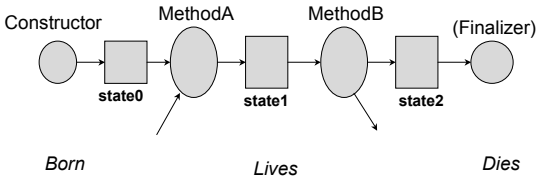
Programming by Contract 2

Object-oriented languages
Class Invariants

Object Lifecycle

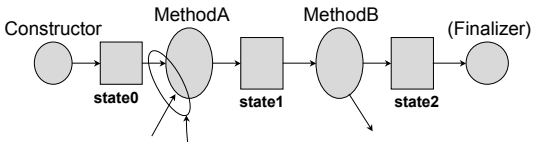


Client example

- From a client's viewpoint:

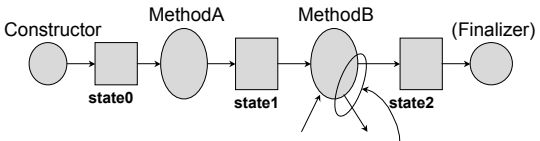
```
{
   Stack s1 = new Stack;
   s1.push(new Integer(4));
   System.out.println(s1.pop());
   // can be collected now
}
```

Preconditions of methods



Precondition depends upon current object state, plus any inputs to the method.

Postconditions of methods



- Postcondition talks about the new state of the object and the method's outputs.
- May need to refer to initial states and inputs for comparison.

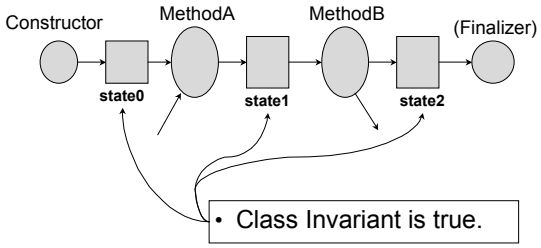
Push Example

```
public class Stack extends Vector {
    …
    public void push(Object val) {
      assert(dataArray != null);
      int origsize = size(); //save it.
      assert(origSize >= 0);

      // Push code . . .

      assert( ! empty() );
      assert( top() == val );
      assert( size() == origsize+1 );
    }
}
```

Class Invariants

- Pre/Postconditions only describe the properties of individual methods.
- We also want to document properties of the whole class. Things which are common to its methods.
- A ***class invariant*** expresses consistency constraints that apply to all objects of that class, all through their lifecycle.

When is class invariant true?



Class Invariant Rules

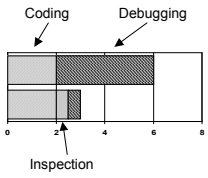
- Constructors must *establish* C.Inv.
- Methods must *maintain* C.Inv.
  - However, within the method, the class invariant may be temporarily broken, while data structures are being updated.
- Finalizer can *assume* C.Inv.
- Subclasses should only *strengthen* C.Inv.

Checking Class Invariants

- Define an *invariant* method in each class.
  - public/protected void invariant();
- Call it at the end of each constructor, and at the end of each method that modifies the object.
  (Challenge: disable these calls in release code?).
- Advantages:
  - Clear documentation of class data structures.
  - Catches corrupt data errors ASAP.
  - Subclasses can refine invariant() adding their own additional checks (*strengthen*).

Invariant() Example

```
public class Stack extends Vector {
    …
    protected void invariant()
    {  super.invariant();
        assert(0 <= size());
        assert(empty() == (size()==0));
        if (!empty())
            assert(top() = elementAt(size()-1);
    }
}
```

Program Inspection

- How many bugs in your new program?
  - Standard ratio, say 100/1000 LOC?
  - Debugging is a *VERY* slow way of removing them (and only removes *some* obvious ones).
- Inspection: (Before debugging!)
  - Read through code carefully.
  - Think deeply about each line.
    - "What could go wrong here?"
    - "How do I know this is right?"