Encapsulation and Inheritance

Amer Diwan

Overview

• Looks at potential problems with inheritance
• Encapsulation
  – Hide internal representation
• Data abstraction
  – Data is manipulated via a set of abstract operations
• Does inheritance break encapsulation?
  – Can I modify the “private” parts of a class without affecting its clients?
Clients of classes

• Two kinds
  – code that interacts with instances of the class
  – classes that inherit from the class
• Should not compromise encapsulation for both kinds of clients
  – Most languages focus only on the first kind

Issues: Instance Variable Access
Option 1: Child accesses all

• Example:
  class T { int i; int j; }
  class S : T { m() {i = i + j; } }

• Disadvantages:
• Advantages:
Issues: Instance Variable Access
Option 2: Accessed through methods

• Example:
  
  ```
  class T {
    private: int i; int j;
    protected: int get_i(); int get_j(); }
  ```

• Advantages:

• Disadvantages:

Inheritance for subtyping

• Advantages: Simple (single mechanism), simpler implementation

• Disadvantages: exposes inheritance structure

• Example language: C++, M3
Problem: inheritance for subtypes

- class T { void print() {...} };
  class Set extends T { void print() {...} };
  class SortedList extends T { void print() {...} };
  T a[];  a[0] = new Set; a[1] = new SortedList;

- Let’s change the implementation of SortedList
  class SortedList extends Collection {
      void print() {...}
  }

- What’s the problem here?

Analysis of the problem

- Programmer used inheritance as a implementation reuse mechanism
- Client perceived inheritance as a subtyping mechanism
A solution from the client

• Object a[];
  a[0] = new Set;
  a[1] = new SortedList;
• Problem: cannot do
  a[i].print()
• Casting will be very awkward too!

Using Java’s interfaces

• interface hasAPrint { void print(); }  
  class Set extends implements hasAPrint
    { void print() {...} };  
  class SortedList implements hasAPrint
    { void print() {...} };  
  hasAPrint a[];
  a[0] = new Set; a[1] = new SortedList;
• Clients only see interfaces and not implementation
Multiple inheritance

- **Conflicts**
  - Two methods have the same name and signature
- **How to deal with conflicts?**
  - The mechanism used may expose the “hidden” inheritance structure

Conflict resolution-1

- Disallow unless from same base class

![Diagram showing conflict resolution in multiple inheritance]
Conflict resolution-2

Redefine locally

Further resolution techniques

- Select “first” definition
- Linearize class hierarchy
- Convert to tree
  - When combined with “redefine locally” alleviates situation when an operation is performed multiple times
Next lecture

- Type inference
- Reading: Scott 7.2.5 (handout)