

CSCI 136:  
Data Structures  
and  
Advanced Programming  
Lecture 7  
Object equality

Instructor: Dan Barowy  
**Williams**

## Topics

- Study tip #2
- Practice Quiz
- Plan for bugs
- How object comparison works

## Your to-dos

1. Lab 2, **due Tuesday 2/22 by 10pm.**
2. Read **before Wed**: Bailey, Ch 5.2 to end of Ch 5.

## Announcements

- Colloquium: **Senior thesis proposals #2**, 2:35pm in Wege Auditorium... with **cookies**.
- Survey: should I take off my mask during lecture?

## Study tip #2

Confusion is **not necessarily a bad thing**.



Good courses are **intentionally designed** to put you in this state.

**Expect** to have this feeling.

## Study tip #2

### Sometimes Confusion is a Good Thing

Tania Lombrozo  
NPR, December 14, 2015

“Students **who were confused** ... as reflected in inconsistent responses on subsequent questions ... **ultimately did better** on a final test assessing whether they learned the key points from the lessons.”

<https://www.npr.org/sections/13.7/2015/12/14/459651340/sometimes-confusion-is-a-good-thing>

## Study tip #2

### Sometimes Confusion is a Good Thing

Tania Lombrozo  
NPR, December 14, 2015

“... [C]onfusion **is itself a step toward learning** — an experience that motivates the learner to reconcile an inconsistency or remedy some deficit. In this view, confusion isn't just a side effect of beneficial cognitive processes, but a beneficial process itself. ... [T]here's **evidence that experiencing difficulties in learning can sometimes be desirable**, leading to deeper processing and better long-term memory.”

<https://www.npr.org/sections/13.7/2015/12/14/459651340/sometimes-confusion-is-a-good-thing>

## Study tip #2

**Frustration** = **confusion** + **time pressure**



Know that you **need time** to be confused and work through that confusion.

**Learning feels inefficient.**

## Study tip #2

Confusion tells you something **valuable**:



It is a signal that you are **not confident in your knowledge**.

Use this signal to **guide your study** (e.g.: **glossary**).

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**.

## Practice Quiz

## (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.

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

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

E.g., `toString()`.

Hint!

```
class FrequencyList {  
    private Vector<Association<String,Integer>> mylist;  
  
    ...  
  
    public String toString() {  
        String s = "[ ";  
        for (Association<String,Integer> a : mylist) {  
            s += "'" + a.getKey()  
              + " = "  
              + a.getValue() + " ";  
        }  
        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.

```
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();  
    }  
}
```

```
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();  
    }  
}
```

Call stack

```
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();  
    }  
}
```

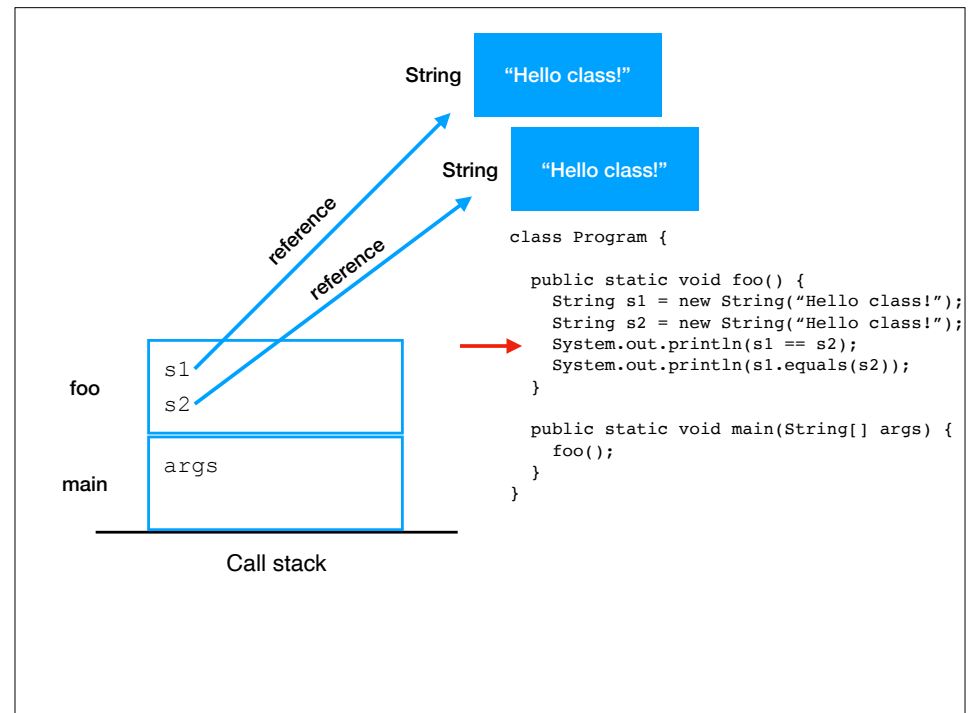
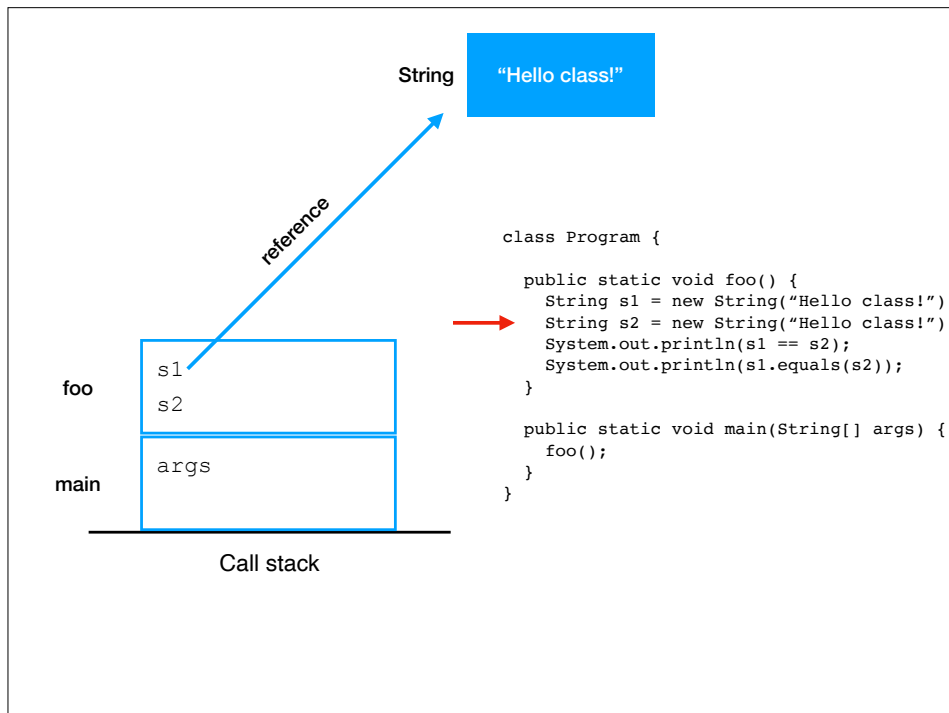
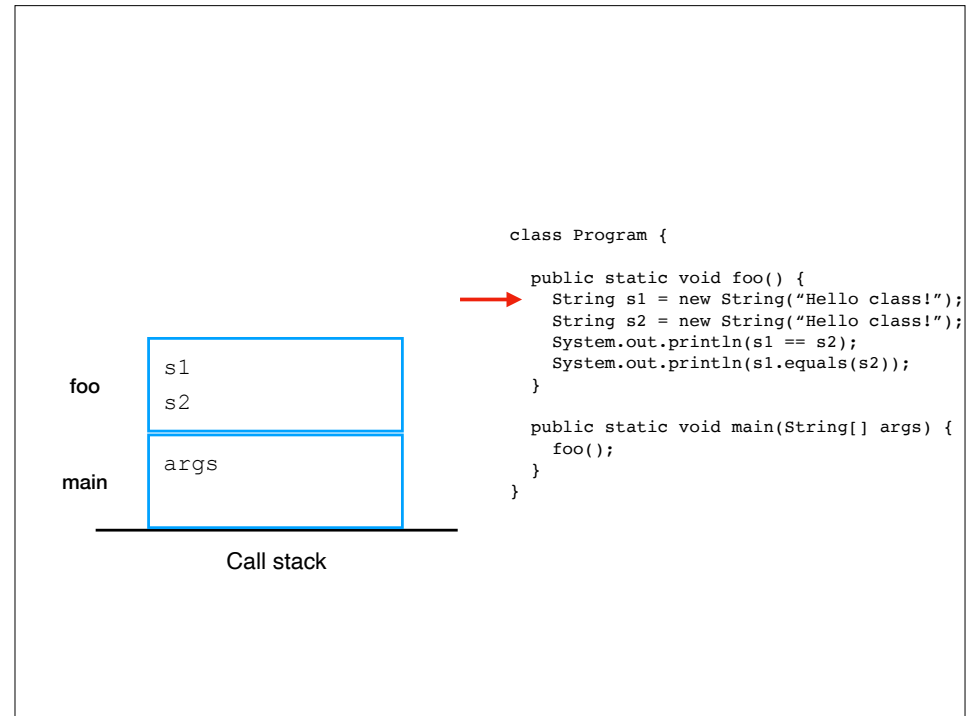
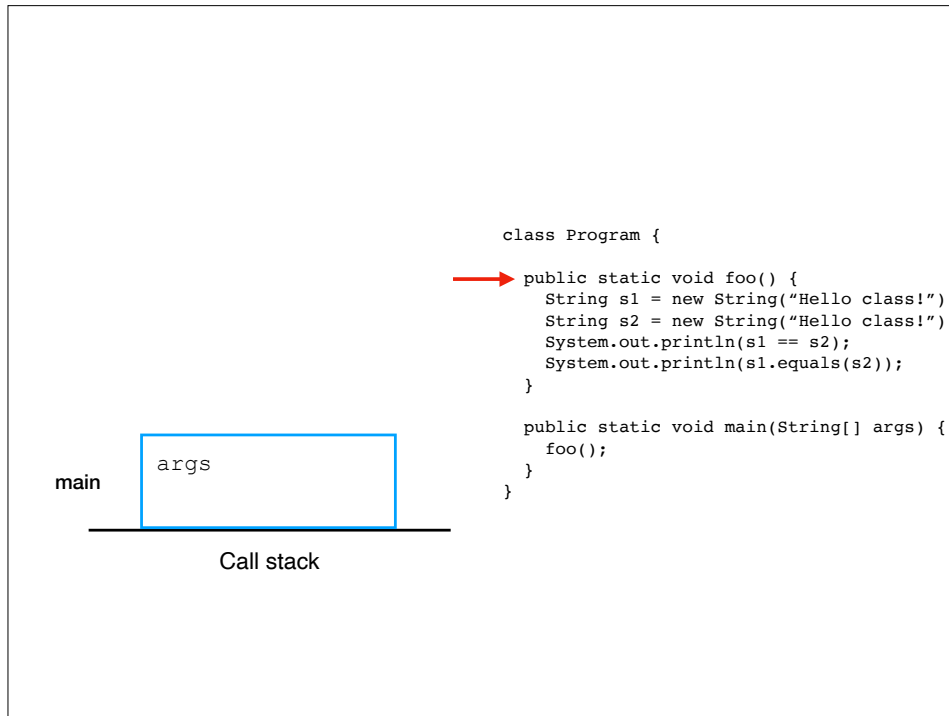
main

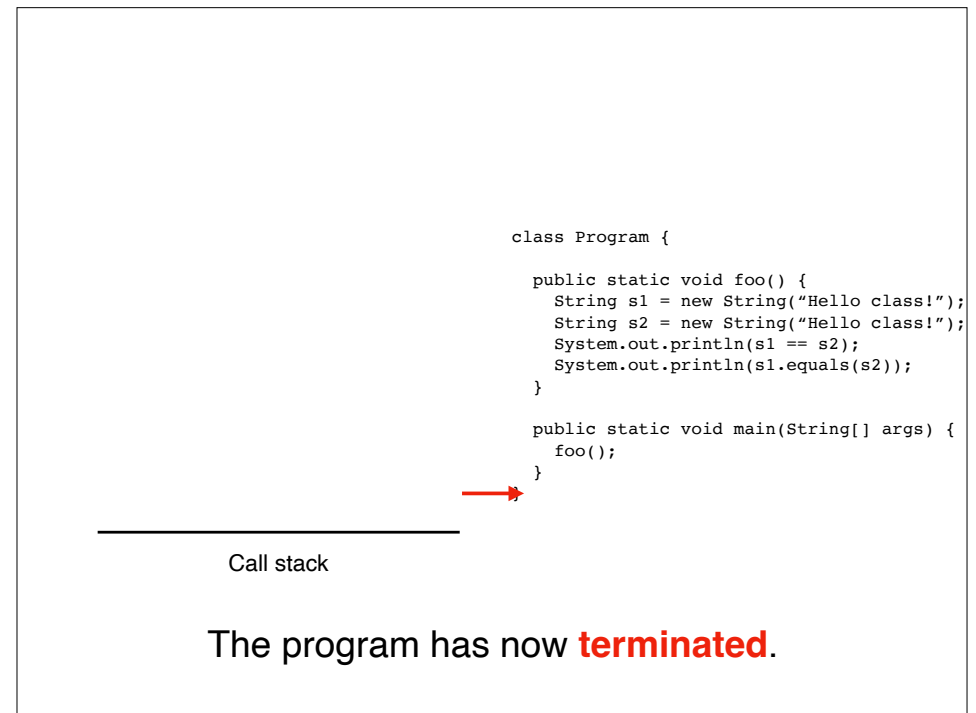
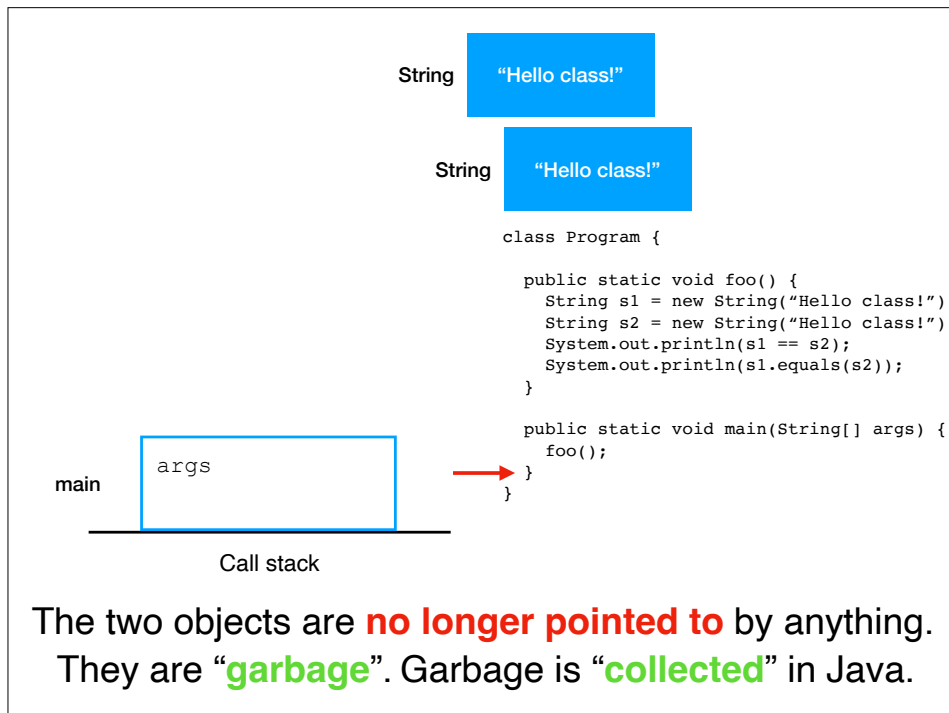
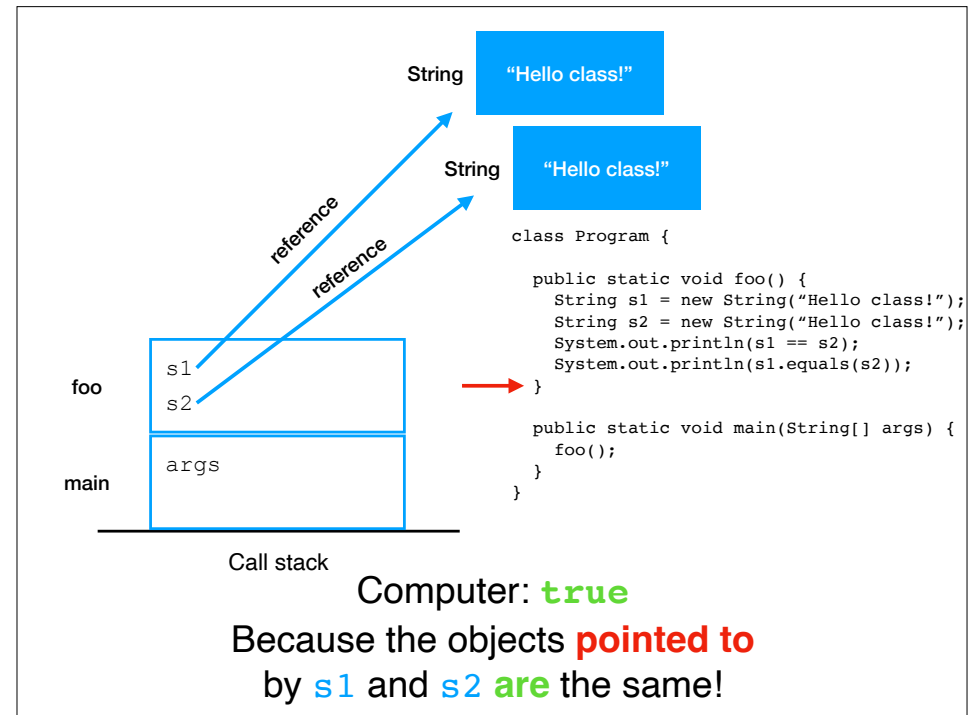
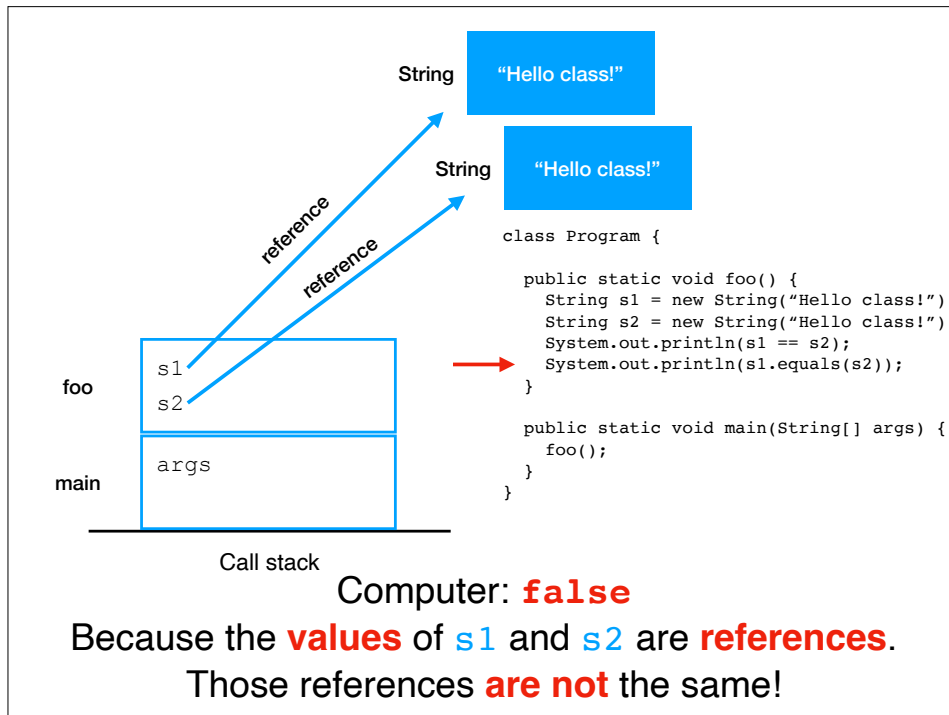
args

Call stack

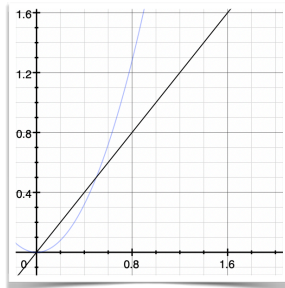
**Stack frame**: reserved space in memory for **local variables**.

**Q**: Do we have any local variables in main?





## Asymptotic analysis



How do we know if an 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