree of abstraction

Illustration: Message Delivery



6 Protection Against Change

The fundamental engineering concern of object-oriented design is *to protect the design and implementation against impact of potential changes*

- modifications to delivered code are expensive
- modifications can introduce errors
- limiting the scope of potential changes reduces cost and mitigates risks

Any proposed design needs to be analyzed with respect to the impact of changes

- processing logic
- processor changes
- device substitution
- elimination of performance bottle necks