CS61B Summer 2006
Instructor: Erin Korber
Lecture 8, 10 July

Today, we'll be talking about what happens "behind the scenes" when we run a Java program - what is actually happening in memory and how it affects us as programmers.

1 The call stack

- Stack *frames*
  - Pushed onto the stack when methods are called
  - Holds the state of the method
    - which line of code is executing
    - values of all the local variables
- Local variables (those declared inside a method)
  - Alive as long as their frame is on the stack
  - In scope only within the method that declared them
  - State persists as long as they live, but they can only be used when they are in scope.
  - These rules are the same for primitive and reference variables
- Parameter passing
  - Recall: Java is pass-by-value, so parameters are always copied.
  - Parameters are just local variables, so the copies live in the stack frame for that method as you would expect.
  - The original values that were copied (in order to pass them) are therefore not changed.
  - Remember that we pass object *references*, not objects, so a method might use a reference that is was passed to make changes to an object that are visible everywhere.
- Exceptions
  - When a method throws an exception, its stack frame pops, throwing the exception to the previous frame. So frames keep popping until an exception handler (try/catch) is reached, or we reach the bottom of the stack.

2 The heap

- All objects live on the heap, regardless of whether the references pointing to them are instance or local variables.
- We know local variables live on the stack, inside their methods - but what about instance variables?
- Instance variables live inside their objects (so they are alive as long as the object is).
- Object creation
  - Remember, objects are only created when we say `new` - just declaring a reference does not create an object.
  - When we instantiate a subclass, the superclass object is created first and the subclass parts are "layered" around it.
- Object Death
  - An object lives as long as there are live references to it.
  - 3 ways to kill an object:
    - Its only reference is a local variable, and that variable's frame pops from the stack.
    - Its only reference is explicitly assigned to another object.
    - Its only reference is explicitly set to `null`.