1: Introduction to Objects

The Progress of Abstraction

- Problem space & solution space
- Alan Kay’s 5 rules for a pure OOP language
  1) Everything is an object
  2) A program is a bunch of objects telling each other what to do by sending messages
  3) Each object has its own memory made up of other objects
  4) Every object has a type
  5) All objects of a particular type can receive the same messages

An Object has an interface

<table>
<thead>
<tr>
<th>Type name</th>
<th>Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface</td>
<td>on()</td>
</tr>
<tr>
<td></td>
<td>off()</td>
</tr>
<tr>
<td></td>
<td>brighten()</td>
</tr>
<tr>
<td></td>
<td>dim()</td>
</tr>
</tbody>
</table>

Characteristics + Behavior

Object:  

```
Light lt = new Light();
```

Request: 

```
lt.on();
```

That’s programming with objects!
The Hidden Implementation
- public, private, protected, and “friendly”

Class Creator
Class User (Client Programmer)

Reusing the Implementation
- Composition: make new objects by combining other objects
  - Type name: Car
    - Interface:
    - Implementation: Engine, Wheel[4], Door[2], etc...

Inheritance: Reusing the interface
- Inheritance: automatically duplicates the interface
- Objects related by inheritance all have the same type (interface)
  - Shape
    - draw()
    - erase()
  - Circle
    - draw()
    - erase()
  - Square
    - draw()
    - erase()
  - Line
    - draw()
    - erase()

(This shows the UML style of drawing diagrams)
Polymorphism

One piece of code ...
... works with all these objects!

void doStuff(Shape s) {
    s.erase();
    // ...
    s.draw();
}

Circle c = new Circle();
Triangle t = new Triangle();
Line l = new Line();
doStuff(c); // "Dynamic binding"
doStuff(t);
doStuff(l);

An Amazing Trick

Object-Oriented programs

• Made up of objects sending messages to each other’s interfaces
OOP Design

- Most designs can be simplified by adding another level of indirection/abstraction

<table>
<thead>
<tr>
<th>Things that Change</th>
<th>Things that Stay the Same</th>
</tr>
</thead>
<tbody>
<tr>
<td>In the design</td>
<td></td>
</tr>
<tr>
<td>At compile-time</td>
<td></td>
</tr>
<tr>
<td>At run-time</td>
<td></td>
</tr>
<tr>
<td>In the environment</td>
<td></td>
</tr>
<tr>
<td>Etc.</td>
<td></td>
</tr>
</tbody>
</table>

- Hard part: decomposing system into objects
- What interfaces do you need?

Object Landscapes & Lifetimes

- How are objects created and destroyed?
- Created in a pool of memory called the heap
- Memory (only) is released by the garbage collector
- Most flexible, simple way to build a system
- Can also cost performance – but not necessarily

Containers & Iterators

- You don’t know how many, what type, lifetime of objects you need
- Where to put objects when you make them?
- Container: generic holder
- Iterator: sequentially selects items in the container (called Enumeration in Java 1.0/1.1, Iterator in Java 2)
Exception Handling for Errors

- Error handling wired in
- Can’t ignore the error
- Recover to a known good state
- Java stands out: you’re forced to use exceptions

Multithreading

- For responsiveness
- For running multiple tasks (animation & networks)
- We’ll cover this in Chapter 14

Persistence

- Objects that live between program invocations
- Convenient, Java has object serialization built in, supports “lightweight persistence.” Primarily added to support remote method invocation (RMI) across networks.
- JDBC could be thought of as another form of persistence
- JavaSpaces, further evolution
Java & the Internet

- A client-server system:
  - Delivers centralized information on demand
  - Also accepts new information, typically into a database (customer info, orders, etc.)
- The web is a big client-server system
  - All the servers are on the same network
  - Client software is a universal "browser"
  - Has been a dumb receiver of information
  - Tedious & wastes bandwidth

Where Java Fits on the Web

- Designed to make interactive client programs, so the Web server doesn't have to do all the work
- Nicer UI, animation, data validation before sending it across the web, provide local answers.
- Example: Plotting data
  - Make a GIF on the server, send it on the net
  - Send a few data points on the net, applet plots them
- The problem of client-side programming on the Web
- Java servlets, JSPs as the solution

Summary

Procedural

Data
Function
Function
Function

Java

Abstract data type
Data
Function
Function
Function

"Message" or "Request"

Member function or "method"

"Encapsulation" (?)