OO Programming Principles

Object-Oriented Analysis and Design

CSCI 6448 - Fall 1998

Kenneth M. Anderson

Goals of the Lecture

- Discuss OO Programming Principles
  - Messages
  - Information Hiding
  - Classes and Instances
  - Inheritance
  - Polymorphism
- Present examples of these concepts

OO Programming Languages

- Support OO Analysis & Design
  - The ability to create new types (classes) allows the solution of a problem be expressed in the vocabulary of the problem domain
- Provide language features that directly support OO principles
  - i.e. the topics covered in this class
- Simula (1967), Smalltalk, C++, Java, Objective-C, Oberon, (many others)...

Alan Kay's Basic Principles

- Everything is an object
- Computation is performed by objects passing messages to each other, requesting actions
  - A message is a request for action plus parameters
- Each object has its own memory
  - Consisting of other objects
- Each object is an instance of a class

Basic Principles, continued

- The class is the repository for behavior associated with an object
  - Each object of a class can respond to the same set of messages
- Classes are organized into a singly rooted tree structure
  - Known as the inheritance hierarchy
  - Memory and behavior associated with a parent class are available to its children

Message Passing

- Objects request actions from other objects
  - These actions are known as operations
  - Example
    - Send flowers to this address
- Methods are implementations of operations
  - Each class can provide a method for each operation or reuse one provided by its ancestors
    - Florist: Send flowers to this address
    - Assistant: Send flowers to this address
    - The same results, different methods

Information Hiding

- The object that makes a request should not know or care how the request is fulfilled
  - When I ask for flowers to be delivered, I don't care how its accomplished, just that the flowers get delivered
  - In general, well designed information hiding leads to lower coupling between objects

Is this information hiding?

```
public class link {
    public anchor[] anchors; -- Java array
    public String name;
}
…
link l = new link();
l.name = "Ken's Link";
anchor a = l.anchors[2];
```

Access Control (Quick Review)

- private String name;
  - not visible to clients;
- protected String name;
  - not visible to clients; visible to subtypes
- public String name;
  - visible to all classes
- String name;
  - visible to all classes within the same package
    - (Java only; C++ defaults to private (I think!))

What about this?

```
public class link {
    private anchor[] anchors;
    private String name;
}
…
Link l = new link();
l.name = “Ken’s Link”; -- not allowed!
anchor a = l.anchors[2]; -- neither is this
```


Getting better...

```
public class link {
    private anchor[] anchors;
    private String name;
    public void setName(String name);
    public String getName();
    public anchor[] getAnchors();
    Public void addAnchor(anchor a);
    Public void removeAnchor(anchor a);
}
```


Why is this better?

- The link class has gained control over its attributes
  - Clients can no longer reach in and arbitrarily change the link’s state
  - The class can now enforce policies such as
    - Legal formats for link names
    - Versioning of values
    - Maximum number of anchors, etc.

How might it be improved?

- The class has exposed the implementation of how it stores anchors
  - public anchors[] getAnchors()
- In fact, with Java, the operation getAnchors() returns a reference to the supposedly private anchors attribute!
- It would be better to hide the implementation method from the outside

Getting better...

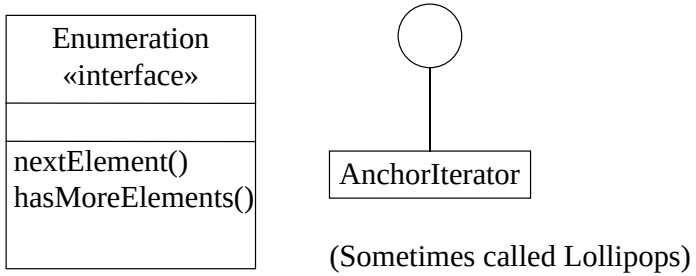
```
public class link {
    private anchor[] anchors;
    private String name;
    public void setName(String name);
    public String getName();
    public Enumeration getAnchors();
    Public void addAnchor(anchor a);
    Public void removeAnchor(anchor a);
}
```


Interfaces

- Enumeration is a pre-defined Java interface
- An interface simply defines operations (and constants), it provides no implementation
- Objects implement interfaces providing an implementation for the standard interface

```
public interface Enumeration {
    public boolean hasMoreElements()
    public Object nextElement()
}
```


UML Notation for Interfaces



(Sometimes called Lollipops)

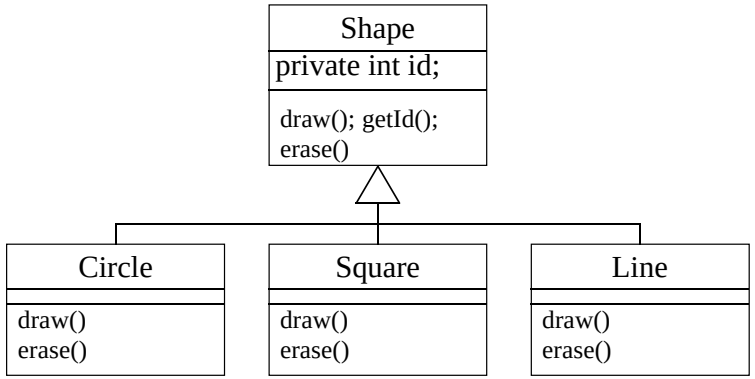
Benefits of information hiding

- With the implementation details hidden
  - We can now change the details at will!
    - Replace the array with a linked list, binary tree, hash table, etc.
    - Or, create a connection with a database and issue an SQL query!
  - As long as the interface doesn't change
    - Clients are never effected
    - They don't even have to be recompiled!

Inheritance

- Inheritance is an implementation of the abstract concept of generalization
- Subtypes inherit attributes and operations from supertypes
- In C++ and Java, inheritance allows classes to share both interfaces and implementations
  - Interfaces allow the sharing of interfaces without sharing implementations

Example (The Famous one…)



Each subtype overrides the methods of the draw() and erase() methods provided by Shape. getId() is reused by the subtypes however. id is hidden.

Does this code work?

```
public void update(Shape s) {
    s.erase();
    s.draw();
}
…
Circle c = new Circle();
Square s = new Square();
Line l = new Line();
update(c); update(s); update(l)
```

How? Polymorphism

- Roughly translated: “Many forms”
- Substitutability
  - A subtype can be used in place of its parent
- Dynamic Binding
  - The method used for an operation is not known at compile time
- Thus, update calls the “right” method based on the type of the instance being pointed at by a variable

Benefits of Polymorphism

- Reusability
  - The update algorithm will work on any shape passed to it
- Extensibility
  - We can freely create new subtypes for Shape and never have to modify the update algorithm