CSCI 136:
Data Structures
and
Advanced Programming

Lecture 6

Composition

Instructor: Dan Barowy

Williams

Outline

UniCS

Study tip

Boxes and arrows

Biased random sampling

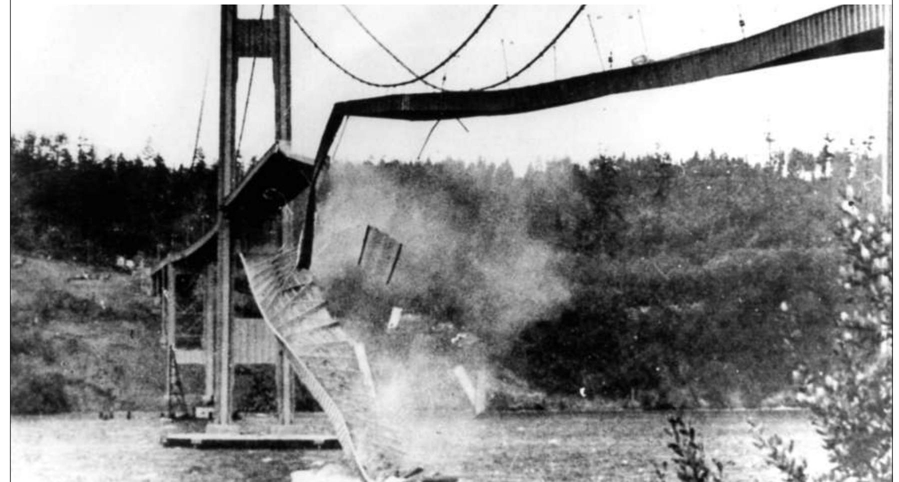
`Dictionary<K,V>`

Quiz

(remember to write your name!)

Life skill #6

"Failure is always an option."



Life skill #6

"'Failure is always an option' came up as a joke … when we were screwing something up over and over again, but it's an awesome way to think about the scientific method. We tend to think about science as … a scientist saying, "I want to prove this thing," and then coming up with an experiment to prove it. Nothing could be further from the truth … [In reality, the] scientist simply says, "I wonder if?" and then builds a methodology to test whether [the] theory is correct, or even to figure out what [the] theory might be. So to think that an experiment could "fail" is ludicrous. **Every experiment tells you something, even if it's just don't do that experiment the same way again.**" — Adam Savage

---

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()

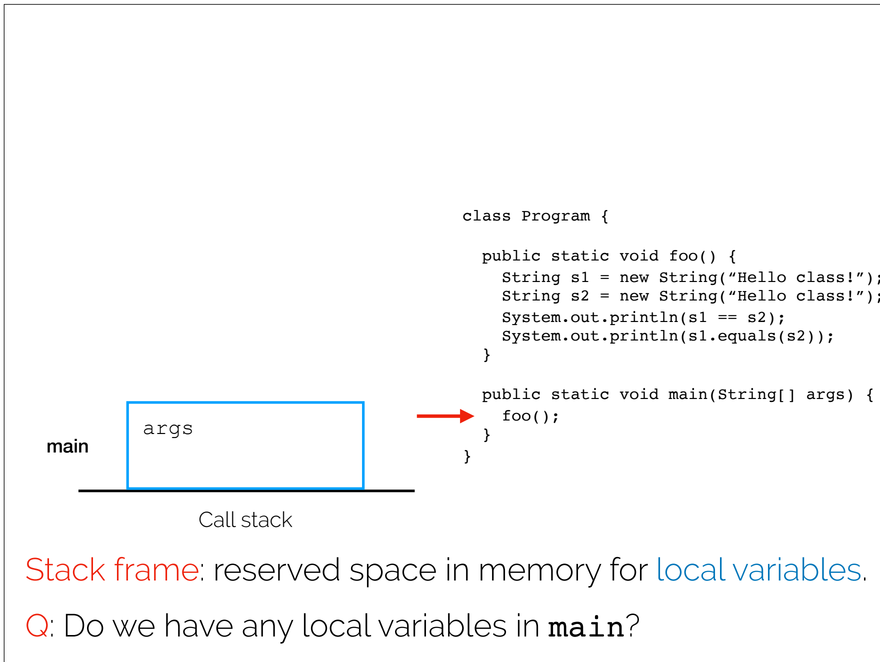
Boxes and arrows
(aka "the data structure inside every computer")

A simple program.

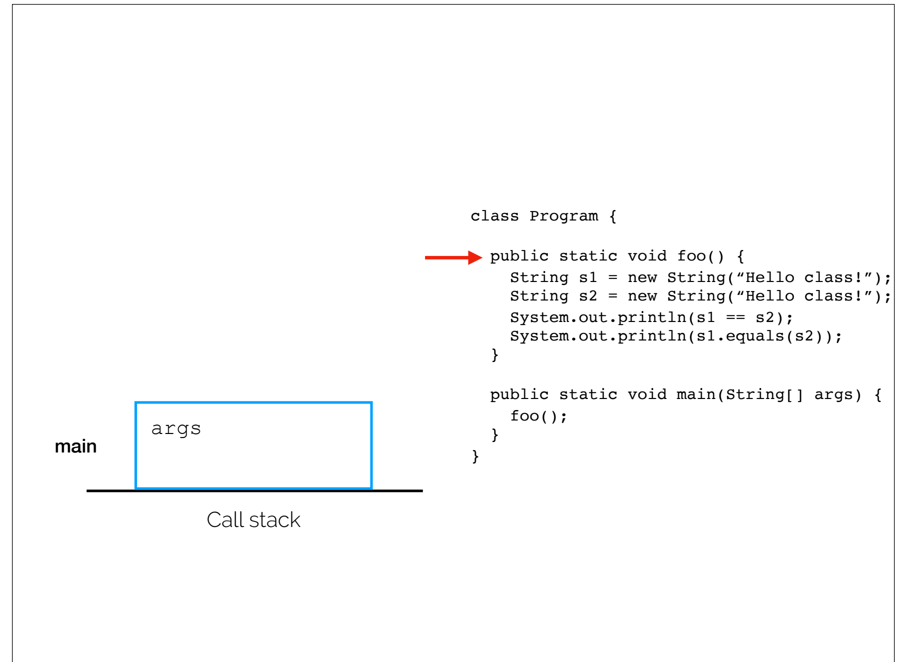
```
class Program {

  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();
  }
}
```

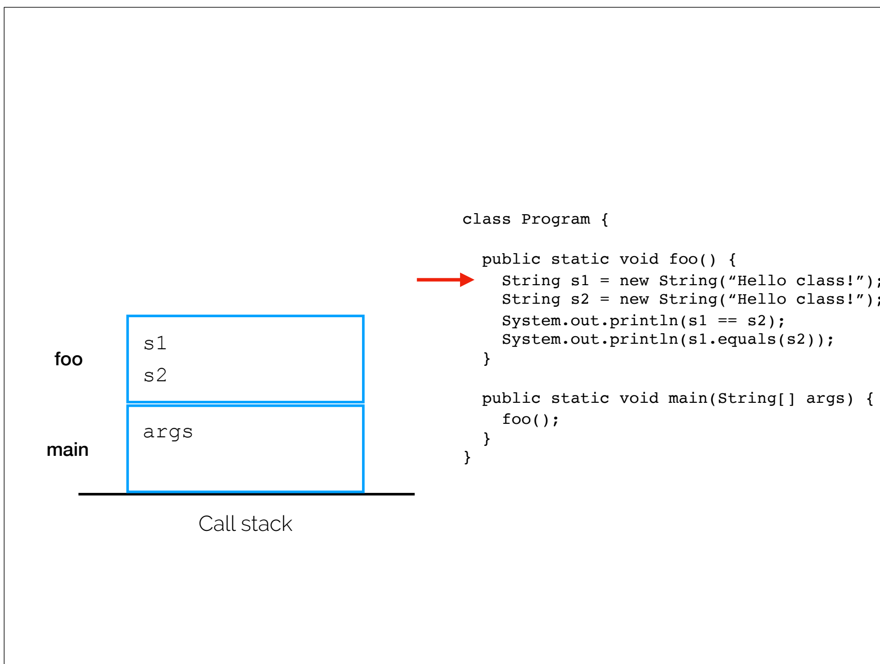
Call stack



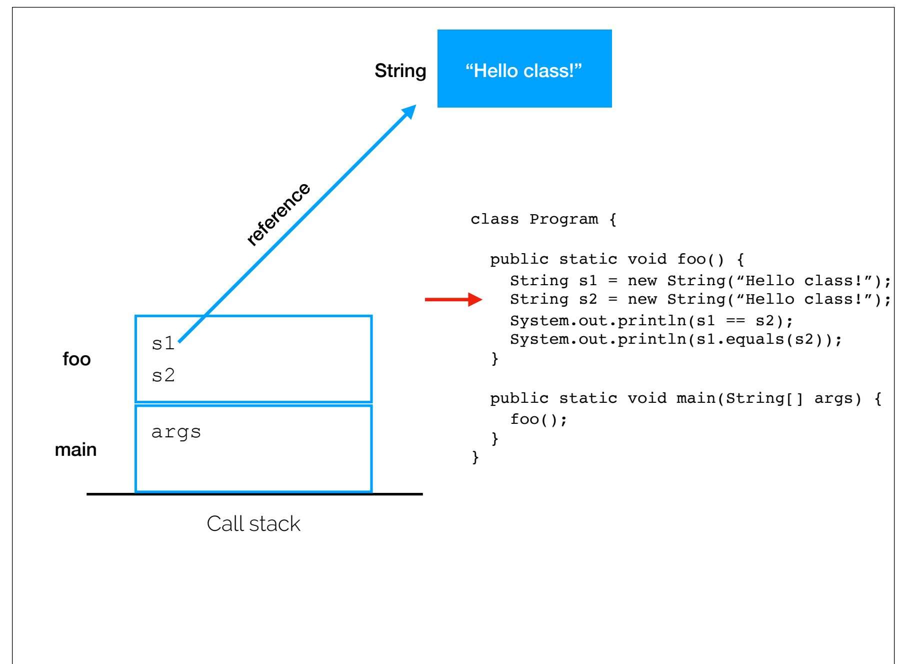
class Program {
public static void foo() {
String s1 = new String("Hello class!");
String s2 = new String("Hello class!");
System.out.println(s1 == s2);
System.out.println(s1.equals(s2));
}
public static void main(String[] args) {
foo();
}
}
main
args
Call stack
Stack frame: reserved space in memory for local variables.
Q: Do we have any local variables in main?



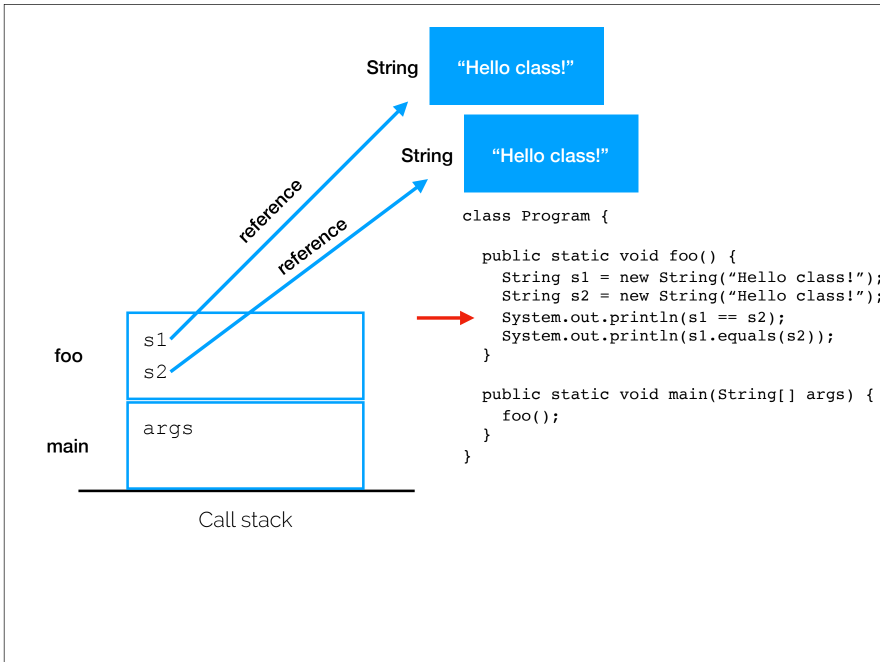
class Program {
public static void foo() {
String s1 = new String("Hello class!");
String s2 = new String("Hello class!");
System.out.println(s1 == s2);
System.out.println(s1.equals(s2));
}
public static void main(String[] args) {
foo();
}
}
main
args
Call stack

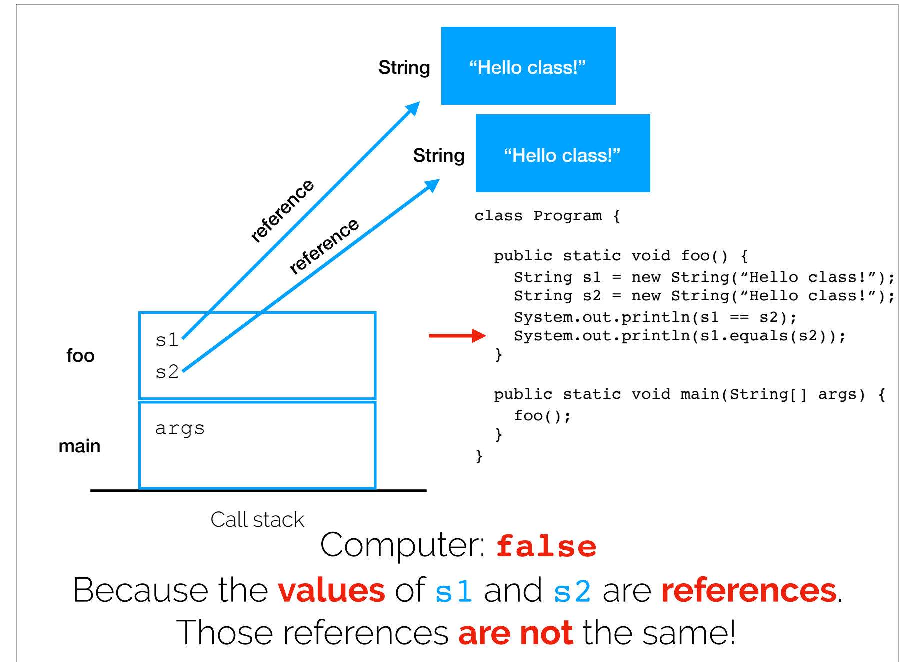


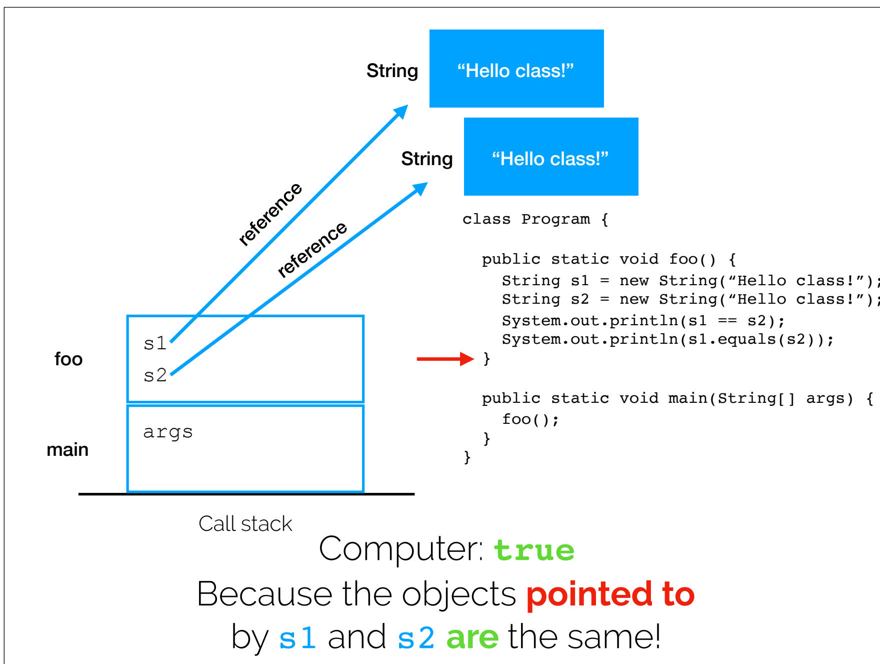
class Program {
public static void foo() {
String s1 = new String("Hello class!");
String s2 = new String("Hello class!");
System.out.println(s1 == s2);
System.out.println(s1.equals(s2));
}
public static void main(String[] args) {
foo();
}
}
foo
s1
s2
main
args
Call stack

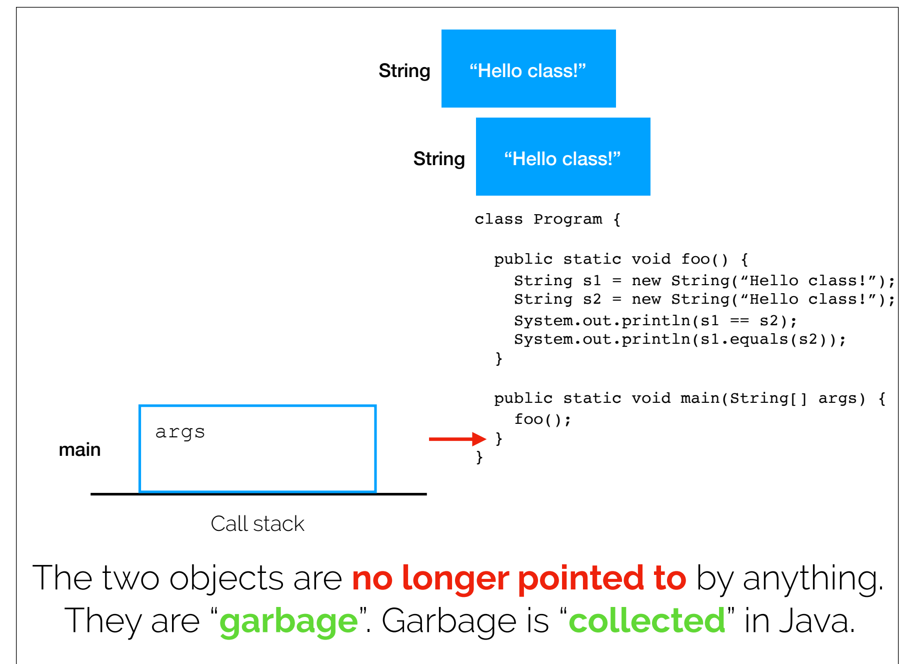


String
"Hello class!"
reference
class Program {
public static void foo() {
String s1 = new String("Hello class!");
String s2 = new String("Hello class!");
System.out.println(s1 == s2);
System.out.println(s1.equals(s2));
}
public static void main(String[] args) {
foo();
}
}
foo
s1
s2
main
args
Call stack


String
"Hello class!"
String
"Hello class!"
reference
reference
foo
s1
s2
main
args
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();
}
}


String
"Hello class!"
String
"Hello class!"
reference
reference
foo
s1
s2
main
args
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();
}
}
Computer: false
Because the values of s1 and s2 are references.
Those references are not the same!


String
"Hello class!"
String
"Hello class!"
reference
reference
foo
s1
s2
main
args
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();
}
}
Computer: true
Because the objects pointed to
by s1 and s2 are the same!


String
"Hello class!"
String
"Hello class!"
main
args
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();
}
}
The two objects are no longer pointed to by anything.
They are "garbage". Garbage is "collected" in 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();
  }
}
```

Call stack

The program is now **terminated**.

Biased sampling

What does this program do?

```
Random r = new Random();
int num = r.nextInt(10);
```

Chooses a value between 0 and 9 inclusive with uniformly random probability.

I.e., all values are equally likely.

What if we want to specify the likelihood?

letter	likelihood
'a'	1
'b'	6
'c'	3

A naïve algorithm

'a'	'b'	'b'	'b'	'b'	'b'	'b'	'c'	'c'	'c'
0	1	2	3	4	5	6	7	8	9

```
char[] arr = new char[10];
// … code to fill array …
Random r = new Random();
int num = r.nextInt(10);
char c = arr[num];
```

A better algorithm

letter	likelihood
'a'	1
'b'	6
'c'	3

1. Compute the **sum of the likelihoods** (here: **10**).
2. Choose a number **n** between **0 … sum** (exclusive) uniformly randomly.
3. For each letter, subtract the **likelihood** from **n**.
4. When **n becomes negative**, you've "found" the right letter.

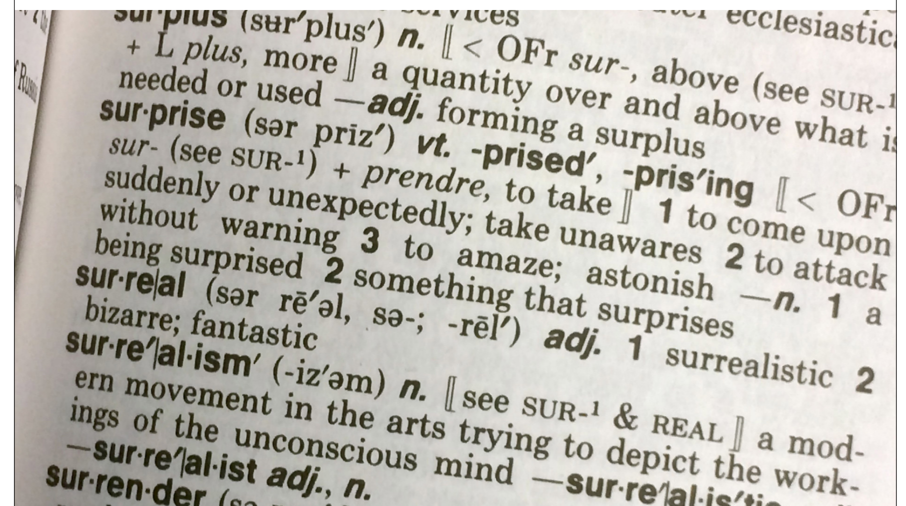
Try it at home!

Notice that you get the **same answer** had you used the naïve method.

'a'	'b'	'b'	'b'	'b'	'b'	'b'	'c'	'c'	'c'
0	1	2	3	4	5	6	7	8	9

1. Compute the **sum of the likelihoods** (here: **10**).
2. Choose a number **n** between **0 … sum** (exclusive) uniformly randomly.
3. For each letter, subtract the **likelihood** from **n**.
4. When **n becomes negative**, you've "found" the right letter.

Dictionary data structure


sur·plus (sur′plus′) ***n.*** ‖ < OFr *sur-*, above (see SUR-1 + L *plus*, more ‖ a quantity over and above what is needed or used —***adj.*** forming a surplus
sur·prise (sər prīz′) ***vt.*** **-prised′**, **-pris′ing** ‖ < OFr *sur-* (see SUR-1) + *prendre*, to take ‖ **1** to come upon suddenly or unexpectedly; take unawares **2** to attack without warning **3** to amaze; astonish —***n.*** **1** a being surprised **2** something that surprises
sur·re|al (sər rē′əl, sə-; -rēl′) ***adj.*** **1** surrealistic **2** bizarre; fantastic
sur·re′al·ism′ (-iz′əm) ***n.*** ‖ see SUR-1 & REAL ‖ a modern movement in the arts trying to depict the workings of the unconscious mind —**sur·re′al·is′tic** —**sur·re′al·ist** ***adj.***, ***n.***
sur·ren·der

Wouldn't it be great if we could look up words in a Java dictionary like this?

```
Dictionary<String,String> d = …
// … code that populates d
d.lookup("computer science");
```

Given **key**

Returns **value** (something like: `"blah blah blah computers"`)

```java
import structure5.*;

class Dictionary<K,V> {
    private Vector<Association<K,V>> v;

    public Dictionary() {
        v = new Vector<>();
    }

    public void add(K key, V value) {
        Association<K,V> a = new Association<>(key, value);
        int i = v.indexOf(a);
        if (i == -1) {
            // add new entry
            v.add(a);
        } else {
            // update
            v.get(i).setValue(value);
        }
    }

    public V lookup(K key) {
        for (Association<K,V> a : v) {
            if (a.getKey().equals(key)) {
                return a.getValue();
            }
        }
        return null;
    }

    public static void main(String[] args) {
        Dictionary<String,String> d = new Dictionary<>();
        d.add("computer science", "blah blah blah computers");
        System.out.println(d.lookup("computer science");
    }
}
```

Recap & Next Class

Today we learned:

- Boxes and arrows (memory!)
- Biased sampling
- `Dictionary<K,V>`

Next class:

- Recursion