06 Tracing Code by Hand

[Close all sections] Only visible sections will be included when printing this document.

▽ Reading and writing are complementary skills

In any language, writing and reading are strongly related skills. *After* writing down a sentence, or a paragraph, it should be possible for the author to read it and confirm that it expresses the ideas they wanted to convey. But really they are also doing this *before* they write down anything. When composing a sentence in your head you already have some idea that it will make sense to another reader once you write it down.

The same is true in programming. As your knowledge of the language—Java in this case—grows you will have a larger vocabulary for expressing algorithmic ideas to the computer *and* you will also know what effect each statement has. This is a skill you can develop on code that you didn't originally write: reading pieces of code and predicting their effect (and then validating this by running the code) is an essential step to being able to write good code. It enables you to reason about the behaviour of a program and, when a program doesn't work as expected (which is inevitable in the long run), you can make a change with some certainty that it will correct the mistake.

▽ Tracing tables

Part 4 of these notes introduced a technique for reading code known as a tracing table (but without explicitly calling it that). A tracing table is a way of keeping track of the value of each variable at different points in a program's execution. As you step through each line of a program you can read it, decide what actions occur (for example, is it an assignment statement, or was some output produced?) and then update the entries in the table to record what changed. Here's a short example:

```
1  int a = 5;
2  int b = 20;
3  a = b * 2;
```

which can be described by either of the following two tracing tables:

Line	a	b
1	5	
2	5	20
3	40	20

Line	a	b
1	5	
2		20
3	40	

In the example of the left we've written down the value of all variables at each line, even if they didn't change, while in the example of the right we've only noted down changes to a variable's value. Both are valid approaches: the one of the left requires more writing but makes it easier to see the most recent (current) value of a variable, while the one on the right saves some writing but if a variable's value is used (read) at a later point in the program then you have to read up the column to find its current value.

Essential programming semantics of assignment, expressions and method calls

As you work through each line of code keep in mind the following about the order in which instructions, particularly assignments, are evaluated.

1. When processing an assignment statement, the expression on the right hand side is evaluated in full before its value is copied into the variable on the left. So, given:

   ```
   int a = 5;
   ```

   the assignment statement

   ```
   a = a + 2;
   ```

   evaluates the expression `a + 2` using `a`'s *current* value of 5 and *then* assigns the value 7 to `a`.

2. Any arithmetic expression is evaluated according to the standard rules of 'BODMAS' (bracketed subexpressions, orders [powers], division and multiplication, addition and subtraction), from left to right.

3. When a method call is reached, its arguments are evaluated in full first, then those values are passed to the method, which executes its statements before the code in the caller resumes. So the code:

   ```
   int m;
   m = Math.max(2 + 2, 5) + 10;
   ```

   is processed in this order:
   1. `2 + 2` is evaluated to 4.
   2. `Math.max(4, 5)` is called and the code within it is executed, in sequence. Eventually it returns a value (which in this case will be 5).
   3. Once `Math.max` returns the addition expression 5 + 10 is evaluated to get a single value on the right hand side: 15
   4. The value 15 is copied into the variable `m`.

Fun Fact: Formal computer science uses the term 'left value' to refer to a variable on the left hand side of an assignment statement and 'right value' to refer to a variable, value literal or expression that appears on the right. That may seem silly and redundant, but the distinction is that a left value represents a memory location in which a value can be stored (*written*), while a right value represents a value only (any variable on the right hand side is *read*). That's why an assignment statement always has a single variable on the left, but may be arbitrarily complex on the right.

▽ Guidelines for constructing a tracing table

Apply the following guidelines to construct a tracing table:

- Create a column for the line number, followed by one column for each variable declared. If the program also produces some visible output then have another column for *Output*.
- Create the table one row at a time as you read through the code.
- Create a row for each line where a variable's value changes or the program produces some visible output.
- Leave blank cells in the table for variables that have not yet been assigned a value.
- Include *only* those lines that are executed and that change the value of a variable or produce output. This becomes important when we add the ability for a program to decide whether or not to carry out instructions or to repeat instructions.
- *If a variable's value changes* then either record the value of all variables immediately after the relevant line has been executed or just the one that changed.
- *If the current statement produces visible output* then record it in the *Output* column.
- Repeat line numbers in the table if those lines are executed multiple times. (Part 9 of these notes discusses loops.).
- You may write String values without quotes (unless their type is ambiguous, such as the String `"5"`, which could be misinterpreted as an integer).
- You do not need to draw grid lines. The examples in these notes have grid lines because that's the format for tables everywhere in these notes.

Numbered or unnumbered

A tracing table can take two forms, depending on where you are writing it down:

1. If you have a copy of the code printed on paper, and there's enough space next to it, then instead of having a line number column you can simply write down the values of the variables next to the actual lines of code that modify them. For example:

   ```
                 a     b
   int a = 10;   10
   int b = 20;         20
   a = b + a;    30
   ```

2. If you either don't have a paper copy of the code (and don't want to transcribe it) or there's not enough room next to the code, then use the approach described above and illustrated in the rest of these notes, with a column for the line number. This is also good when tracing code that involves loops (covered in a later part of the notes) since it's easy to show that certain lines of code are executed more than once.

A worked example

Let's trace the execution of the following program fragment:

```
1  int a;
2  char z;
3  String s = "start";
4
5  a = 0;
6  z = s.charAt(a);
7  a = (a + 2) * 10;
8  System.out.println("The first character of " + s + " is " + z);
9  System.out.println("And a is " + a);
```

At line:

1. The integer variable `a` is declared, but has no value yet. (Add a column for `a`)
2. The character variable `z` is declared, but has no value yet. (Add a column for `z`)
3. The String variable `s` is declared and refers to a new String object "start". (Add a column for `s` and write its value in, against line 3)
4. *is blank* (so don't add anything to the table)
5. The value 0 is assigned to the variable `a` (note this down in the table)
6. The method `s.charAt()` is called with the value of `a` (0). Any code in `charAt` is executed and then a single `char` value (`'s'`) is returned. That value is stored in the variable `z` (note this down in the table)
7. The arithmetic expression is evaluated as expected, `(a + 2)` first, so (0 + 2), then 2 * 10, to get the value 20, which is then stored in the variable `a` (not this down in the table)
8. The expression `"The first character of " + s + " is " + z` is evaluated left to right, which concatenates (joins) each component to yield the new String `"The first character of start is s"`. This value is passed to the `println` method, which produces some visible output (note this in the table, adding a column if you haven't already)
9. The expression `"And a is " + a` is evaluated to the String value `"And a is 20"`, which is then displayed (note this output in the table).

Which would lead to us to construct the tracing table:

Line	a	z	s	Output
3			"start"	
5	0			
6		's'		
7	20			
8				The first character of start is s
9				And a is 20

▽ ☑ Test Yourself: Tracing Simple (parts of) Programs

Activity: Work through some of the sample problems in *Appendix: Code Tracing Problems*. It contains a number of sample problems with solutions, covering a range of code structures, including some not yet described in this series of notes, so only attempt those samples which include language constructs with which you are familiar.

Activity: You may also wish to trace the execution of the solution to the *Star Wars* First Name Generator activity in the previous topic of notes. At the points where the program asks for user input you can use whatever values you would type in to the program. Are their inputs that might not work?

▽ Situations where tracing tables are less effective

There are some situations where using a tracing table is either too tedious to be useful or risks hiding important information. We won't ask you to create one for any of these situations, but so you're aware of them they include:

1. When working with a very large program. Tracing tables are good for small sections of code and, if a large program has been designed properly, you'll only ever need to use them on short pieces of code (for instance, when tracking down the cause of a fault).
2. When the code involves variables that are complex objects. In these situations you may wish to write down the values of the properties of the object outside the table, with a border to represent the 'object', and then show an arrow from the variable's cell in the table to the 'object' you've drawn.
3. When the code involves object assignment. Assigning one object reference to another object reference variable only copies the reference (so you have two variables that both refer to the same object in memory). This can be difficult to depict in a tracing table, but can be done by drawing some representation of the object (a box or circle) next to the table (as in the previous situation) and having the 'value' in those cells be an arrow pointing to that.