CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2007
Doc 9 Object Coupling
Feb 22, 2007
References

Object Coupling and Object Cohesion, chapter 7 of Essays on Object-Oriented Software Engineering, Vol. 1, Berard, Prentice-Hall, 1993, pp 92-111
Object Coupling
  
  Interface Coupling

  Internal Coupling

  Inside Internal Coupling

  Outside Internal Coupling

    From the side

    From Underneath
Internal Coupling & Cohesion

Internal Coupling
Physical relationships among the items that comprise an object

Cohesion
Logical relationships among the items that comprise an object
Interface Coupling

One object refers to another specific object, and the original object makes direct references to one or more items in the specific object's public interface

Includes module coupling already covered

Weakest form of object coupling, but has wide variation

Issues
- Object abstraction decoupling
- Selector decoupling
- Constructor decoupling
- Iterator decoupling
Object Abstraction Decoupling

Assumptions that one object makes about a category of other objects are isolated and used as parameters to instantiate the original object.

C++/Java 1.5 Example

class LinkedListCell {
    int cellItem;
    LinkedListCell* next;

    // code can now use fact that cellItem is an int
    if (cellItem == 5) print("We Win");
}

template <class type>
class LinkedListCell#2 {
    type cellItem;
    LinkedListCell* next;

    // code does not know the type, it is just a cell item,
    // it becomes an abstraction
}

Selector Decoupling

Counter Example

class Counter{
    int count = 0;

    public void increment() { count++; }
    public void reset() { count = 0; }
    public void display() {
        // Java Swing code to display the counter
        // in a slider bar
    }
}

Selector Decoupled

class Counter{
    int count = 0;

    public void increment() { count++; }
    public void reset() { count = 0; }
    public int count() { return count; }
    public String toString() { return String.valueOf( count ); }
}
Primitive Methods

Any method that cannot be implemented simply, efficiently, and reliably without knowledge of the underlying implementation of the object

Functionally cohesive, they perform a single specific function

Small, seldom exceed five "lines of code"

Types

Selectors (get operations)
Constructors (not the same as class constructors)
Iterators
Composite method

Any method constructed from two or more primitive methods

sometimes from different objects
Selectors

Return state information about their encapsulated object and
Do not alter the state of their encapsulated object

```java
public void display() {
    Swing GUI code to display the counter
}

public String toString() {return String.valueOf( count );}
```

Selector
decoupling
Primitive Objects

Primitive objects are objects that are both:

- Defined in the standard for the implementation language
- Globally known

Primitive objects don't count in coupling with other objects

Why not?
Constructors

Operations that construct a new, or altered version of an object

```java
class Calendar {
    public void getMonth( from where, or what) { blah }
}

class Calendar {
    public static Calendar fromString( String date ) { blah}
}
```
Composite Object

Object *conceptually* composed of two or more objects

**Heterogeneous Composite Object**

Object *conceptually* composed from objects which are not all *conceptually* the same

```java
class Date{
    int year;
    int month;
    int day;
}
```

**Homogeneous Composite Object**

Object *conceptually* composed from objects which are all *conceptually* the same

list of names - each item is a member of the same general category of object – a name
Iterator

Allows the user to visit all the nodes in a homogeneous composite object and to perform some user-supplied operation at each node.
Object Coupling

Interface Coupling

Internal Coupling

Inside Internal Coupling

Outside Internal Coupling

From the side

From Underneath
Inside Internal Object Coupling

Coupling between state and operations of an object

The big issue: Accessing state

Changing the structure of the state of an object requires changing all operations that access the state including operations in subclasses

Solution: Access state via access operations

C++ implementation
   Provide private functions to access and change each data member
Outside Internal Coupling from Underneath

Coupling between a class and subclass involving private state and private operations

Major Issues

Access to inherited state
  Direct access to inherited state
  Access via operations

Unwanted Inheritance

  Parent class may have operations and state not needed by subclass
Outside Internal Coupling from the Side

Class A accesses private state or private operations of class B

Class A and B are not related via inheritance

**Main causes**

Using non-object-oriented languages
Special language "features"
C++ friends