7-0: Box and Arrow Tracing

- Tracing by hand is a useful way to understand what's going on in a program.
- **box and arrow** tracing is a high-level way to understand the execution of a program.
- Boxes represent memory cells.
- Arrows represent pointers to memory cells.

7-1: Example

- int x = 4;
- String y = "hello world";
- Student z = new Student();
- z.setName("bob");

7-2: Example

- x is a primitive
  - Data is stored in a box
- y and z are object references.
  - y and z are pointers to memory locations.

7-3: Tracing practice

- Perform box and arrow tracing of the following program.
  - Create a multi-box for every object created with new (one cell for every data member). Point an arrow from the object reference cell to the multi-box representing the object.
  - When a method is called, create boxes for each parameter and local variable. Include a box for "this". Be sure and put the actual parameter values into the new boxes representing the formal parameters.

7-4: More detailed tracing

- Box-and-arrow tracing is nice, but too high-level sometimes.
- Doesn't let us keep track of how memory is allocated.
- We will also do more detailed tracing of programs to see what's actually happening.
7-5: **Run-time environment**

- The run-time environment refers to the way in which memory is used/arranged.
- Memory is arranged as a sequence of *addresses*.
- Each address refers to a word in memory.
- We can break the runtime environment into four sections:
  - Program code: Where the program itself resides.
  - Global data area: Global and static data is stored here.
  - Run-time stack: This contains an *activation record* for each method that is called.
  - Heap: Dynamically-allocated data (with new or malloc) is stored here.

7-6: **Activation Records**

- An activation record sets a context for a method’s execution.
- It contains:
  - Space for all parameters, including ‘this’, a pointer to the object itself.
  - Space for a return value.
  - Space for local variables.
- Each time a method is called, its activation record is pushed onto the stack.
- When the method exits, its activation record is removed.

7-7: **Symbol table**

- The *symbol table* is responsible for mapping variable names to addresses.
- This is how the Java interpreter knows the value that is currently associated with a variable.

7-8: **Activation Records**

- Simplifying assumptions:
  - Code section begins at address 0
  - Global data at 1000
  - Runtime stack starts at 2000
  - Heap starts at 5000
  - Integers get 4 bytes
  - Chars get 2 bytes
  - floats get 8 bytes

7-9: **Activation Records**

- In class: trace JavaSimple.java.
- On your own: try Baseball example.