TRACING CODE: THE MEMORY MODEL

Computer Memory

There are two areas of computer memory for a running program:

Run-Time Stack: (a.k.a. *call stack*):

- information that is local to methods, like parameters, local variables, and which line of code is being executed.
- When a method terminates, all this information is erased.

Heap:

- longer-lived information.
- objects and their contents (anything created with `new`)
- static information.

The memory model traces how the computer uses these two areas while running a program.*

* *The memory model rules deal only with running code.* They do not describe what happens at compile time, such as figuring out whether a private variable can be accessed.

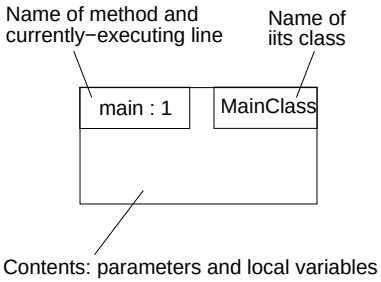
Stack: Method Space (the run-time stack) One box

("method frame") for each running method. Each frame contains that method's parameters and local variables.

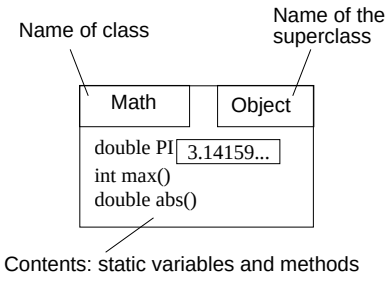
Heap: Static Space One box for each class, containing static variables and static methods. (These methods are not running. They are drawn to show that they are available.)

Heap: Object Space One box for each object, containing instance variables and instance methods. (These methods are not running. They are drawn to show that they are available.)

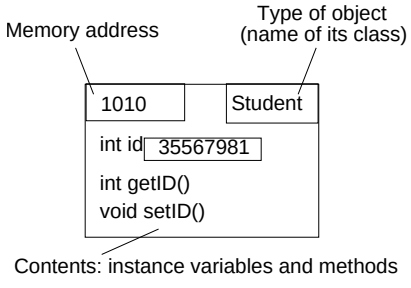
Method frame:



Static box:



Instance:



Tracing Program Execution

1. **Load the classes.**

   See the rules on loading a class.

2. **Call method main.**

   Begin execution by tracing a call to method main.

3. **Trace each statement line by line.**

   Follow the rules on the next several slides.

Statement Syntax

These are the types of statements we have to trace in step 3.

Statement type	Syntax
void method call	`expression.methodname(args);` (args is a comma-separated list of expressions) **Example:** `s.length();`
declaration	`type identifier;` **Example:** `String s;`
assignment	`identifier = expression;` **Example:** `t = -55;`
initialization	`type identifier = expression;` (initializations combine declarations and assignment statements) **Example:** `int i = 3;`
return	`return expression;` **Example:** `return f();`

Step 1: Loading the Classes

Rules for drawing the static space of a class or interface:

1. Draw a box for the class or interface.
2. Write the name of the class or interface in the upper left corner.
3. Write the name of the parent class in the top-right corner, along with any interfaces that follow the keyword `implements`.
4. Draw:
   - static variables: type, name and value
   - static methods: return type and name

Note that only copy of each static member (variable or method) exists, no matter how many instances of a class have been created.

Tracing statement execution

Method call

1. In the code, label the arguments with roman numerals to indicate the order in which the arguments will be evaluated.
2. In order, evaluate each argument and draw a box on the top of the stack to hold the argument value.
3. Draw a frame for the method on top of the stack; include the argument boxes from step 2 inside the new frame.
4. Write the method name in the upper left corner and the method scope in the upper right corner.*
5. Any argument values will be on top of the method stack from step 1. Rename the box for each value to the corresponding parameter name.
6. Write `:1` (the line number) just after the name.
7. Execute the method line-by-line, incrementing the line number.

*The method scope is the address of an object if the method is non-static, and is the name of a class if the method is static.

Other Statements and Expressions

Declaration:

In the current frame, write the variable type and name and draw a box to hold the value.

Assignment:

1. Evaluate the expression on the right side of `=`.
2. Write the result in the variable referred to on the left side.

Initialization: Do the declaration and then the assignment (as above).

return: Evaluate the expression and replace the current method frame with the result value.

Expression: Evaluate the expression inside-out and left to right.

A special expression: operator new*

- **new:**
  1. Draw a new object in the object space.

     Use a stack of boxes to represent each class in the inheritance hierarchy. For each:

     - Write the class name in the upper right corner, along with implemented interfaces.
     - Draw the types, names, and default values of instance variables, and the return types and names of instance methods.

  2. In the topmost box, write the address of the object in the top-left corner. Represent the address with an arbitrary four-bit number (e.g. `0010`, `1010`).
  3. Execute the constructor call. (Its scope is the new object).
  4. When the constructor is done, the value of the `new` expression is the address of the new object.

*Strings are special: `"Wombat"` is shorthand for `new String("Wombat")`.