CS 535 Object-Oriented Programming & Design
Fall Semester, 2000
Doc 18 Heuristics: Objects, Classes, Inheritance
Contents

Reference

Object-Oriented Design Heuristics, Riel, 1996

Classes verses Objects

2.11 Be sure that the abstractions that you model are classes and not simply the roles that object play

5.15 Do not turn objects of a class into derived classes of the class. Be very suspicious of any derived class for which there is only one instance.

Is Mother a

- Role for a Person object or
- Subclass of the Person class

Is USCurrency a

- Role for a Currency object or
- Subclass of Currency class

Does Mother have different responsibilities than Person

How different are the responsibilities?

Sign of Violating the Heuristic

A class has many subclasses &

Subclasses tend to have one instance

3.9 Do not turn an operation into a class

Be suspicious of any class whose name is a verb or is derived from a verb, especially those that have only one piece of meaningful behavior

Gets, sets and toString methods do not count as meaningful behavior

```
public CurrencyAdd {
   int add( int a, int b ) { blah}
}

public CurrencyOperations  {
   int add( int a, int b ) { blah}
   int subtract( int a, int b ) { blah}
etc.
}
```

Signs of Violating the Heuristic

Class name is a verb
Class contains operations but no data
All class methods are static (or could be made static)

3.2 Do not create god classes/objects in your systems

Be vary suspicious of a class whose name contains Driver, Manager, System, or Subsystem

God class

God Class/Object

- An object that performs most of the work
- Rest of the classes become trivial struct like classes

Occurs when designing procedurally

Inheritance
Terms

Derived class means child class or subclass

Base class means parent class or superclass

Some consider subclass & superclass ambiguous

A child class has can have more fields and methods

So:

- A parent class has a subset of the methods/fields of the child
- A child class is a superset of the parent

5.2 Derived classes must have knowledge of their base classes by definition, but base classes should not know anything about their derived classes.

If a base class knows about it derived classes then adding a new derived class means changing the base class

5.3 All data in a base class should be private; do not use protected data.

Use protected methods to access the base data

A base class can have many derived classes that may be written by other people

Private data allows a base class to change data representations without affecting derived classes

5.4 In theory, inheritance hierarchies should be deep – the deeper the better

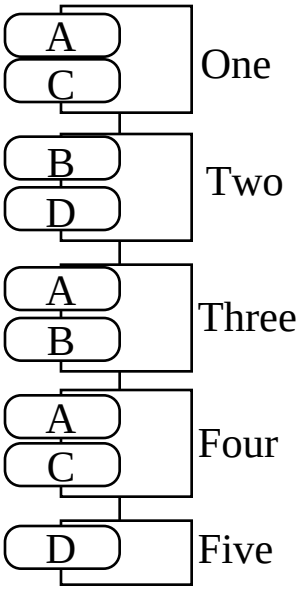
Deeper hierarchies provide more abstractions

5.5 In practice, inheritance hierarchies should be no deeper than about 6 levels.

Deep hierarchies can be hard to understand and maintain

Yo-yo Problem

	Function			
Class	A	B	C	D
One	Print A, 1 B(); C();		Print C, 1	
Two		Print B, 2 D();		Print D, 2
Three	Print A, 3 Two::A();	Print B, 3 Two::B();		
Four	Print A, 4 Three::A();		Print C, 4 Three::C();	
Five				Print D, 5 Four::D();



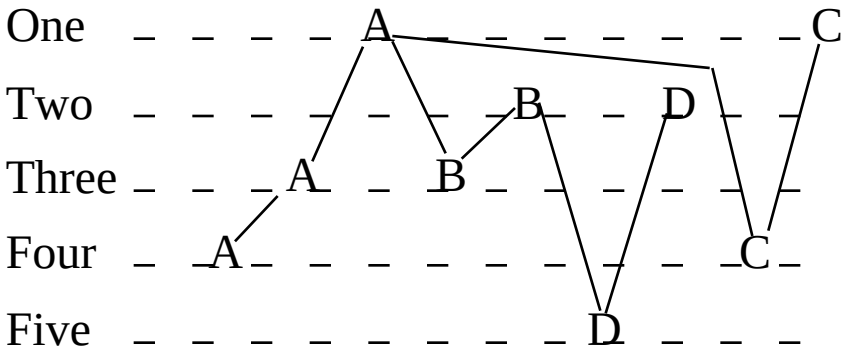
Yo-yo Problem

What is the result of the following program?

```
main()
{
   Five  yoyo;

   yoyo.A();
};
```

Control Flow Trace for Function A in Class Five



Output of Program

A, 4	D, 5
A, 3	D, 2
A, 1	C, 4
B, 3	C, 1
B, 2	

5.6 All abstract classes must be base classes.

An abstract class can not be instantiated

5.7 All base classes should be abstract classes.

Let B be the parent (base) class of C

Let B and C be concrete classes

What happens when B needs a change that does not apply to C?

If A was base class for B and C then B could vary independently of C

5.8 Factor the commonality of data, behavior, and/or interface as high as possible in the inheritance hierarchy.

Allow as many classes as possible to use the common behavior and data

5.9 If two or more classes share only common data (no common behavior), then that common data should be placed in a class that will be contained by each sharing class.

5.10 If two or more classes share common data and behavior, then those classes should inherit from a common base class that captures those data and methods.

5.11 If two or more classes share only a common interface (messages not methods), then they should inherit from a common base class only if they will be used polymorphically.

5.12 Explicit case analysis on the type of an object is usually an error. Polymorphism should be used in most of these cases.

In Java avoid

```
if x instanceof String
   blah
else if x instanceof Vector
   more blah
else if x instanceof StudentRecord
   even more blah
```

6.1 If you have an example of multiple inheritance in your design, assume you have made a mistake and prove otherwise

Multiple inheritance is used more often than it is needed