CIS W338

Introduction to OO Design
Type Conformance, Closed Behavior, and the Perils of Inheritance and Polymorphism

A robust class hierarchy

• A robust class hierarchy is heavy-duty and a joy forever: Objects of a class will happily stand in for objects of a superclass, and the hierarchy as a whole will withstand the trials of time.
The Usual Case

Classes, classes, everywhere
More than you’d ever think.
Classes, classes, everywhere
Nor any two would link.

Type vs. Class

• Type --
  – Defined by its external interface.
  – State space
  – Behavior.
• Class --
  – Implements a type
Principal of Type Conformance

If S is a true subtype of T, then S must conform to T. An object of S can be provided in any context where an object if type T is expected, and correctness is preserved when any accessor operation of the object is executed.
Is a Circle a subtype of Ellipse?

• By the definition given, yes.
• If, on the other hand, the Ellipse class has a stretch operation, then a Circle is not an Ellipse since stretching the Circle makes it no longer a Circle.
• Thus, the definition of type conformance restricts itself to accessor operations only (called const functions in C++).

Design Principle

• In a sound object-oriented design, the type of each class should conform to the type of its superclass.
• In other words, the class/subclass inheritance hierarchy should follow the principle of type conformance.
Contravariance and Covariance

- Every operation of the superclass has a corresponding operation in the subclass with the same name and signature.
- Every operation’s precondition is no stronger than the corresponding operation in the superclass. (contravariance)
- Every operation’s postcondition is at least as strong as the corresponding operation in the superclass. (covariance)

Example

- Employee is a superclass of Manager.
- Employee.gradelevel > 0.
- Manager.gradelevel > 20.
- Manager’s invariant is stronger than Employee.
Precondition

<table>
<thead>
<tr>
<th>Emp.perfEval</th>
<th>Mgr.perfEval</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – 5</td>
<td>0 – 5</td>
<td>Equal</td>
</tr>
<tr>
<td>0 – 5</td>
<td>0 – 8</td>
<td>Weaker</td>
</tr>
<tr>
<td>0 – 5</td>
<td>-1 – 9</td>
<td>Weaker</td>
</tr>
<tr>
<td>0 – 5</td>
<td>1 – 5</td>
<td>Stronger</td>
</tr>
<tr>
<td>0 – 5</td>
<td>2 – 4</td>
<td>Stronger</td>
</tr>
</tbody>
</table>
The sender of calcBonus does not know whether the receiver is a “Employee” or a “Manager”. If Manager had a stronger precondition (2 - 4), then sending it a value of 1 would cause a problem. On the other hand, if Manager had a weaker precondition (0 -6), then a value of 5 would not cause a problem if fed to an employee.
Summary of Requirements

- **State Space**
  - The state-space of $S$ must have the same dimensions as $T$. (But $S$ may have more dimensions)
  - In the dimensions that $S$ and $T$ share, the state-space of $S$ must be either equal or lie within the state-space of $T$. 
Requirements (cont.)

- Operations
  - S.op must have the same name as T.op
  - The argument list of S.op must correspond to the argument list of T.op
  - The precondition of S.op must be equal to or weaker than T.op’s precondition.
  - The poscondition of S.op must be equal to or stronger than T.op’s postcondition.

Principal of Closed Behavior

In an inheritance hierarchy based on a type/subtype hierarchy, the execution of any operation on an object of class C – including any operation inherited from C’s superclass(es) – should obey C’s class invariant.
Case 1

Invoking move on a triangle preserves all of the properties of the triangle.

Case 2

Applying addVertex to a triangle destroys the “triangleness.”
- Avoid inheriting this operation.
- Override it so that it does nothing.
- Override it so that it re-classifies the object.
Run-Time Type Checking

• In designs where there is not closed behavior (see Case 2), the sender of a message must:
  – Check the run-time class of the target, or
  – Restrict polymorphism on the variable that points to the target, or
  – Design the message on the assumption that the target is the class with the greatest constraint on its behavior.

Factoring Subclasses

```
Polygon
    +--- FixedSidedPolygon
        +--- Triangle
        +--- Rectangle
    +--- VariableSidedPolygon
        +--- addVertex (newVertex : Point)
```
Abuses of Inheritance

- Mistaken aggregates
- Inverted hierarchy
- Confusing class and instance
- Misapplying *is a*

Mistaken aggregates

```
Airplane
  /   \
Wing  Tail
  /   /  \
Engine Fuselage
```
Mistaken aggregates (2)

How do we account for two wings?

Inverted hierarchy

The boss’s view.
Confusing class and instance

Corrected inheritance hierarchies
Expanded hierarchy

Misapplying *is a*

```
Cuboid
  volume ()
  stretch (...)
  rotate (...)

Room
```

aRoom.volume() makes sense.
What about aRoom.stretch(…)?
aRoom.rotate(…)?

How about cylindrical rooms?
Two approaches for cylindrical rooms

A Room has a Shape
The Danger of Polymorphism

- Polymorphism promotes conciseness in object-oriented programming by allowing an operation to be defined with the same name on more than one class and by allowing a variable to refer to an object of more than one class.
- Polymorphism thus enables the operating environment to choose automatically the correct operation and to execute as a result of a message, without the need for a complicated `case` statement.
- What happens if an object receives a message it doesn’t understand?

The Scope of Polymorphism

- The scope of polymorphism of an operation is the set of classes upon which the operation is defined.
- A scope of polymorphism forms a branch of the inheritance hierarchy.
The structure of a COP

A ragged SOP
Polymorphism of variables

- The scope of polymorphism of a variable $v$ is the set of classes to which objects pointed to by $v$ (during $v$’s entire lifetime) may belong.
- In C++ and similar languages this is restricted to descendants of a given class.
- In some other languages, such as Smalltalk, all variables are of type `Object` from which all classes are derived.
- In Java a variable may be of type `Object`, but it may also be of a specific class, in which case its value is restricted to descendents of that class.

Polymorphism in messages

- A message is composed of a variable that points to the target object and an operation name that states the operation to be invoked.
- Both the variable and the operation have a scope of polymorphism.
Two Cases

- Case 1
  - `factoryDevice` always points to an object of class `Tap`, `Motor`, or `Light`.
  - All of these know how to switch on.

- Case 2
  - `factoryDevice` points to any device in the factory. (Light, Door, …)
  - Not all of these know how to switch on.

Note: in some languages case 2 will result in a run-time error, in others it will be a compile time error.
Polymorphism and genericity

• A parameterized class (i.e. a C++ template) takes a class name as an argument whenever one of its objects is instantiated.
• In general, there is no restriction on the class that may be used as a parameter.
• In C++ there will be a compile-time error if the parameter class is assumed to have an operation that it does not.

Polymorphism and genericity(2)

template <class T> SortedTree {
public:
    Print(); // prints each element
    Insert(T k); // uses <
...
}

SortedTree<real>; // can both print and compare
SortedTree<Customer>; // can print, but < not defined
SortedTree<Animal>; // neither print nor compare
Polymorphism and genericity (3)

- Some languages, e.g. Ada, require explicit specification of the operations defined on generic parameters.
- The C++ standard library provides additional template arguments, with defaults, for required operations.