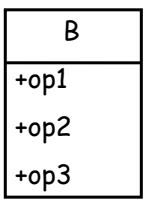
"Implementation" vs "behavioural" inheritance (mentioned in CSCU9P5)

- When one or more classes extend another class *solely for the purpose of sharing code*, this is called "implementation inheritance"
  - In the *model* being constructed there is no *conceptual relationship* between the entity modelled by the superclass and the entity being modelled by the subclass
  - Indeed, the superclass might not "*model*" anything!
  - For example, in a library system, the staff list, the catalogue and the loans list might all inherit from a list manager class – but there is no conceptual link between them

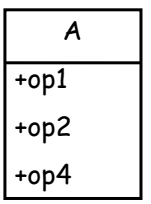
- When there is a conceptual "is-a" relationship between the sub/superclasses, this is called "behavioural inheritance" or "specification inheritance"
  - The classes are *conceptually* related *in the model*
  - For example, in a library system, the Book and Magazine classes might (and probably should) extend the Publication class
- Bruegge & Dutoit caution that the code sharing benefits of inheritance may be outweighed by the subsequent difficulties if it is not genuine behavioural inheritance
  - Example: next slide

- Example where *implementation inheritance* gives difficulties: (diagrammatically on the next slide)
  - Suppose that we choose to make class B inherit from class A simply because some of B's public operations are already provided by A
  - A probably also provides other public operations, inherited by B although not strictly required by it
  - A client of B, say class C, may then make use of *any* of A's public operations inherited by B
  - If we now find a better way to implement B, say by inheriting from D instead of A, then C may fail because D may not provide *all* the operations from A that C had come to depend on
  - We would *not* expect this problem to arise if B "is-an" A, because *all* of A's public operations would be *natural* and *required* parts of B. Any D replacing A would naturally have to fulfil all the same requirements

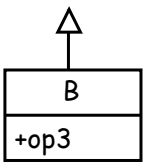
- Suppose we want:



and we have:



- Then we could implement B by inheritance from A:



- But then we *could* have a client C containing:

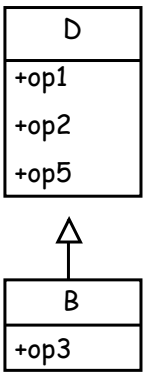
```
myB.op1();
myB.op2();
myB.op4();
```

← Naughty/danger: B is not supposed to offer op4!

- If A's implementation changes then C may become invalid! (See next slide)

- We may find a class D that *should* substitute for A, because it offers the same *intended* public operations of B: op1 and op2

  (Maybe D is cheaper, more reliable, faster)



- So, we adjust B to inherit from D:
- But now C is garbage:

```
myB.op1();
myB.op2();

myB.op4();
```

⟵ B no longer offers op4!

- Behavioural inheritance would require "B is-an A" and "B is-A D", and "D is-an A" for valid substitution – which it clearly is NOT above