File scanning loops & loop invariants

A Simple Java Indenter

Indenting Java code

Poorly indented code hides bugs

Auto-indenting makes it easier to see missing and extra braces

Let’s build our own!

What I want ...

for (int i = 0; i <= s.length(); i++)
    char ch = s.charAt(i);
    if (Character.isLowerCase(ch))
        hasLower = true;
    else if (Character.isUpperCase(ch))
        hasUpper = true;
    else if (Character.isDigit(ch))
        hasDigit = true;
    else if (isPunct(ch))
        hasPunct = true;
    else
        return false;

Strategy?

Even before pseudocode ...

What general strategy can we use?

We want to ignore the original indentation, and apply indentation according to program structure. It’s not hard!

Throw away old indentation ... replace with the right amount for each line, using spaces only
Tools you may need ...

(from Zinn and the Art of Road Bike Maintenance)

Better tools for this job ...

String s;
s.charAt(i) the character at line i (from 0)
s.length() how many characters in s
s.trim() copy of s without leading spaces

plus File manipulation (open file, read from file, write to file)
and “noggin” (brain).

Finding tools ...

Finding tools ... API documentation
What’s in my toolbox?

Lots ... our tool might be in there somewhere

Let’s check this one out

Tools gathered ... we need a strategy to use them

Full tool description

charAt

```java
public char charAt(int index)
```

Returns the char value at the specified index. An index ranges from 0 to `length - 1`. The first char value of the sequence is at index 0, the next at index 1, and so on, as for array indexing.

If the char value specified by the index is a surrogate, the surrogate value is returned.

**Specified by:**
- `charAt` in interface `CharSequence`

**Parameters:**
- `index`: the index of the char value.

**Returns:**
- The char value at the specified index of this string. The first char value is at index 0.

**Throws:**
- `IndexOutOfBoundsException`: if the index is negative or not less than the length of this string.

In some cases we’ll see multiple methods with the same name and different arguments, “overloaded”, and choose the one we want. These were generated from JavaDoc comments in the library source code.

Strategy

Read line by line, printing modified line
Discard spaces at beginning of each line
Keep track of counts of ‘{ ’ and ‘}’

Or just keep track of their difference;
call that “block level”
Insert enough spaces for current block level
Printing with indentation

*Code will look like this:*

```java
for (int tab=0; tab < blockLevel; ++tab) {
    System.out.print(" ");
} System.out.println( line );
```

Controlling indentation

```java
while ( ... ) {
    ++blockLevel
    body
    ++blockLevel
    if ( ... ) {
        ... ???
    } else { --blockLevel
        ... --blockLevel
    } --blockLevel
}
```

What’s the right order?

Adjust block level before or after printing line?

Pseudocode?

Assume we’ve already got a scanner ... what’s the main loop look like?
**For each line** ...

Trim leading blanks
Print leading spaces
Print line
Adjust block level for next line

(this isn’t quite right)

**Better** ...

For each line
adjust block level according to first character
print line indented for current block level
adjust block level according to the rest

This will handle } else { correctly

**Loop invariants**

“Invariant” : Something that doesn’t change
Loop invariant:

When I here in the loop,
the invariant is true every time

**Reasoning with invariants**

initialize before loop;

// Invariant must be true here
while ( ... ) {
   // Invariant must also be true here
   ...
}

Invariants, basis, progress

... // Establish the invariant
while ( ... ) {
  if (basis case) {
    ...
  } else { // progress case
    ... // maintain the invariant
  }
}

Invariant for indentation

Block level = 0; // Invariant established
For each line {
  adjust block level according to first character
  print line indented for current block level
  adjust block level according to the rest
  // Invariant maintained: Block level is the
  // difference between counts of ‘{‘ and ‘}’
}

Invariants, basis, progress (2)

... // Establish the invariant
while ( not basis case ) {
  ... // progress case:
  ... // maintain the invariant while
  ... // reducing problem size
} // basis case

Let’s write it.