CS/ENGRD 2110
SPRING 2019

Lecture 4: The class hierarchy; static components

http://cs.cornell.edu/courses/cs2110

Announcements

We're pleased with how many people are already working on **A1**, as evidenced by Piazza activity!

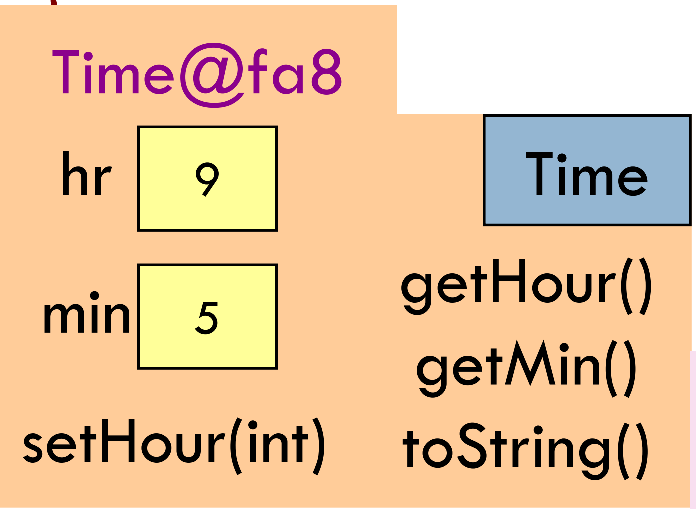
- Please be sure to look at **Piazza note @10** every day for any updates.
- Also search existing questions!
- **Groups:** Forming a group of two? Do it **well before** you submit – at least one day before. **Both members must act:** one invites, the other accepts. Thereafter, only ***one*** member has to submit the files. If one of you submits before forming the group, the course staff will have to do extra work, and you'll receive a small penalty of 4 points.
- **Reminder**: groups must complete the assignment working together.

Big ideas so far

- Java variables have *types* (L1)
  - A type is a set of values and operations on them (`int`: `+, -, *, /, %,` etc.)
- *Classes* define new types (L2)
  - *Methods* are the operations on objects of that class.
  - *Fields* allow objects to store data (L3)
- A software engineering principle: give user access to *functionality*, not the *implementation details*…

Review: Method specs should not mention fields

```
public class Time {
  private int hr;    //in 0..23
  private int min; //in 0..59
  /** return hour of day*/
  public int getHour() {
     return hr;
  }
```



Time@fa8

hr 9

min 5

Time

getHour()

getMin()

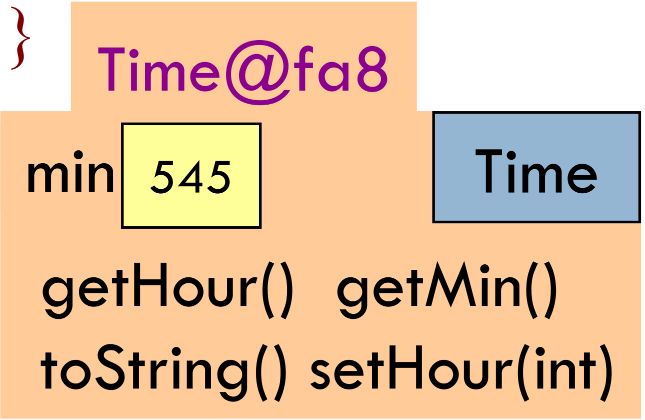
setHour(int)

toString()

Decide to change implementation

```
public class Time {
     // min, in 0..23*60+59
     private int min;

     /** return hour of day*/
     public int getHour() {
          return min / 60;
     }
```



Time@fa8

min 545

Time

getHour() getMin()

toString() setHour(int)

Specs of methods stay the same.

Implementations, including fields, change!

Today's topics

- Class **`Object`**
- Extends, is-a
- Method **`toString()`**, object names, overriding
- Keyword **`this`**, shadowing
- Static components

Running example: Class W (for Worker)

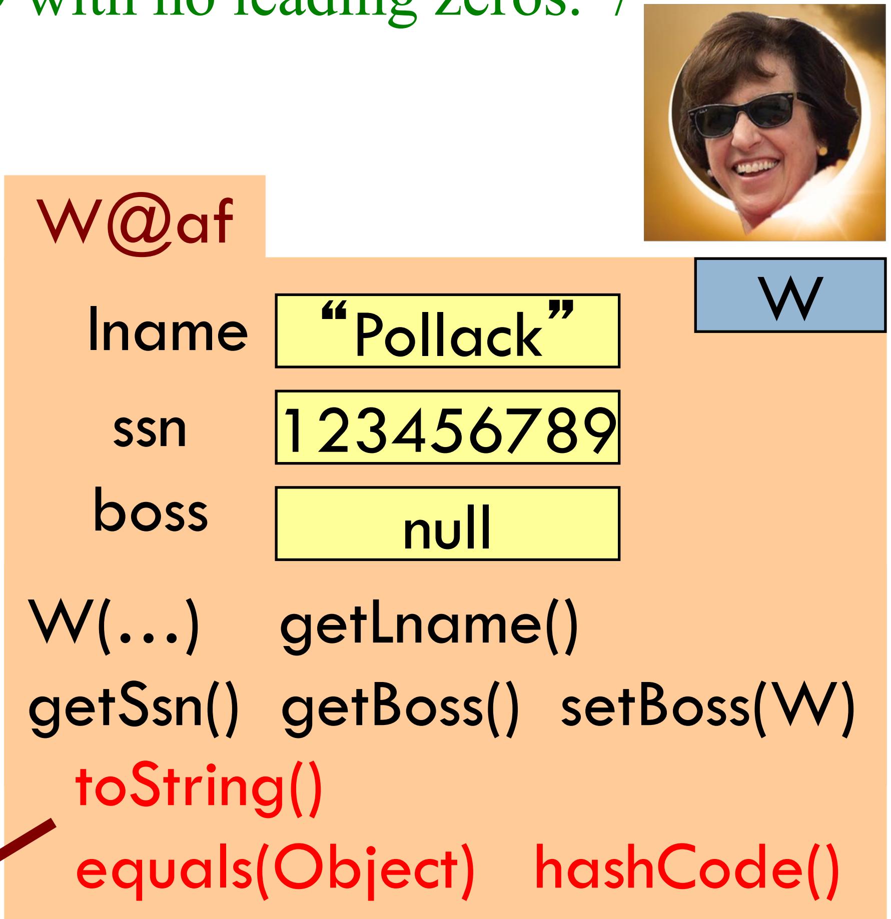
```
/** Constructor: worker with last name n,  SSN s, boss b (null if none).
    Prec:  n not null,  s in 0..999999999 with no leading zeros.*/
public W(String n, int s, W b)

/** = 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!

Class Object

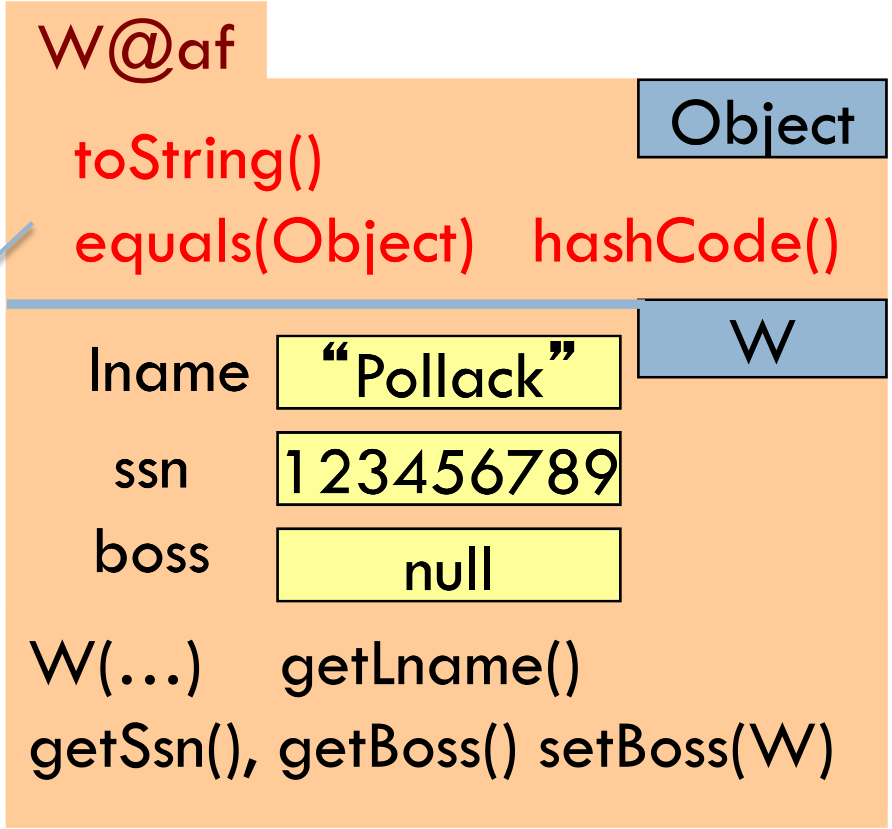
Java: Every class that does not extend another extends class Object. That is,

public class W {…}

is equivalent to

public class W **extends** Object {…}

We draw object like this:



W@af

Object

toString()

equals(Object) hashCode()

W

lname “Pollack”

ssn 123456789

boss null

W(…) getLname()

getSsn(), getBoss() setBoss(W)

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

Extends: "Is A"

- Extension should reflect **semantic data model:** meaning in real world
- A should extend B if and only if **A "is a" B**
  - 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

Extends: “Is A”

Which of the following seem like reasonable designs?

A. Triangle extends Shape { … }

B. PhDTester extends PhD { … }

C. BankAccount extends CheckingAccount { … }

Extends: “Is A”

Which of the following seem like reasonable designs?

A. Triangle extends Shape { … }
   A. Yes! A triangle is a kind of shape.

~~B. PhDTester extends PhD { … }~~
   A. No! A PhDTester “tests a” PhD, but itself is not a PhD.

~~C. BankAccount extends CheckingAccount { … }~~
   A. No! A checking account is a kind of bank account; we likely would prefer:

      CheckingAccount extends BankAccount { … }

Investigate: JFrame

1. How many levels deep is JFrame in the class hierarchy?
   - (Object is JFrame's super-super-…-superclass. How many supers are there?)

2. In which class is JFrame's getHeight() method defined?
   - (hint: it's not JFrame!)

What’s in a name?

The name of the object below is

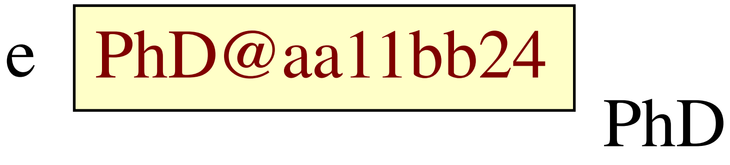
PhD@aa11bb24

The name is <class> @ <address in memory>.

Variable e, declared as

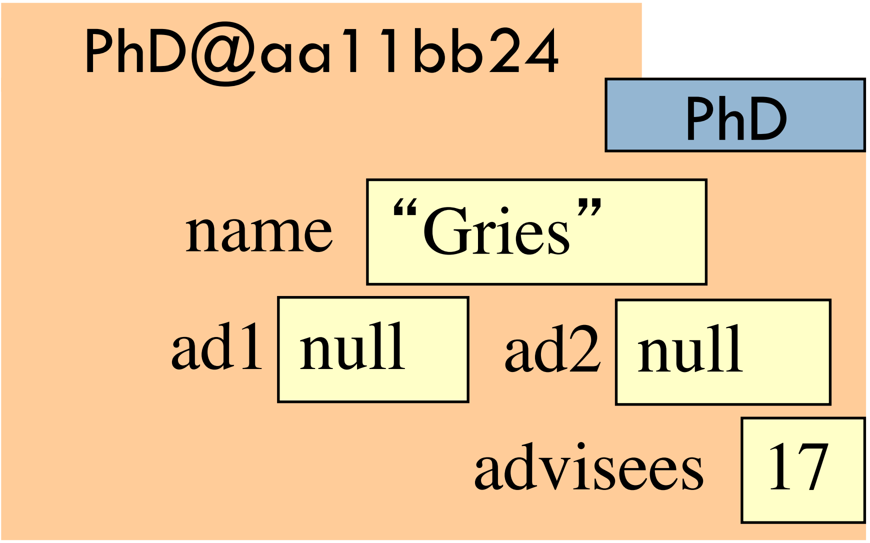
PhD e;

contains not the object but the name of the object (i.e., it is a reference to the object).



e PhD@aa11bb24

PhD



PhD@aa11bb24

PhD

name “Gries”

ad1 null

ad2 null

advisees 17

Method toString()

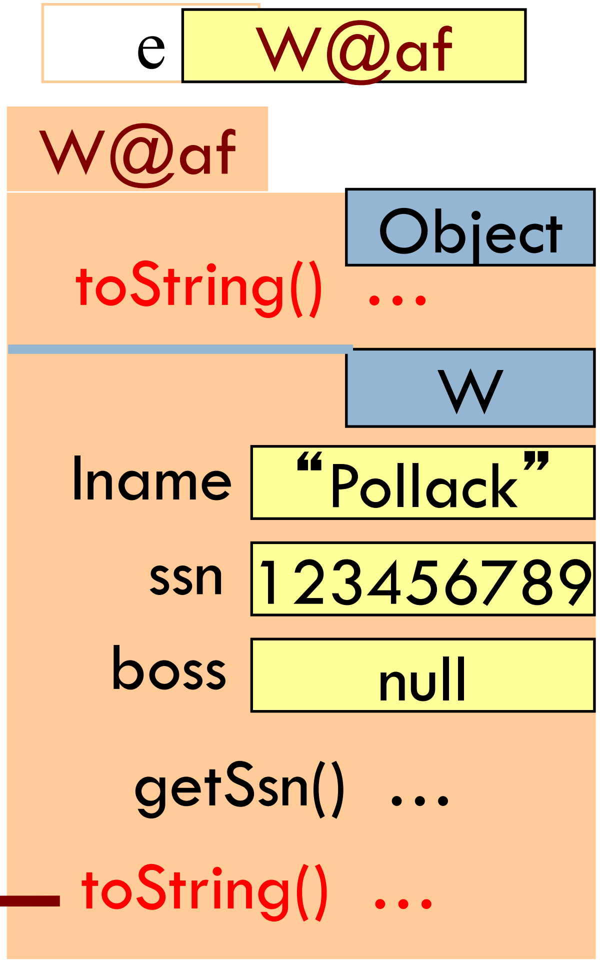
toString() in Object returns the name of the object: W@af

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
e   automatically does e.toString()

e.toString() calls this method



Method toString()

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

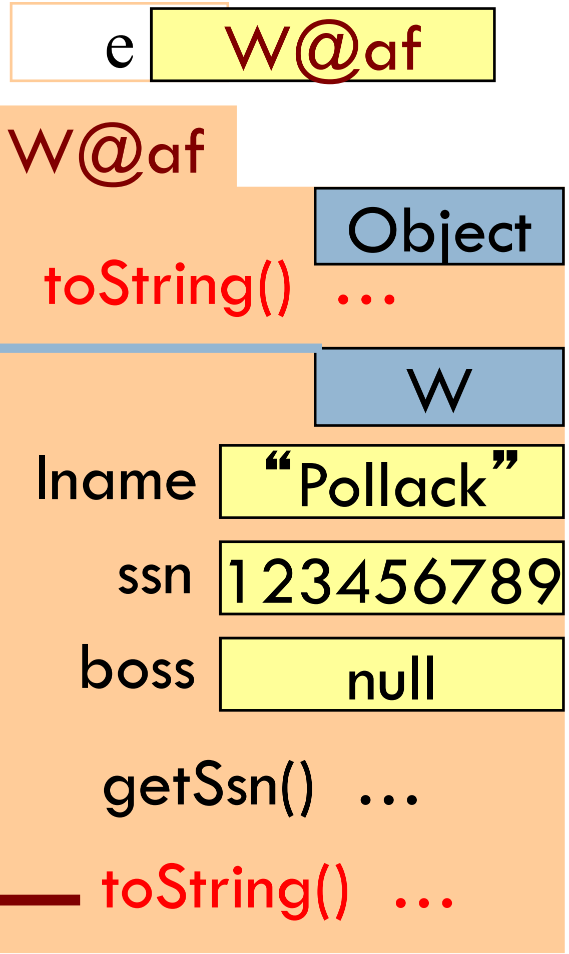
```
public class W {

  …

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

conditional expression

e.toString() calls this method

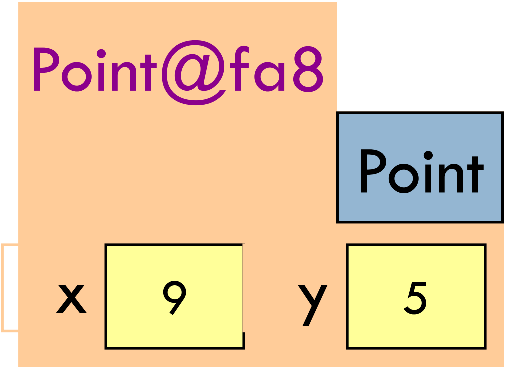


Another example of toString()

```
/** 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  + “)”;
    }
}
```

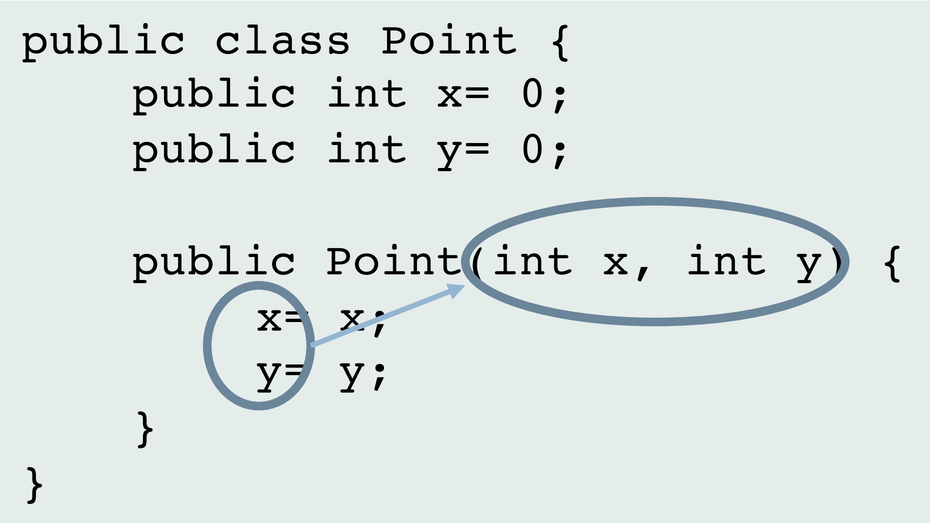


(9, 5)

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

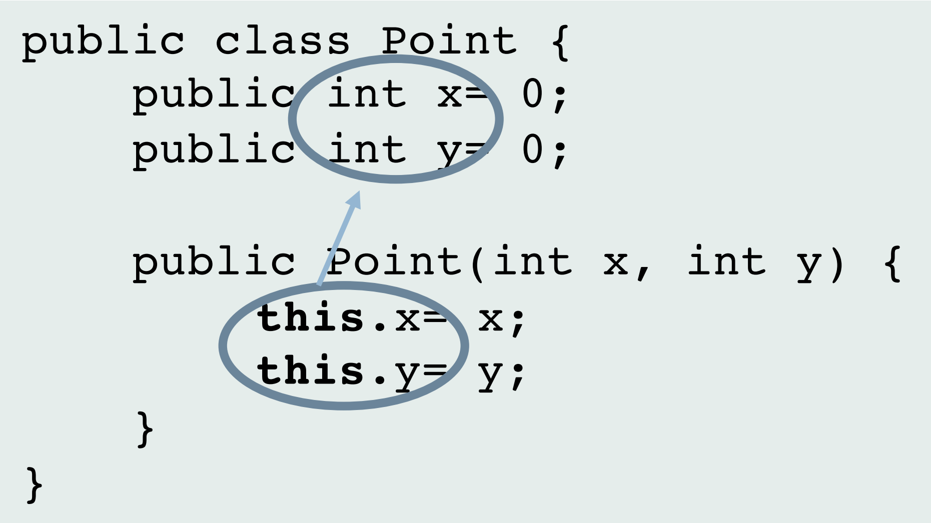
this: the object's own name

- **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
- Example: a shadowed class field



```
public class Point {
    public int x= 0;
    public int y= 0;

    public Point(int x, int y) {
        x= x;
        y= y;
    }
}
```



```
public class Point {
    public int x= 0;
    public int y= 0;

    public Point(int x, int y) {
        this.x= x;
        this.y= y;
    }
}
```

Static components

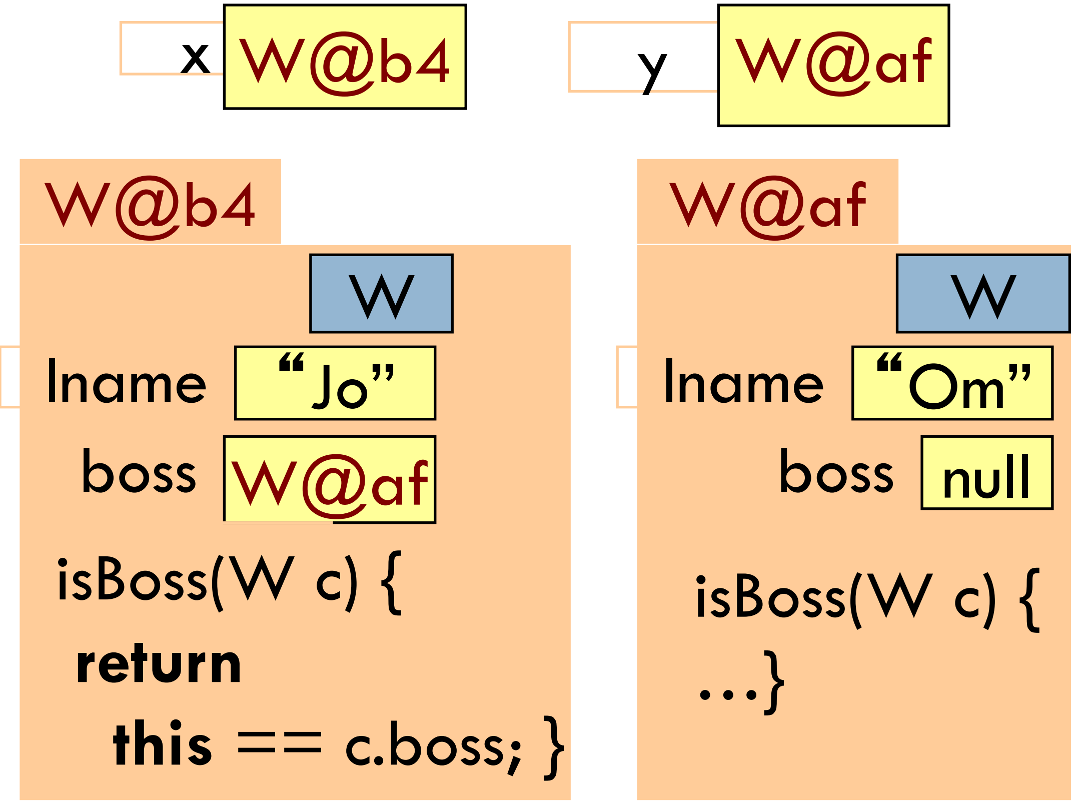
```
/** = “this object is c’ s boss”.
      Pre: c is not null. */
public boolean isBoss(W c) {
      return this == c.boss;
}
```

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

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

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 W@b4

y W@af

W@b4

W

lname “Jo”

boss W@af

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

W@af

W

lname “Om”

boss null

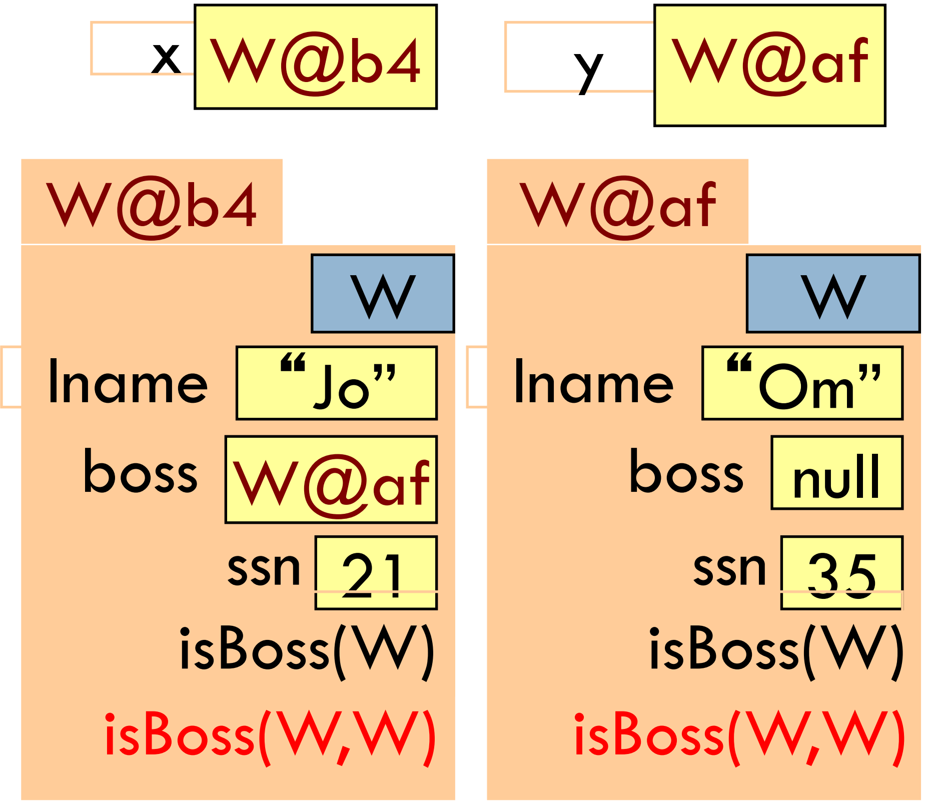
isBoss(W c) {
…}

Static components

```
/** = “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;
}
```



Static components

```
/** = "b is c' s boss".
    Pre: b and c are not null. */
public static boolean isBoss(W b, W c) {
    return b == c.getBoss();
}
```

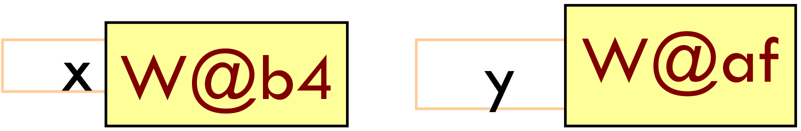
static: there is only one copy of the method. It is *not* in each object

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

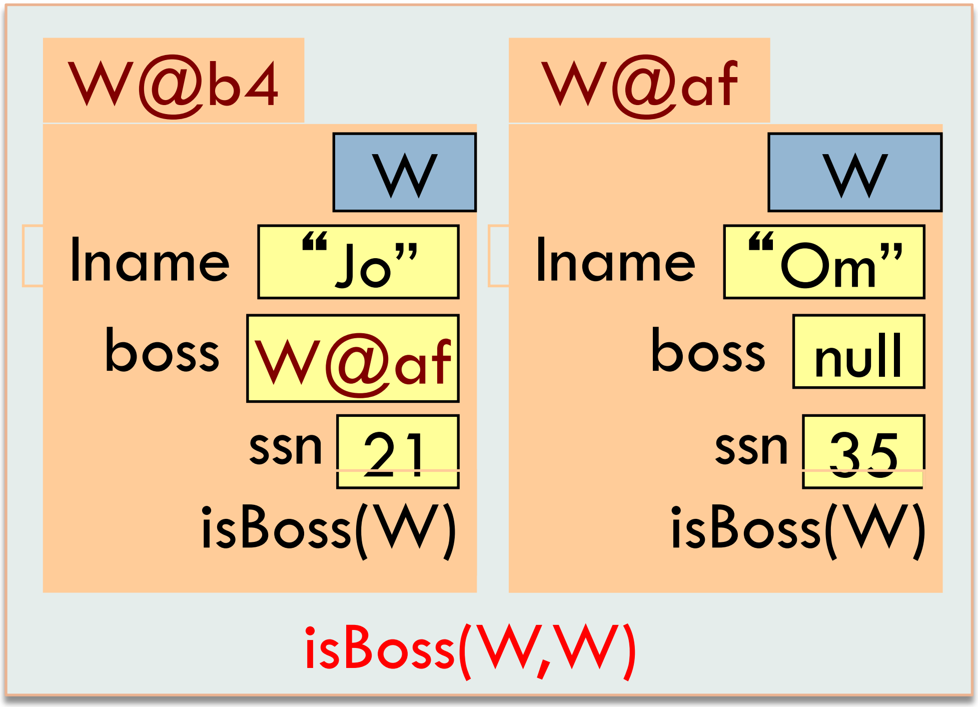
~~y.isBoss(x, y)~~

Preferred:

W.isBoss(x, y)



Box for W (objects, static components)



Good example of static methods

java.lang.Math

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

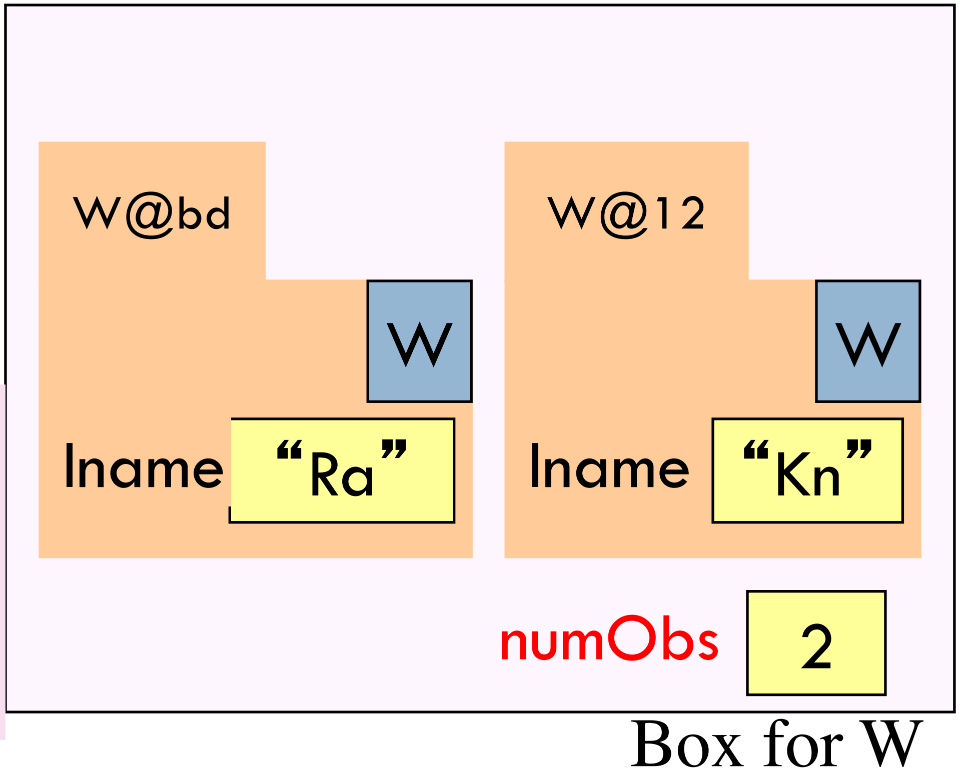
(Or find it by googling java.lang.Math 8)

A use for static fields (aka class variables): Maintain info about created objects

```
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.



Class java.awt.Color uses static fields

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= …;
…
```

Java application

Java application: a program 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

Uses of static fields:
Implement the Singleton pattern

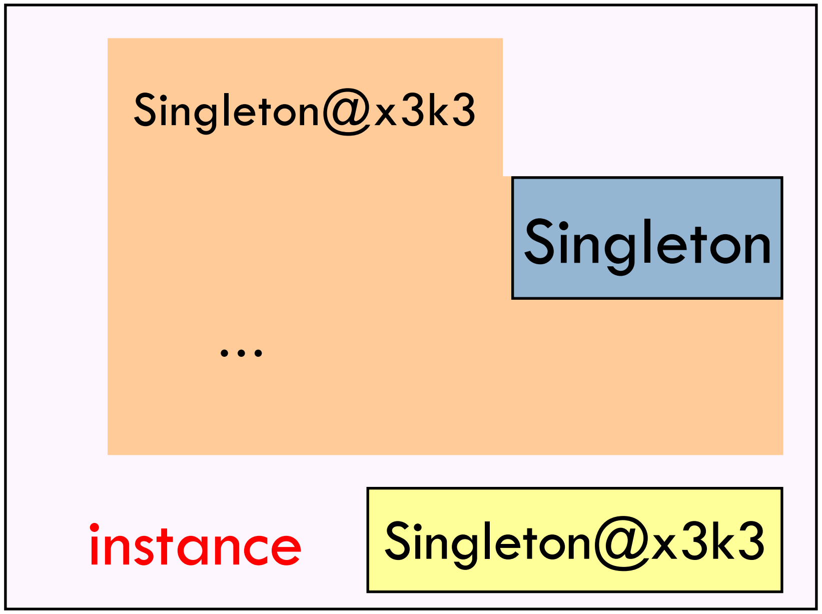
Only one Singleton can ever exist.

```
public class Singleton {
    private static final Singleton instance= new Singleton();

    private Singleton() { }  // ... constructor

    public static Singleton getInstance() {
        return instance;
    }

    // ... methods
}
```



Box for Singleton

Looking ahead: Recitation 3

- No prework! Concentrate on **A1** this weekend
- TA teaches testing; you test a class using Junit
- You can work in groups of up to 3; form a CMS group **before** submitting
- You will find faults in the class (fun!) and fix them
- Upload to CMS when done
  - Hopefully during recitation
  - If not, on/by Friday