Inside-out rule; use of this, super

Developing methods (using Strings).

Read sec. 2.5, stepwise refinement

Listen to Plive, 2.5.1–2.5.4.

Reading for next lecture: the same

Office hours are being held

Reread on spelling:

According to a research at Cambridge University, it doesn’t matter in what order the letters in a word are, the only important thing is that the first and last letter be at the right place. The rest can be a total mess and you can still read it without problem. This is because the human mind does not read every letter by itself, but the word as a whole.

Office hours are being held

A1

A3: Adding functionality to A1

• Keeping class invariant true
• Use already-written functions
• Boolean expressions
• Use of null and testing for it
• Use of static variables

Due Wednesday, 29 September

Remember frame boxes and figuring out variable references?

The inside-out rule (see p. 83)

Code in a construct can reference any of the names declared or defined in that construct, as well as names that appear in enclosing constructs. (If a name is declared twice, the closer one prevails.)

Parameter n would be found in the frame for the method call.

Parameter name “blocks” the reference to the field name.

A solution: this and super

Within an object, this evaluates to the name of the object.

Within a subclass object, super refers to the partition above the one that contains super.

Because of the keyword super, this calls toString in the Object partition.
Strings are (important) objects that come with useful methods.

```java
String s = "abc d";
```

Note the “index (number) from 0” scheme:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

DO NOT USE == TO TEST STRING EQUALITY!

- `s == t` tests whether `s` and `t` contain the name of the same object, not whether the objects contain the same string.
- Use `s.equals(t)`

Text pp. 175–181 discusses Strings

Look at API specs for String

- `s.length()` is 5 (number of chars)
- `s.charAt(2)` is 'c' (char at index 2)
- `s.substring(2,4)` is "c " (NOT "c d")
- `s.substring(2)` is "c d"
- "bcd   " .trim() is "bcd" (trim beginning and ending blanks)
- `s.indexOf(s1)` – index or position of first occurrence of `s1` in `s` (-1 if none)

Principles and strategies embodied in stepwise refinement

Develop algorithm step by step, using principles and strategies embodied in “stepwise refinement” or “top-down programming.”

READ Sec. 2.5 and Plive p. 2.5.

- Take small steps. Do a little at a time
- Refine. Replace an English statement (what to do) by a sequence of statements to do it (how to do it).
- Refine. Introduce a local variable — but only with a reason
- Compile often
- Intersperse programming and testing
- Write a method specification — before writing its body
- Separate concerns: focus on one issue at a time
- Mañana principle: next slide

Anglicizing an Integer

```java
/** = the anglicization of n.
   * Precondition: 0 < n < 1,000,000 */
   * @param n an integer
   * @return the anglicization of n.
   */
   public static String anglicize(int n) {
       return "one hundred twenty three";
   }
   ```

We develop this function, in DrJava, using the principles and strategies of stepwise refinement (also called top-down programming).