Objects and Design in Smalltalk

3. Abstract Classes

Éric Tanter
etanter@dcc.uchile.cl





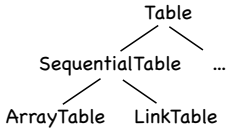
Abstract class

- Class that never has any instance!
  - why is this interesting?
  - actually they are even indispensable in good object-oriented design
- Act as a center for gathering behavior and expectations common to its subclasses
  - e.g. Magnitude gathers comparability
  - a cognitive distinction in the everyday world

Class Object

- The most abstract class in Smalltalk
  - instances not really useful
  - gathers expectations of all Smalltalk objects
    - copy, display, test for equality, etc.
- How can copying or displaying be defined once for _all_ subclasses??
  - it cannot!
  - that's why: abstract = repository of expectations
  - actual code is likely to be in subclasses

Exercise

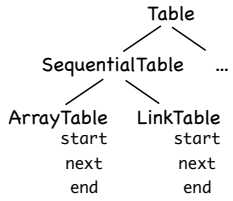


- Searching tables

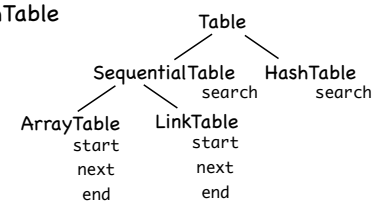
```
search for an <item>
  start
  loop while (not end and next != <item>)
        end loop
  if end then return not_found
      else return found
```

- in which classes should search, start, end, and next be coded?

- next: depends on the kind of table
  - ArrayTable: `index := index + 1`
  - LinkTable: `current := next`
  - each subclass has its version of next
- start: similar
  - ArrayTable: `index := 1`
  - LinkTable: `current := head`
- end: similar
  - ArrayTable: `index > size`
  - LinkTable: `current = nil`

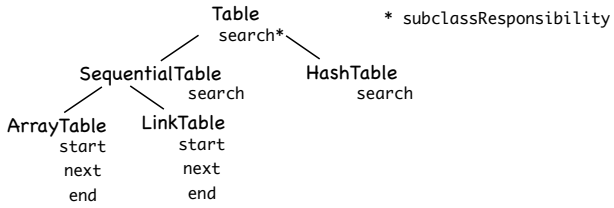


- search: same code!
  - write it in SequentialTable
- why not write it in Table?
  - consider a HashTable

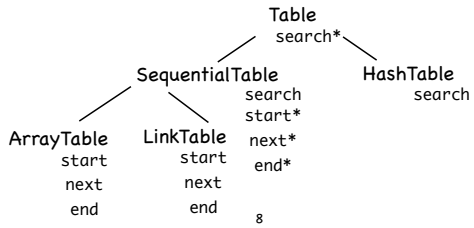


Abstract Methods
(subclass responsibility)

- Table has no behavior at all
  - but every table should be searchable! (expectation)
- put an abstract search method in Table that does nothing
  - announces searchability
    - to consumers of Table's subclasses
    - to programmers as well!
  - contract between consumers and developers
    - Smalltalk: informal contract
    - C++/Java: enforced contract (by the compiler)



- What other abstract class?
- Opportunity for abstract methods?



Ensuring no instances

- Can we prevent someone to do `Table new`?
  - Smalltalk cannot, C++/Java can (compiler)
  - no method `search` in Table: `doesNotUnderstand`
  - better:

```
search: anItem
    self subclassResponsibility
```

  - gives documentation!

Concrete Behavior in Abstract Classes

- Not all methods of an abstract class need be abstract!
- A concrete method invokes some abstract ones
  - overriden in (concrete) subclasses
  - e.g. `search` calling `start`, `next`, `end`

TYPICAL CHARACTERISTIC OF SOUND OBJECT-ORIENTED DESIGN!

Another Example

Magnitude

```
max: anotherMag
   self < anotherMag
       ifTrue: [ ^anotherMag ]
       ifFalse: [ ^self ]
```

LOOK FOR THIS IN YOUR DESIGNS!

- < is implemented in subclasses (Date, Time, ..)
  - `13.7 max: 17.3`
  - `max:` is found in Magnitude
  - < is called on 13.7 (late binding of self!)
- Maximizes reuse (one `max:`, many <)
- Subclass implementors focus on specific behavior (<) and enjoy `max:` for free!

3 Forms of Methods in Abstract Classes

- abtract
  - aka. pure virtual, subclassResponsability, etc.
- concrete
  - self-contained
- hybrid
  - like SequentialTable>>search, Magnitude>>max:
  - subclass developers need to provide some concrete behavior, but do not override the whole method
  - aka. template, base, yo-yo, down-call, etc.