Starting Reader #2

Object-oriented perspective

Operating system = computer interface

Shell/libraries/system calls = OS interface

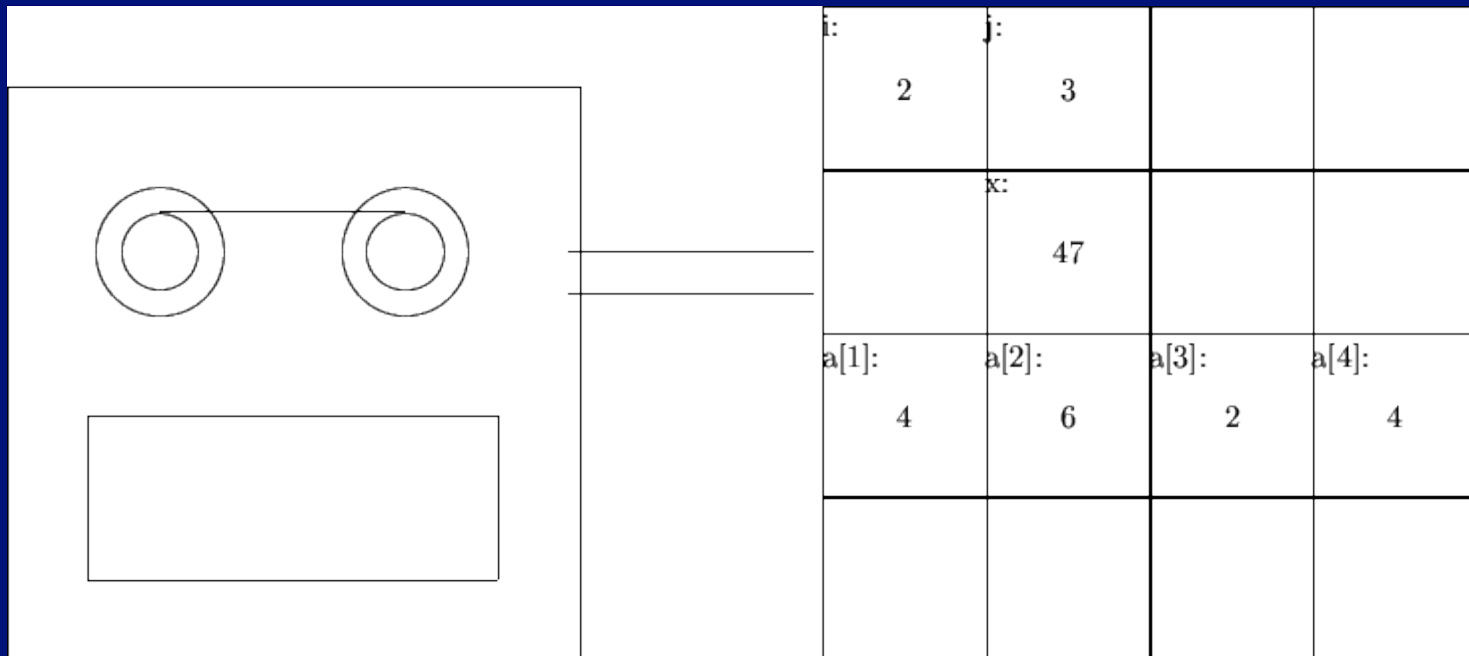
Will return to OS topics in upcoming lectures. Now: OO intro.

Objects

- Include *things*
  - Stack, queue, list, …
  - Window, spaceship, recipe, …
- Also include *concepts*
  - Power, trajectory, mood, …
- Can represent people, places, roles, …
- In programming: an object is a software entity encapsulating data and/or methods

Imperative programming (not OOP)

- Data, and the operations that manage the data are separate entities (physically and *logically*)



- What are implications of this programming style?

Kay's Description of OOP

1. Everything is an object.
2. Objects perform computations by making requests of each other through the passing of messages.
3. Every object has its own memory, which consists of other objects.
4. Every object is an instance of a class. A class groups similar objects.
5. The class is the repository for behavior associated with an object.
6. Classes are organized into a *singly-rooted* tree structure, called an inheritance hierarchy.

Alan Kay: "Simple things should be simple, complex things should be possible."

Solving problems *with* objects

- First decide what objects are needed
  - Instead of what functions are required
  - And instead of how specifically to handle data
- Then give each object responsibilities
  - Which probably include storing some data and performing some functions
- Finally, have objects interact by sending messages (usually method calls) to one another
  - i.e., they collaborate to fulfill responsibilities

T. Budd's "real life" example

- Chris decides to send flowers to his friend Robin
- First he selects an *agent*: Fred, a capable florist
  - Then he sends a message to Fred – not unlike:

```
fred.sendBouquet(1, &robin);
```

- The next step is Fred's responsibility
  - Chris does not participate in this part of the process
  - Likely that *many other agents* do participate though!
- Finally Fred probably sends a message to Chris:

```
chris.pay(bouquetPrice, this);
```

Elements of OOP - Objects

- 1. Everything is an object
  - Actions in OOP are performed by agents, called *instances* or *objects*.
- Several agents in the example scenario, including Chris, Robin, Fred, the florist in Robin's city, driver, flower arranger, grower
  - Each agent has a part to play, and the result is produced when all work together in the solution of a problem.

Elements of OOP - Messages

- 2. Objects perform computations by making requests of each other through the passing of messages.
  - Actions in OOP are produced in response to requests for actions, called *messages*. An instance may accept a message, and in return will perform an action and return a value.
- To begin the process of sending the flowers, Chris gives a message to Fred. She in turn gives a message to the florist in Robin's city, who gives another message to the driver, and so on.

Information hiding

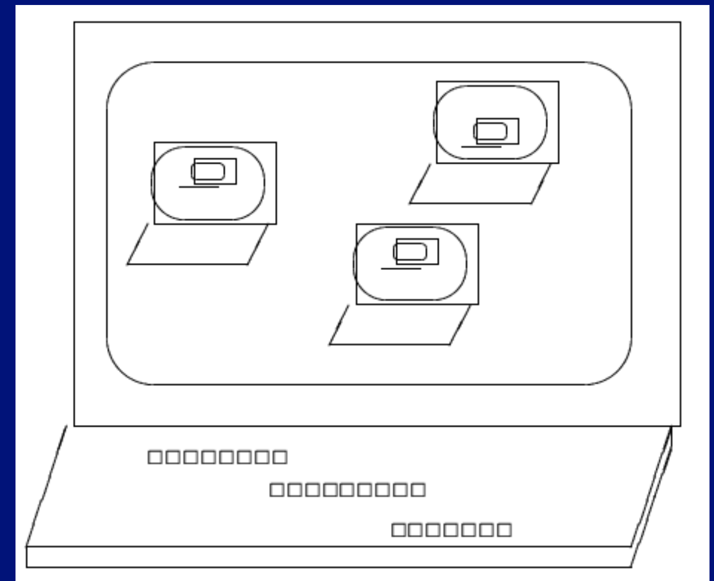
- Notice how a user of a service being provided by an object, need only know the name of the messages that the object will accept.
  - They need not have any idea how the actions performed in response to these requests will be carried out.
- Having accepted a message, an object is responsible for carrying it out.

Receivers and behavior

- Messages differ from traditional function calls in two very important respects:
  a) A designated *receiver* accepts the message
  b) The interpretation of the message may be different, depending upon the receiver
- Although different objects may accept the same message, the actions (*behavior*) the object will perform will likely be different
  – Might not even know what behavior to perform until run-time – a form of *late binding*

Elements of OOP – Recursive Design

- 3. Every object has its own memory, which consists of other objects.
  - The structure of the part mirrors the structure of the larger unit.
- Principle of non-interference: "Ask not what you can do *to* your data structures, but ask what your data structures can do *for* you." (Timothy Budd)

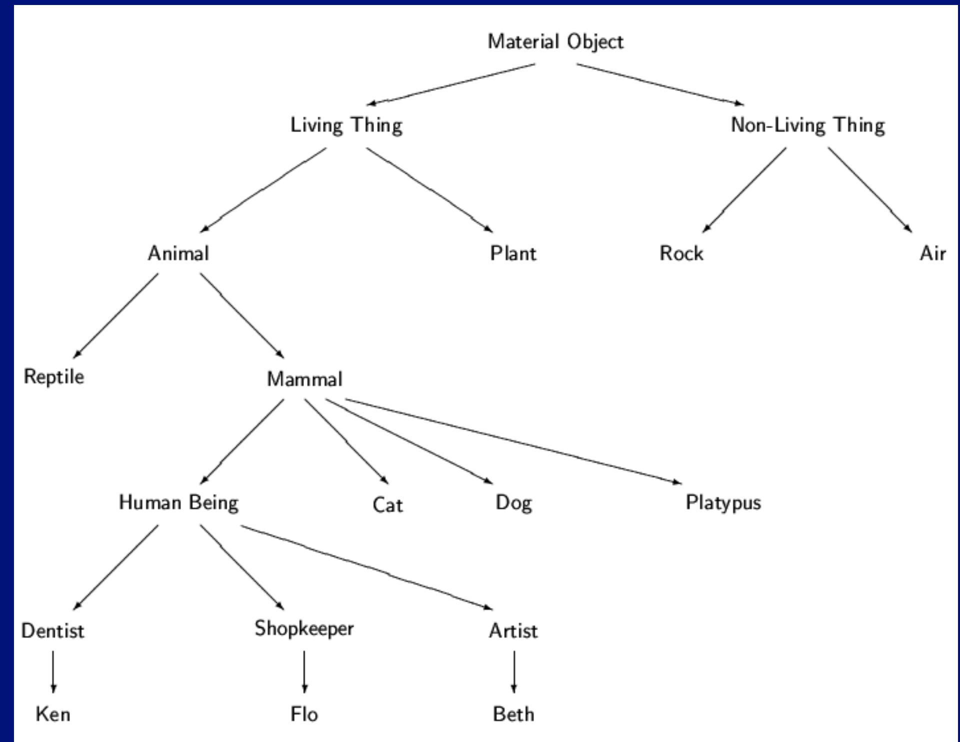


Elements of OOP - Classes

- 4. Every object is an instance of a class. A class groups similar objects.
  - Fred is an *instance* of the *class* Florist
- 5. The class is the repository for behavior associated with an object.
  - All objects that are instances of a class use the same method in response to similar messages.

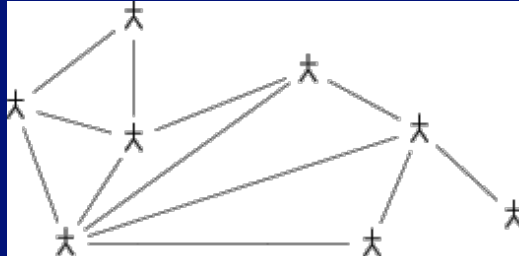
Elements of OOP - Inheritance

- 6. Classes are organized into a *singly-rooted* tree structure, called an inheritance hierarchy
- Data and *general* behavior at one abstraction level extend to lower levels
  - But can *override* behavior (a later topic)



Levels of abstraction 1

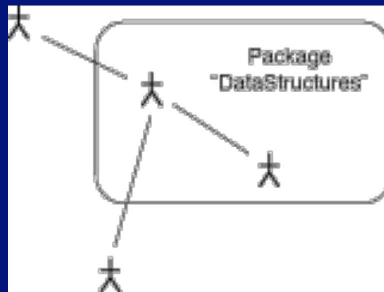
- Communities of interacting objects



  - Internally: within the program system
  - And externally: team of programmers, each responsible for different parts of the system
- Focus here is on *communication at the highest level* of abstraction
  - i.e., lines of communication between the agents

Packages and Namespaces

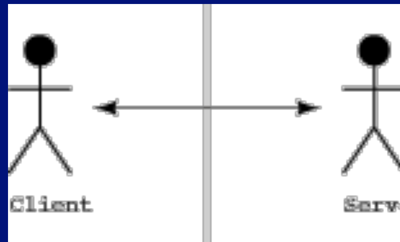
- Used to surround a collection of objects (a small community in itself) with a layer



- To control visibility from outside the module
  - A form of information hiding – promotes low coupling, and thus modifiability, reuse potential, and so on

Levels of abstraction 2

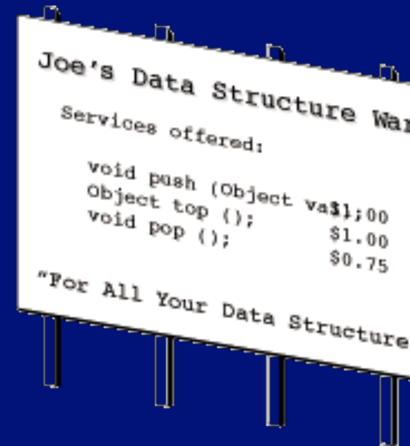
- Clients and servers – abstraction about the relationship between two individual objects



  - Typically one is providing a service, and the other is using the service
- Note: not specifically web servers/clients – a more general idea about interacting objects

Levels of abstraction 3, 4, …

- 3. Describing services
  - Focus is on a server
  - Independent of clients
  - i.e., defining the *interface*



- 4. Implementing the interface – from point of serving the client(s)
- … Implementing individual functions, and other background features about which the clients have no need to know

Finding the right abstraction level

- A critical problem to solve in early stages of development – not easy, and no “right way”
  - Must determine what details are appropriate at each level of abstraction
  - And (often more importantly) must decide what details should be omitted – to be considered later
- Don’t want to ignore important information
  - But don’t want to manage too much information, or have excessive information hide critical details