

# Data Abstraction and Object-Oriented Languages

Prof. Evan Chang  
Meeting 27, CSCI 3155, Fall 2009



## Announcements

- HW10 out, practice only
- Review for final on Thu. Come with questions.
- Final preparation
  - Midterm 1 and 2
  - Homeworks and Project
  - Skills List
  - Midterms/finals from prior semesters
- TA FCQs in recitation today

2

## Review Data Abstraction




## One-Slide Summary

- An **abstract data type (ADT)** is a type definition with a set of operations manipulating values of that type
- The **representation** or **implementation type** is hidden from clients. This is known as **information hiding**.
- An ADT allow developers to impose a **representation invariant**—a property of the data structure preserved across operations
- Objects give rise to a related but different form of data abstraction

4

## User-Defined ADTs

- Example: A stack of values

an empty value   
push a new value   
pop a value 

5

## ADTs in SML

- Signatures and Structures

6

## Data Abstraction in OO Languages

- Java-like languages permit information hiding through access control

```

class T {
  public int x;
}

class S {
  private int x;
  public int getX() { ... }
  public void setX(int y) { ... }
}

t.x = ...
... = t.x;

s.x = ... (does not work)
... = s.getX()

// accesses
// public-private promoted - only one & only addresses
// 'friend' 'private'
  
```

7

## (Implementation) Information Hidden?

```

class Stack {
  private public int a[];
  private public int topIndex;
  public void push (int v) { ... }
  public int pop () { ... }
}

// client code
Stack s = new Stack();
s.a[0] = ... x not allowed
s.push(5) ✓
  
```

8

## Why is Information Hiding Useful?

- user doesn't do more than it says
  - ⇒ can't violate any invariant you have *Representation Invariant*
- can change representation (hidden fields) without affecting client code

9

## What is different about Abstract Data Types and Objects with public interfaces?

- i.e., SML-style data abstraction and Java-style data abstraction
  - SML - signatures *(with abstract types)* (not comparable)
  - Java - access control (public, private) (not comparable)

10

## What is different about Abstract Data Types and Objects with public interfaces?

- i.e., SML-style data abstraction and Java-style data abstraction

11

## One-Slide Summary

- An **abstract data type (ADT)** is a type definition with a set of operations manipulating values of that type
- The **representation** or **implementation type** is hidden from clients. This is known as **information hiding**.
- An ADT allow developers to impose a **representation invariant**—a property of the data structure preserved across operations
- Objects give rise to a related but different form of data abstraction

12

### About Your Classmates

- Favorite class is Philosophy of Science
- Lived in China
- Works on cars (used to have a Mustang)
- Enjoys poking old people softly to see whether or not they are dead

13

## Class-Based Object-Oriented Languages

### One-Slide Summary

- An **object** is a record of data fields and operations.
- Operations on objects are called **methods** (or messages). Methods are self-referential: they can refer to the object through a "self" reference (this in Java)
- Inheritance is a way to reuse code and is a defining feature of class-based object-oriented languages.
- A class that is defined through inheritance is a **derived class** or a **subclass** (inherits from a **parent class** or a **superclass**).
- **Dynamic dispatch** (or dynamic binding of messages, virtual method dispatch) with **overridden methods** is another defining feature

15

### Inheritance


*For subclassing = subtyping*

- Want to define a class B that is just slightly different from class A. What are things you might want to change?
  - ✓ • add fields (tags, ...)
  - ✗ • change type of a field
  - ✓ • add methods
  - ✓/✗ • change method implementations
  - ✗ • delete field
  - ✗ • delete method

16

### Inheritance

```
class C {  
  pub int x;  
  pub void f() { ... }  
}  
  
class D extends C {  
  m + y  
  void f() { ... }  
}  
  
D d = new D();  
d.f();
```



17

### Does subclassing imply subtyping?

- Adding methods? ✓ Yes

18

### Does subclassing imply subtyping?

- Dropping methods? *No*

$\{f, g\} \subset \{f\}$

19

### Does subclassing imply subtyping?

- Overriding methods? *Type-wise YES Behavior-wise no?*

```
class C {  
    void p() { print "I'm C"; }  
}  
class D {  
    void p() { print "I'm D"; }  
}  
-----  
 $\{p\} \subset \{p\}$ 
```

20

### Does subclassing imply subtyping?

- Overriding methods with different types?

$C \{ T m(A, a) \dots \}$   
 $D \text{ extends } C \{ U m(B, a) \dots \}$   
-----  
most generally, if  $A \subset B$  and  $U \subset T$

Java  $U \subset T$  and  $A = B$  (I believe)  
C++  $U = T$  and  $A = B$  *But if  $B \subset A$  (uses runtime checks)*

### Dynamic dispatch

```
class C { (virtual) void p() { print "I'm C"; }  
class D extends C { (override) void p() { print "I'm D"; }  
}  
C c = new D();  
c.p(); with dynamic dispatch, it prints "I'm D"
```

$\text{void m}(C c) \{ c.p(); \}$

22

### For Next Time

- Questions for review session

23