Lessons from the *glob* Homework

CS 420/520

Andrew P. Black

Portland State UNIVERSITY

Test first development

- A student wrote:

> I enjoyed the immediate feedback which directed me towards the possible problem area in my design immensely, but at the same time this made me more skeptical about design with every failing test case. I guess at times, I felt like I was implementing patches to a design that I failed. I don't know if this is due to the fact that I indeed failed with the design or because it was my first time dealing with this many failing tests and I subconsciously felt bad about it.

- Solution:
  - add one test case at a time
  - helps you focus on a single issue
  - if you need to change your design, do it one step at a time

Use More Objects

- A student writes:

> I was pretty happy with my code before I posted a question to the class forum about the behavior of extra symbols inside brackets. At that point I had only one "bracketParseState", rather than the "leftBracketParseState" and "bracketCharsParseState" I ended up with. I chose to raise an error any time one of the other symbols appeared inside brackets, if a left bracket appeared without a right bracket, or if a right bracket appeared before a left bracket.
>
> After reading the discussion on the forum, I switched to the two state implementation, where one is used when a left bracket first appears, and the second one is used to fill the brackets with characters. Now the combinations of symbols described above are all treated as plain characters rather than raising errors. I'm happier with this version of the code.

Tests/Specs should *Communicate*

testAddWithOccurrences

```
| sizeOfBag1Ref  sizeOfBag1Act occuranceOfCharRef occuranceOfCharAct errMsg testName |
 testName:='testAddWithOccurrences'.

"---Populate Bag1---"
 bag1 add: #Q withOccurrences: 15.
 occuranceOfCharAct := bag1 occurrencesOf: #Q.
 occuranceOfCharRef := 15.
 sizeOfBag1Act := bag1 size.
 sizeOfBag1Ref := 15.

"---Do asserts on Bag1---"
 errMsg:=testName, '-- Bag1 size failed;  REF:', sizeOfBag1Ref asString,  ' ACT:',
sizeOfBag1Act asString.
 self assert: (sizeOfBag1Ref = sizeOfBag1Act)  description: (errMsg).

 errMsg:=testName, '-- Bag1 numQs failed; REF:', occuranceOfCharRef asString, ' ACT:',
occuranceOfCharAct asString.
 self assert: (occuranceOfCharRef = occuranceOfCharAct) description: (errMsg).
```

testAddWithOccurrences

```
bag1 add: #Q withOccurrences: 15.
self assert: bag1 size equals: 15 .
self assert: (bag1 occurrencesOf: #Q) equals: 15.
```

States exist to simplify tokenization

- Ideal tokenization loop:

```
state := initialState
patternChars.do { ch →
    state.consume(ch)
}
state.finalize
```

Avoid “encodings”

- Compare

```
patternChars.do { ch →
    state.consume(ch)
}
state.finalize
```

to

```
patternChars.do { ch →
    state.consume(ch)
}
state.consume ""  // use "" to mean "end of input"
```

- The Once and Only Once rule:
  - everything that the program has to say *should be said*, and
  - it should be said *just once*
- There are *two situations*, thus *two things* to say: (1) consuming a character, and (2) the end of the input.
  - give each its own method

```
def plainCharsState = object {
  method consume(ch) { ... }
  method finalize { ... }
}
```

Shopping *vs.* Building

- Constructing an Object-oriented application is a process of shopping for the components that one needs
  - occasionally, we add a new item to the shelf
  - often, we can find a component that almost fits
- The *openness* of an OO language allows the programmer to change the component that *almost* fits into one that is a *good* fit.
  - works only if we have a rich set of components on the shelf, and if they are open to change.

Know thy libraries

- A mature OO-ecosystem like Java's is likely to have a component that you can modify
- Even in Grace, you should take advantage of the available components

e.g., Grace strings have

```
substringFrom(start) to(stop) → String
```

but also

```
substringFrom(start) size(max) → String
```

and

```
substringFrom(start) → String
```

- Grace has Dictionaries: no need to simulate them with 2 parallel sequences:

```
method transition(ch){
    def edgeNames = ["*", "?", "[", ""]
    def states = [ starState, wildcardState,
       emptySquareState "[",
       accumulateState “" ]
    if (edgeNames.contains(ch) && accum.isEmpty.not) then {
       addToken(plainToken(accum))
    }
    def stateIndex = edgeNames.indexOf(ch) ifAbsent {
       state := accumulateState(accum ++ ch)
       return
    }
    state := states.at(stateIndex)
}
```

- Code is hard to read, *and* inefficient

- Instead:

```
method transition(ch) {
    def transitions = dictionary [
          "*"::starState,
          "?"::wildcardState,
          "["::emptySquareState "[",
          ""::accumulateState "" ]
    if (transitions.containsKey(ch) &&
          accum.isEmpty.not) then {
        addToken(plainToken(accum))
    }
    state := transitions.at(ch)
          ifAbsent {accumulateState(accum ++ ch)}
}
```

Hoist Constants

- transitions never changes:

```
method transition(ch) {
    def transitions = dictionary [
          "*"::starState,
          "?"::wildcardState,
          "["::emptySquareState "[",
          ""::accumulateState "" ]
    if (transitions.containsKey(ch) &&
          accum.isEmpty.not) then {
        addToken(plainToken(accum))
    }
    state := transitions.at(ch)
          ifAbsent {accumulateState(accum ++ ch)}
}
```

Hoist Constants

- `transitions` never changes:

```
def transitions = dictionary [
            "*"::starState,
            "?"::wildcardState,
            "["::emptySquareState "[",
            ""::accumulateState "" ]
method transition(ch) {
   if (transitions.containsKey(ch) &&
            accum.isEmpty.not) then {
       addToken(plainToken(accum))
   }
   state := transitions.at(ch)
         ifAbsent {accumulateState(accum ++ ch)}
}
```

Ockham's Razor

- *pluralitas non est ponenda sine necessitate*, or "plurality should not be posited without necessity"
- In our context: don't use a `class` when all you need is an `object`