

# Intro to Programming II

## Tracing

Chris Brooks

Department of Computer Science  
University of San Francisco

Department of Computer Science — University of San Francisco — p. 177

### 9-2: Memory Layout

- Understanding how memory is arranged and laid out is essential to becoming a good programmer.
- Helps you understand:
  - Runtime errors
  - Equality tests
  - Constructors
  - Differences between primitives and references
- Two models:
  - Box-and-arrow tracing
  - Runtime tracing

Department of Computer Science — University of San Francisco — p. 277

### 9-3: 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.

Department of Computer Science — University of San Francisco — p. 277

### 9-4: Box and Arrow Tracing

- Primitives are represented by boxes
  - Data is stored directly.
- Object references are represented by boxes and arrows.
- Multiboxes are used to represent multiple 'slots' within an object.

Department of Computer Science — University of San Francisco — p. 477

### 9-5: Example

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

Department of Computer Science — University of San Francisco — p. 577

### 9-6: 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.

Department of Computer Science — University of San Francisco — p. 577

### 9-7: 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.

### 9-8: 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, sizes, or where things are stored.
- We will also do more detailed tracing of programs to see what's actually happening.

### 9-9: 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.

### 9-10: 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.

### 9-11: 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.

### 9-12: 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

### 9-13: Activation Records

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