Topics

• Plan for bugs
• How object comparison works
• Boxes and arrows
• Big thoughts to think before Friday

Announcements

• CS Colloquium this Friday, Sept 23 @ 2:35pm in Wege Auditorium (TCL 123)

Michael Greenberg (Stevens Institute of Tech)
Formal Support for the POSIX Shell

The POSIX shell is a widely deployed, powerful tool for managing computer systems. The shell is the expert’s control panel, a necessary tool for configuring, compiling, installing, maintaining, and deploying systems. Even though it is powerful, critical infrastructure, the POSIX shell is maligned and misunderstood. Its power and its subtlety make for a dangerous combination.

How can we support the POSIX shell? I’ll describe two recent lines of work—Smoosh, a formal, mechanized, executable small-step semantics for the POSIX shell—and ffs—a tool for helping users manipulate semi-structured data (like JSON and YAML) in the shell.
Lab 1

How did it go?
{great, ok, not great}

Two objectives of Labs/Office/TA hours:
• Maximize help during hours most in demand.
• Time to work on problem by yourself.

Remember: Learning feels inefficient.

(Un) fun fact:
On average, only 30% of professional programmer time is spent writing new code.

The other 70% is spent designing and debugging code.

(on average, ~50% of total time spend debugging)

Learning how to debug is at least as important as learning how to code.

Expect to do a lot of debugging.

Assume that your code will fail, and build-in checks.

E.g., toString().

Source: “The Mythical Man-Month”, Fred Brooks; University of Cambridge Judge Business School; etc.
Example

```java
class Nim {
    private int[] board;
    private int currentPlayer;

    public String toString() {
        String s = "Player: " + (currentPlayer + 1);
        // draw each row
        for (int row = 0; row < board.length; row++) {
            // draw each match in a row
            s += "[" + (row + 1) + "] " + ";
            for (int m = board[row]; m > 0; m--) {
                s += "| " + "\n";
            }
            s += "\n";
        }
        return s;
    }
}
```

Q: Why do I have to use .equals() to compare String objects?

A:

When comparing values, use ==

When comparing objects, use .equals()

Boxes and arrows (aka “the data structure inside every computer”)
A simple program.

```java
class Program {
    public static void foo() {
        String s1 = new String("Hello class!");
        String s2 = new String("Hello class!");
        System.out.println(s1 == s2);
        System.out.println(s1.equals(s2));
    }
    
    public static void main(String[] args) {
        foo();
    }
}
```

Stack frame: reserved space in memory for local variables.

Q: Do we have any local variables in `main`?
Because the values of s1 and s2 are references. Those references are not the same!
class Program {
    public static void foo() {
        String s1 = new String("Hello class!");
        String s2 = new String("Hello class!");
        System.out.println(s1 == s2);
        System.out.println(s1.equals(s2));
    }
    public static void main(String[] args) {
        foo();
    }
}

Computer: true
Because the objects pointed to by s1 and s2 are the same!

The two objects are no longer pointed to by anything. They are "garbage". Garbage is "collected" in Java.

What does static mean?

The program has now terminated.
Classes are **prototypes**.
Objects are **copies** ("instances").

Methods are functions that are tied to either:
1. a **class**, or
2. an instance of a class (an **object**).

**static methods** are "attached" to class.
**instance methods** are "attached" to object.
How do we know if one algorithm is faster than another?

Why can’t we just measure “wall time”?

Recap & Next Class

Today:

• Study tip #2
• Plan for bugs
• Object comparison

Next class:

• Time and space
• Recursion