Stepwise refinement: Switching time formats

To the right are definitions of time in a 24-hour format and an AM-PM format. The AM-PM format is illogical, but that's what we are stuck with in the U.S. For example, midnight should be 00:00AM but instead it is 12:00AM. Similarly, two minutes after noon should be 00:02PM but instead is 12:02PM.

24-hour format: "<hrs>:<mins>"
<hrs> is in 0..23, <mins> is in 0..59
Examples: "0:0" is midnight
"12:00" is noon
"2:21" "12:1" "3:5" "23:59"

Also note that the hours and minutes in 24-hour format can be 1 or 2 digits, but in the AM-PM format, they must be two-digits, with a leading 0 if necessary.

AM-PM format:
"<hrs>:<mins>AM" or "<hrs>:<mins>PM"
<hrs> is in 1..12, <mins> is in 0..59
Both <hrs> and <mins> are 2 digits.
Examples: "12:00AM" is midnight
"12:00PM" is noon
"02:21AM" "12:01PM"
"03:05AM" "11:59PM"

Our task is to write function *toAMPM*:

```
/** Return time t in AM-PM format.
  * Precondition: t is in 24-hour format. */
public static String toAMPM(String t)
```

How will this work? We have to extract the hours and minutes from parameter *t*, figure out how to change the hours to AM-PM format and construct a string that contains either "AM" or "PM", and finally build the resulting string and return it. We write this carefully as the following statement-comments.

```
// 1. Store in h and m the hours and minutes of String t

// 2. Change h and store either "AM" or "PM" in variable ampm for the time in AM-PM format.

// 3. Construct the result string from h, m, and ampm and return it.
```

Well, that was an example of stepwise refinement: We divided the task into three subtasks, each well defined.

We can implement these three tasks in any order. But we usually start with the first because that allows us to begin testing sooner. When step 1 is completed, we can use println statements to make sure we implemented it correctly. We won't do that here, but we actually did it when developing the method.

In the AM-PM format, the hour and the minute can be 1 or 2 characters. So we write the following:

```
// 1. Store in h and m the hours and minutes of String t
int k= t.indexOf(":");
int h= Integer.parseInt(t.substring(0, k));
int m= Integer.parseInt(t.substring(k+1));
```

We now work on step 2. Because of the illogical nature of the U.S. AM-PM format, before writing it in Java, we describe in more detail what should be placed in variables *h* and *ampm*. Whenever a more complex problem arises, go more slowly and make less drastic refinements.

There are four cases to consider:

```
// 2. Change h and construct a String ampm for time in the AM-PM format.
/* case h = 0: set h to 12 and ampm to "AM"
 * case 0 < h < 12: set ampm to "AM"
 * case h = 12: set ampm to "PM"
 * case 12 < h: subtract 12 from h and set ampm to "PM" */
```

How can we best implement this? Having a four-case analysis seems messy. Let's see. Perhaps it's best to handle assignments to *h* and *ampm* separately. Ah, it's easy to implement the assignment to *ampm*:

```
String ampm= h < 12 ? "AM" : "PM";
```

What about *h*? If *h* is 0, make it 12, and if it is greater than 12, subtract 12:

```
if (h == 0) h= 12;
else if (12 < h) h= h – 12;
```

That's not bad!

```
String = h < 12 ? "AM" : "PM";
if (h == 0) h= 12;
else if (12 < h) h= h – 12;
```

Finally, we work on step 3:

// 3. Construct the result string from *h*, *m*, and *ampm* and return it.

That should be fairly easy:

return *h* + ":" + *m* + *ampm*;

But look carefully at the definition of AM-PM format. Both *h* and *m* must be written as two characters, with a leading 0 prepended if necessary. So we may have to add a leading 0 in two places. That suggests using the Mañana Principle; we formulate a helper method named *char2*, complete the code, then later complete the helper method.

/** **Return** n with at least 2 chars, prepending a 0 if necessary. */
public static String *char2*(**int** *n*)

Then we write step 3 as:

// 3. Construct the result string from *h*, *m*, and *ampm* and return it.
return *char2*(*h*) + ":" + *char2*(*m*) + *ampm*;

Finally, function *char2* is easy to implement.

We leave the first two statement-comments in the final program, as well as the case analysis for figuring out how to change *h* and set *ampm*. This will help any reader of the program. We decide not to include the statement-comment for step 3, since the code is self-explanatory.

That ends the development of the method using stepwise refinement. Three important refinements were

(1) the initial breaking up of the task into a sequence of three tasks.

(2) writing the case analysis before writing Java code, and

(3) Using the Manana Principle.

```
/** Return time t in AM-PM format.
  * Precondition: t is in 24-hour format. */
public static String toAMPM(String t) {
   // 1. Store in h and m the hours and minutes of String t
   int k= t.indexOf(":");
   int h= Integer.parseInt(t.substring(0, k));
   int m= Integer.parseInt(t.substring(k+1));

   // 2. Change h and construct a String ampm for time in the AM-PM format.
   /* case h = 0: set h to 12 and ampm to "AM"
    * case 0 < h < 12: set ampm to "AM"
    * case h = 12: set ampm to "PM"
    * case 12 < h: subtract 12 from h and set ampm to "PM" */
   String= h < 12 ? "AM" : "PM";
   if (h == 0) h= 12;
   else if (12 < h) h= h – 12;

   return char2(h) + ":" + char2(m) + ampm;
}

/** Return n with at least 2 chars, prepending a 0 if necessary. */
public static String char2(int n) {
   if (0 <= n && n < 10) return "0" + n;
   return "" + n;
}
```