# cs2220: Engineering Software

## Class 3:
## Java Semantics

Fall 2010
University of Virginia
David Evans

# Menu

Java **Semantics**

Stack and Heap

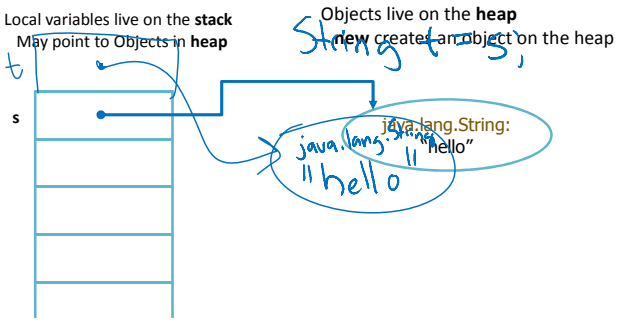

# The Stack and Heap

**String s = new String ("hello");**

(Almost) equivalent to: **String s = "hello";**

Local variables live on the **stack**
May point to Objects in **heap**

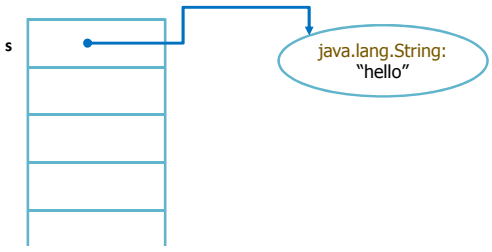
Objects live on the **heap**
**new** created an object on the heap




# The Stack and Heap

String s = new String ("hello");

**String t = s;**

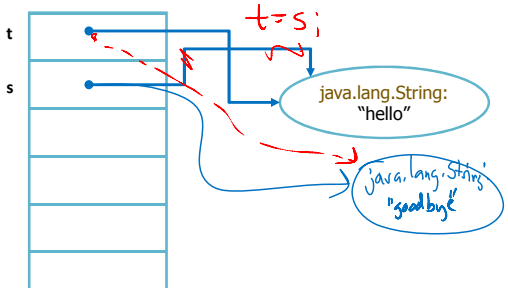



# The Stack and Heap

String s = new String ("hello");

String t = s;

**s = new String ("goodbye");**




# Primitive Types

*Almost* everything in Java is an Object

The exceptions are ***primitive types:***

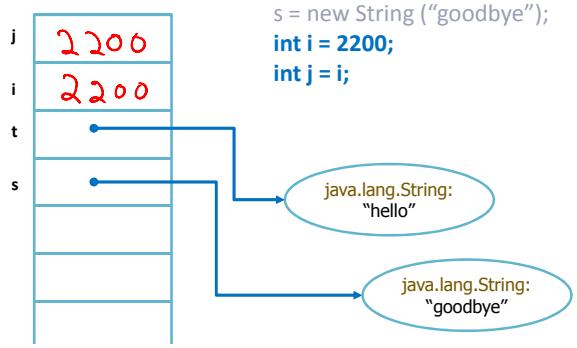
**boolean**, byte, char, **double**, float, **int**, long, short

Primitive types have different semantics!

Values of a primitive type are stored directly on the stack

# Primitive Types

```
String s = new String ("hello");
String t = s;
s = new String ("goodbye");
int i = 2200;
int j = i;
```




# Does it matter?

Does it matter if something is stored on the stack or the heap?

Can we see the difference between primitive types and objects?

# Equality

***x == y***

Object Types: **same objects**

Primitive Types: **same value**

***x*.equals (*y*)**

Object Types: method that compares **values** of objects

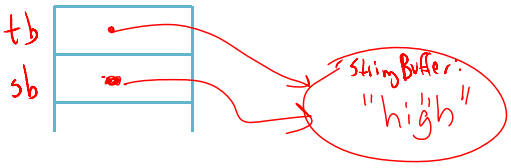
Primitive Types: **doesn't exist**

Preview: the **equals** method is defined in java.lang.Object, which is the ultimate **superclass** of all classes. Other classes **override** equals to mean different things.

# Mutability

When an Object is mutated, all references to the Object see the new value.

```
StringBuffer sb = new StringBuffer ("hi");
StringBuffer tb = sb;
tb.append ("gh");
```




# Immutable/Mutable Types

Types can be **mutable** or **immutable**

Objects of an immutable type never change value after they are created

**String** is immutable, **StringBuffer** is mutable

**String.concat** creates a new String object

**StringBuffer.append** mutates this object

Note: **StringBuilder** is almost identical to **StringBuffer**.

```
public class Strings {
  public static void test (String [] args) {
    String s = new String ("hello");
    String t = new String ("hello");
    StringBuffer sb = new StringBuffer ("he");
    StringBuffer tb = sb;
    String s1 = "hello";
    String t1 = "hello";

    sb.append ("llo");
    tb.append (" goodbye!");
    s.concat (" goodbye!");
    t = s.concat (" goodbye!");
  }
}
```

What are the values of **s**, **t**, **sb** and **tb** no
Which of these are/must be true:

a) **s == t**
b) **s1 == t1**
c) **s == s1**
d) **s.equals (t)**
e) **sb == tb**
f) **t.equals (tb)**

```
public class Strings {
  public static void test (String [] args) {
    String s = new String ("hello");
    String t = new String ("hello");
    StringBuffer sb = new StringBuffer ("he");
    StringBuffer tb = sb;
    String s1 = "hello";
    String t1 = "hello";

    sb.append ("llo");
    tb.append (" goodbye!");
    s.concat (" goodbye!");
    t = s.concat (" goodbye!");
  }
}
```

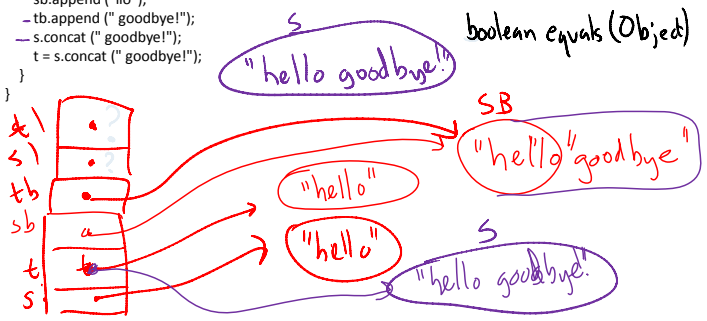
s.equals(t) == t.equals(s)

What are the values of **s**, **t**, **sb** and **tb** now?
Which of these are/must be true:

a) **s == t** FALSE
b) **s1 == t1** TRUE
c) **s == s1** FALSE
d) **s.equals (t)** FALSE
e) **sb == tb** TRUE
f) **t.equals (tb)** FALSE

new String("hello")

boolean equals (Object)




## Java Language Specification

(Section 3.10.5: String Literals)

Each string literal is a reference (§4.3) to an instance (§4.3.1, §12.5) of class String (§4.3.3). String objects have a constant value. String literals-or, more generally, strings that are the values of constant expressions (§15.28)-are "interned" so as to share unique instances, using the method String.intern.

## Summary

Java sacrificed simplicity and coherence for performance: primitive types are not Objects

**Cost:** programmers have to think about stack, heap, semantics differences

**Benefit:** saves memory, perhaps better performance, more like C/C++

PS2: Part I posted now; Part II posted later.

Reading before next class: Chapters 3 and 9