

# CS/ENGRD 2110

## FALL 2016

Lecture 4: The class hierarchy; static components  
<http://courses.cs.cornell.edu/cs2110>

# Announcements

2

- We're pleased with how many people are already working on **A1**, as evidenced by Piazza activity
  - ▣ Please be sure to look at **Piazza note @54** every day for frequently asked questions and answers
  - ▣ **Groups:** Forming a group of two? Do it well before you submit – at least one day before. **Both members must act:** one invites, the other accepts. Thereafter, only **one** member has to submit the files.
- **A2: Practice with strings**
  - ▣ We will give you our test cases soon!

# References to text and JavaSummary.pptx

3

- A bit about **testing** and test cases
- Class **Object**, **superest** class of them all.  
Text: C.23 slide 30
- Function **toString()** C.24 slide 31-33
- **Overriding** a method C15–C16 slide 31-32
- **Static** components (methods and fields) B.27 slide 21, 45
- Java **application**: a program with a class that declares a method with this signature:

**public static void** main(String[])

# Homework

4

1. Read the text, about applications: Appendix A.1–A.3
2. Read the text, about the if-statement: A.38–A.40
3. Visit course website, click on **Resources** and then on Code Style **Guidelines**. Study
  2. Format Conventions
  - 4.5 About then-part and else-part of if-statement



# A bit about testing

5

**Test case:** Set of input values, together with the expected output.

Develop test cases for a method from its specification --- even before you write the method's body.

```
/** return the number of vowels in word w.
```

```
Precondition: w contains at least one letter and nothing but letters */
```

```
public int numberOfVowels(String w) {
```

```
    ...
```

```
}
```

How many vowels in each of these words?

creek

syzygy

yellow

Developing test cases first, in “critique” mode, can prevent wasted work and errors

# Class W (for Worker)

6

**/\*\* Constructor: worker with last name n, SSN s, boss b (null if none).  
Prec: n not null, s in 0..999999999 with no leading zeros.\*/**

**public W(String n, int s, W b)**

**/\*\* = worker's last name \*/**

**public String getLname()**

**/\*\* = last 4 SSN digits \*/**

**public String getSsn()**

**/\*\* = worker's boss (null if none) \*/**

**public W getBoss()**

**/\*\* Set boss to b \*/**

**public void setBoss(W b)**

Contains other methods!

W@af

lname

"Obama"

ssn

123456789

boss

null

W(...) getLname()

getSsn() getBoss() setBoss(W)

toString()

equals(Object) hashCode()

W

# Class Object: the superest class of them all

7

**Java:** Every class that does not extend another extends class Object. That is,

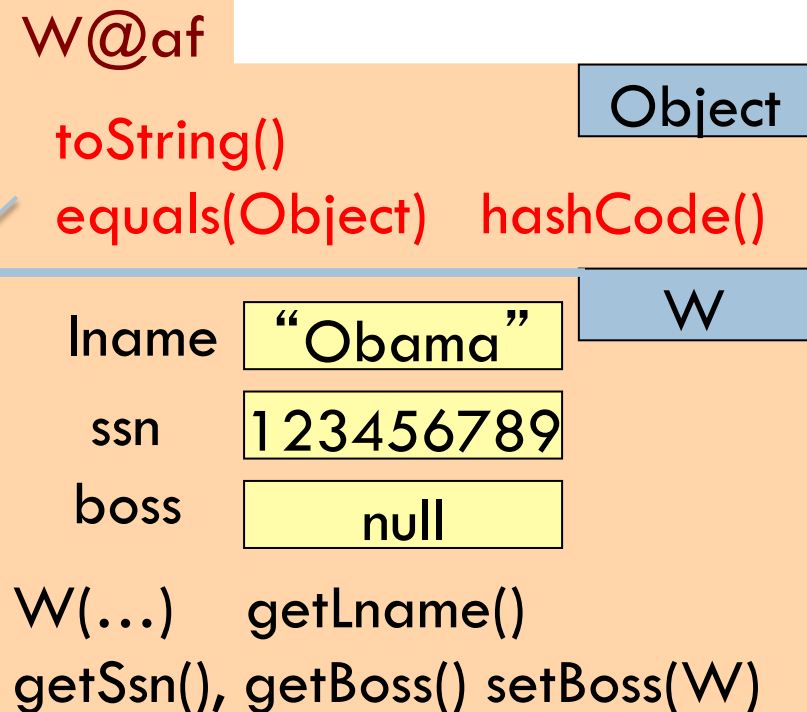
**public class W {...}**

is equivalent to

**public class W extends Object {...}**

We often omit this partition to reduce clutter; we know that it is always there.

We draw object like this



# A note on design

8

- Don't use **extends** just to get access to hidden members!
- **A** should extend **B** if and only if **A** “is a” **B**
  - A PhDTester is *not* a PhD Student!
  - An elephant is an animal, so **Elephant extends Animal**
  - A car is a vehicle, so **Car extends Vehicle**
  - An instance of any class is an object, so **AnyClass extends java.lang.Object**
- The inheritance hierarchy should reflect **modeling semantics**, not implementation shortcuts



# What is “the name of” the object?

9

The name of the object below is

**PhD@aa11bb24**

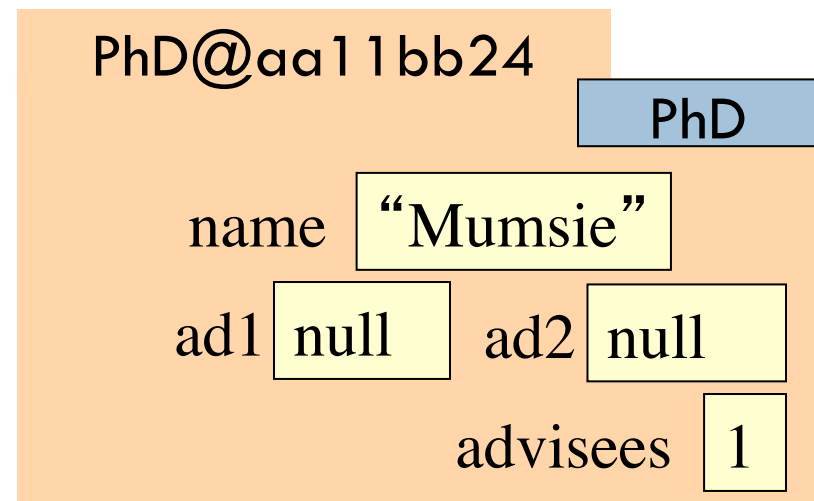
It contains a pointer to the object –i.e. its address in memory, and you can call it a pointer if you wish. But it contains more than that.

Variable **e**, declared as

**PhD e;**

contains not the object but the name of the object (or a pointer to the object).

e **PhD@aa11bb24** PhD



# Method toString

10

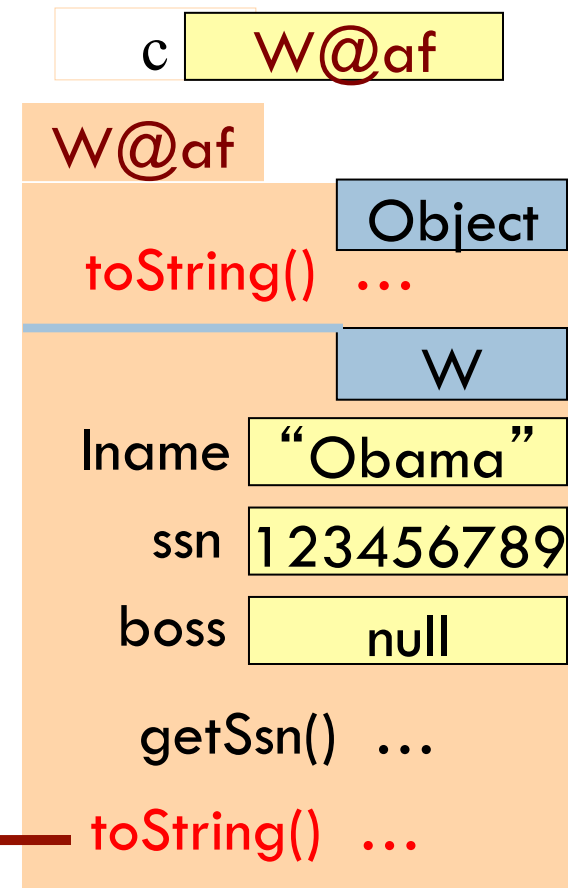
`toString()` in `Object` returns the name of the object: `W@af`

**Java Convention:** Define `toString()` in any class to return a representation of an object, giving info about the values in its fields.

New definitions of `toString()` **override** the definition in `Object.toString()`

In appropriate places, the expression `c` automatically does `c.toString()`

`c.toString()` calls this method



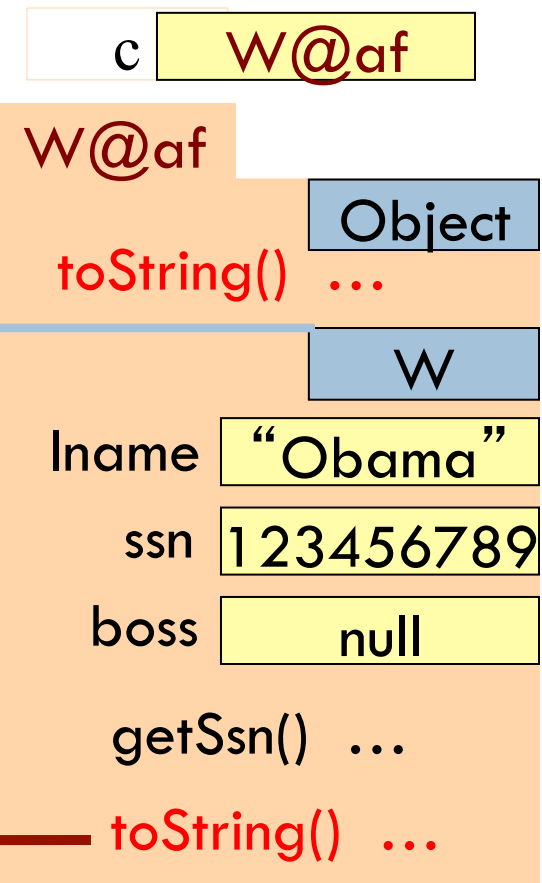
# Method toString

11

`toString()` in `Object` returns the name of the object: `W@af`

```
public class W {  
    ...  
    /** Return a representation of this object */  
    public String toString() {  
        return "Worker " + lname  
            + " has SSN ???-??-" + getSsn()  
            + (boss == null  
                ? ""  
                : " and boss " + boss.lname);  
    }  
}
```

`c.toString()` calls this method



# Another example of toString()

12

**/\*\* An instance represents a point (x, y) in the plane \*/**

**public class Point {**

**private int x;** // x-coordinate

**private int y;** // y-coordinate

...

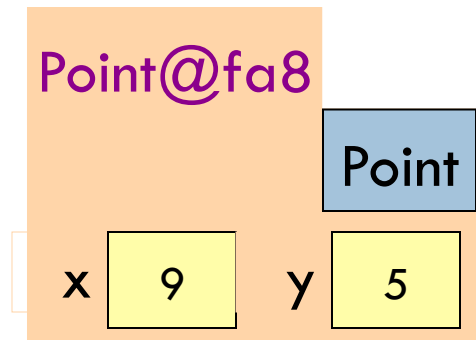
**/\*\* = repr. of this point in form “(x, y)” \*/**

**public String toString() {**

**return “(” + x + “,” + y + “)”;**

**}**

**}**



(9, 5)

Function toString should give the values in the fields in a format that makes sense for the class.

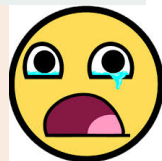
# What about **this**

13

- **this** keyword: **this** evaluates to the name of the object in which it occurs
- Makes it possible for an object to access its own name (or pointer)
- Example: Referencing a shadowed class field

```
public class Point {  
    public int x = 0;  
    public int y = 0;  
  
    //constructor  
    public Point(int x, int y) {  
        x = x;  
        y = y;  
    }  
}
```

Inside-out rule shows that  
field x is inaccessible!



```
public class Point {  
    public int x = 0;  
    public int y = 0;  
  
    //constructor  
    public Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```

# Intro to static components

14

**/\*\*** = “this object is c’s boss”.

**Pre:** c is not null. **\*/**

```
public boolean isBoss(W c) {  
    return this == c.boss;  
}
```

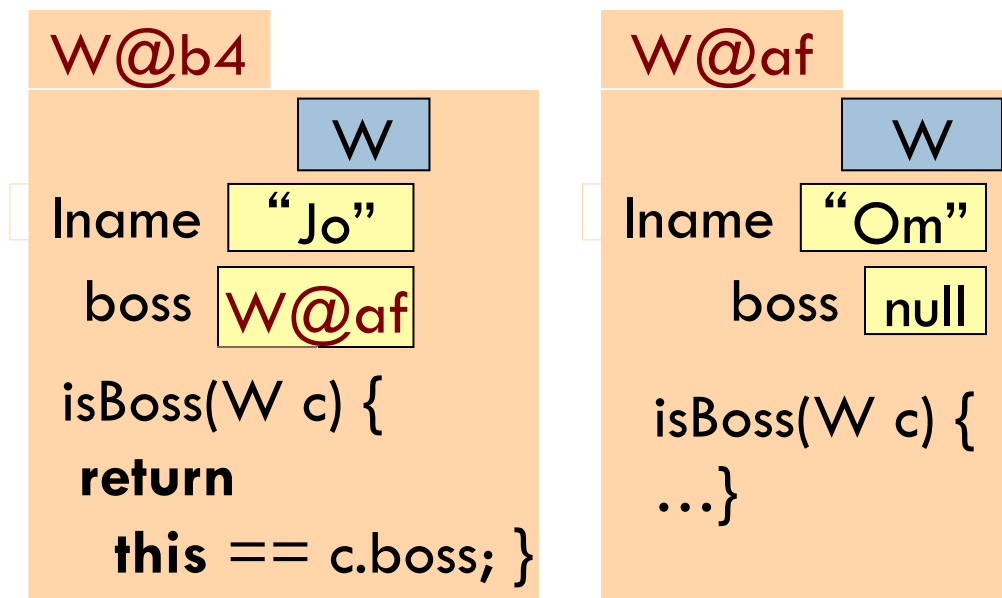
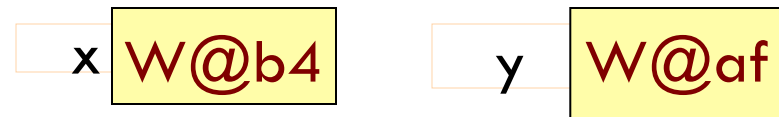
**Spec:** return the value of that true-false sentence.

True if this object is c’s boss, false otherwise

keyword **this** evaluates to the name of the object in which it appears

x.isBoss(y) is **false**

y.isBoss(x) is **true**



# Intro to static components

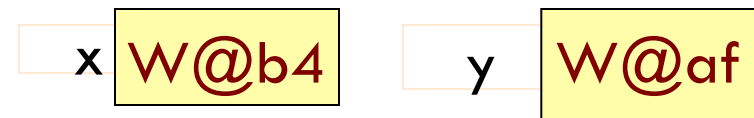
15

/\*\* = “b is c’ s boss”.

Pre: b and c are not null. \*/

```
public boolean isBoss(W b, W c) {  
    return b == c.getBoss();  
}
```

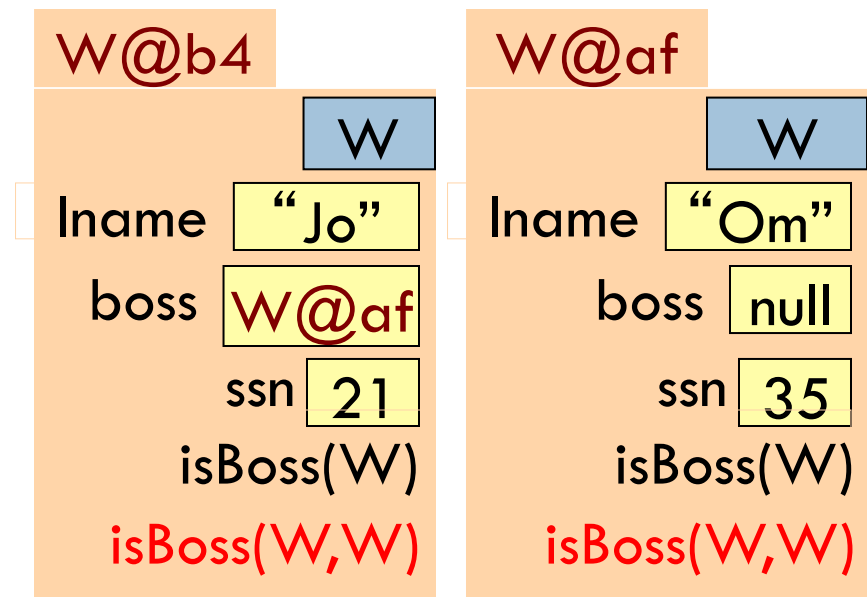
Body doesn't refer to any field or method in the object.  
Why put method in object?



/\*\* = “this object is c’ s boss”.

Pre: c is not null. \*/

```
public boolean isBoss(W c) {  
    return this == c.boss;  
}
```



# Intro to static components

16

```
/** = "b is c's boss".
```

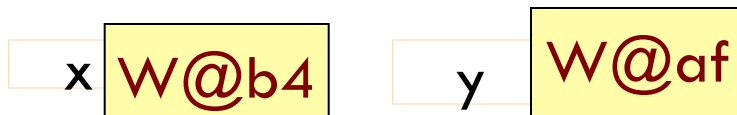
```
Pre: b and c are not null. */
```

```
public static boolean isBoss(W b, W c) {  
    return b == c.getBoss();  
}
```

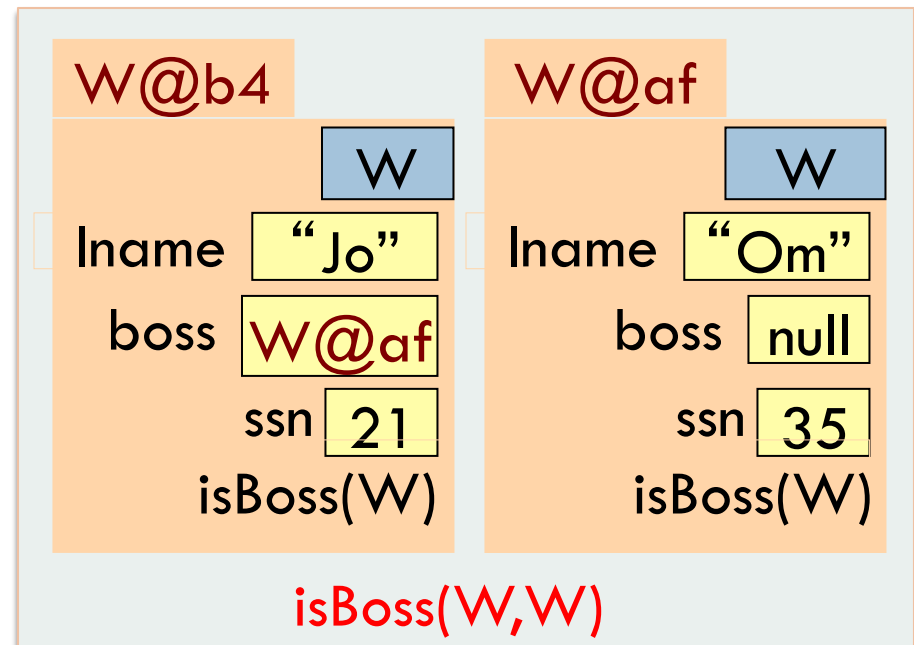
**static**: there is only **one** copy of the method. It is *not* in each object

~~x.isBoss(x, y)~~  
~~y.isBoss(x, y)~~

**Preferred:**  
W.isBoss(x, y)



Box for **W** (objects, **static** components)





# Good example of static methods

17

□ `java.lang.Math`

<http://docs.oracle.com/javase/8/docs/api/java/lang/Math.html>

# Java application

18

Java application: bunch of classes with at least one class that has this procedure:

```
public static void main(String[] args) {  
    ...  
}
```

Type String[]: array of elements of type **String**.  
We will discuss later

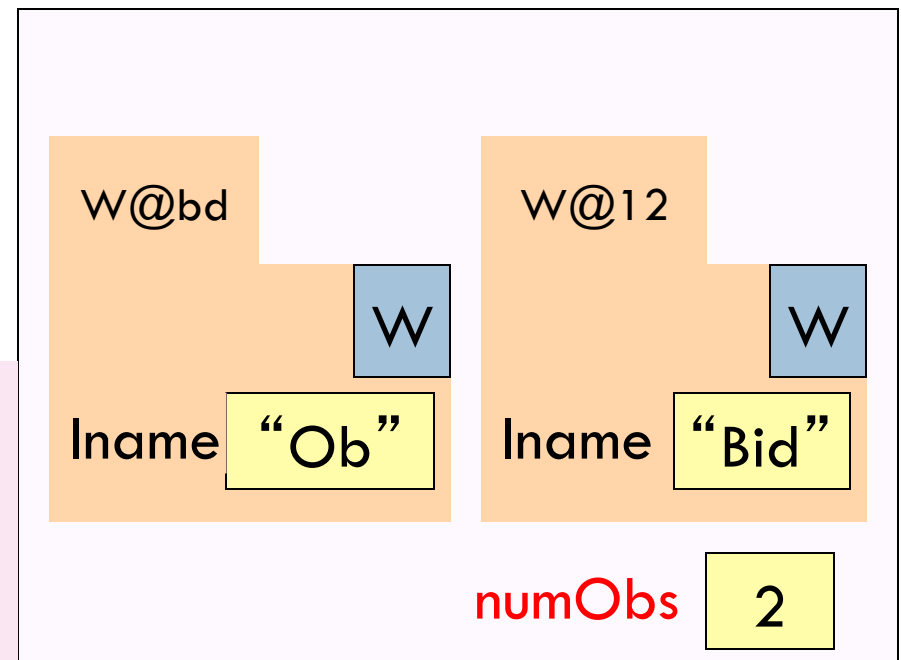
Running the application effectively calls method **main**  
Command line arguments can be entered with **args**

## Use of static variables: Maintain info about created objects

19

```
public class W {  
    private static int numObs; // number of W objects created  
  
    /** Constructor: */  
    public W(...) {  
        ...  
        numObs= numObs + 1;  
    }  
}
```

To have **numObs** contain the number of objects of class **W** that have been created, simply increment it in constructors.



Box for W

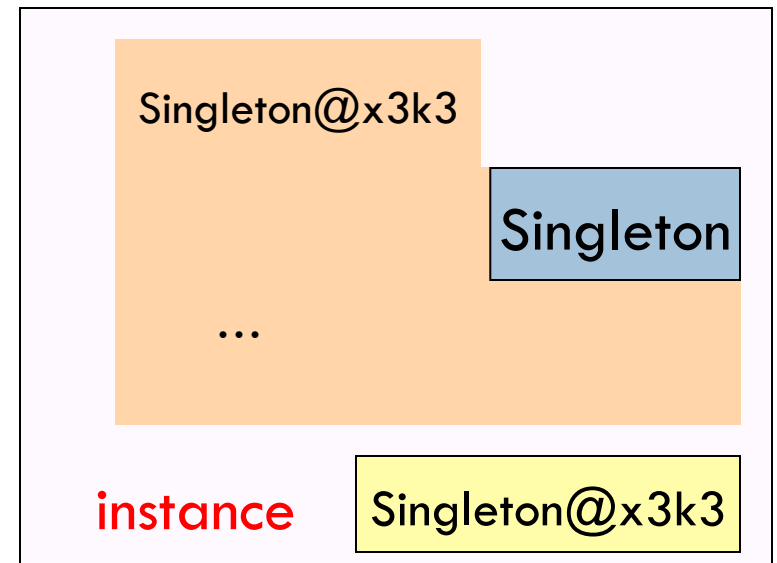
# Uses of static variables:

## Implement the Singleton pattern

20

Only one Singleton can ever exist.

```
public class Singleton {  
    private static final Singleton instance = new Singleton();  
  
    private Singleton() { } // ... constructor  
  
    public static Singleton getInstance() {  
        return instance;  
    }  
  
    // ... methods  
}
```



Box for Singleton

# Class `java.awt.Color` uses static variables

21

An instance of class `Color` describes a color in the RGB (Red-Green-Blue) color space. The class contains about 20 static variables, each of which is (i.e. contains a pointer to) a non-changeable `Color` object for a given color:

```
public static final Color black = ...;
public static final Color blue = ...;
public static final Color cyan = new Color(0, 255, 255);
public static final Color darkGray = ...;
public static final Color gray = ...;
public static final Color green = ...;
...
```