CS1130 lecture 4   3 Feb 2012
Testing; class Object; toString; static variables/methods
the API package and its specification

Reading for this lecture: Testing with JUnit (Appendix I.2.4 & pp. 385—388),

class Object (pp. 153-154),

function toString (pp. 112-113),

static variables and methods (Sec. 1.5, p. 47).

Reading for next two lectures: Executing method calls, if-statements, the return statement in a function, local variables. Chapter 2 except 2.3.8 and 2.3.9.

This reading will some clarify some concepts, such as method parameters, that we have had to gloss over so far.

A1: first submission due **Mon, 6 Feb** on CMS.
Ignore "Extended Until" on CMS.

(We put in a fake extension to work around a CMS limitation.)

1

Testing —using JUnit

Bug: Error in a program. (Always expect them!)
Debugging: Process of finding bugs and removing them.
Testing: Process of analyzing, running program, looking for bugs.
Test case: A set of input values, together with the expected output.

Get in the habit of writing test cases for a method from the method's specification —even *before* writing the method's body.

```
/** = number of vowels in word w.
Precondition: w contains at least one letter and nothing but letters*/
public int numberOfVowels(String w) {
 // (nothing here yet!)
}
```

A feature called **JUnit** in DrJava helps us develop test cases and use them. You *have* to use this feature in assignment A1.   2

Here are two test cases

1. w1= **new** Worker("Obama", 1, **null**);
    Name should be: "Obama";   SSN: 1;   boss: **null**.

2. w2= **new** Worker("Biden", 2, w1);
    Name should be: "Biden";   SSN: 2;   boss: w1.

Need a way to run these test cases, to see whether the fields are set correctly. We could use the interactions pane, but then repeating the test is time-consuming.

To create a testing framework: select menu **File** item **new JUnit test case…**. At prompt, put in class name **WorkerTester**. This creates a new class with that name. Save it in same directory as class Worker.

The class imports **junit.framework.TestCase**, which provides some methods for testing.

3

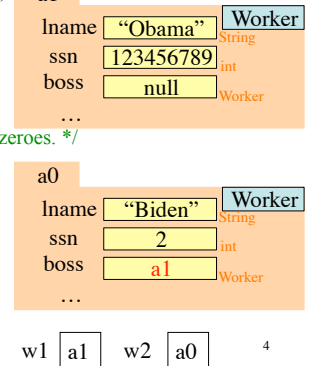**Spec, headers for methods in class Worker**
/** Constructor: a worker with last name n ("" if none), SSN s, and boss b (null if none).
     Precondition: n is not null, s in 0..999999999 with no leading zeros.*/
public Worker(String n, int s, Worker b)

/** = worker's last name */
public String getLname()

/** = last 4 SSN digits without leading zeroes. */
public int getSsn()

/** = worker's boss (null if none) */
public Worker getBoss()

/** Set boss to b */
public void setBoss(Worker b)

4

Testing the constructor (also getter methods)

File->new JUnit test case … [save in same directory as WorkerTester.java]
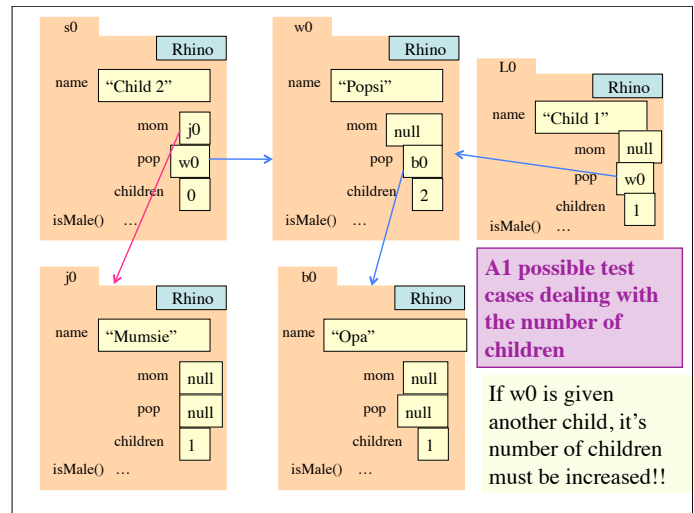
```
/** Test constructor (and getters) */
public void testConstructor() {
    Worker w1= new Worker("Obama",
123456789, null);
    assertEquals("Obama", w1.getLname());
    assertEquals(6789, w1.getSSN4());
    assertEquals(null, w1.getBoss());

    Worker w2= new Worker("Biden", 2, w1);
    assertEquals("Biden", w2.getLname());
    assertEquals(2, w2.getSSN4());
    assertEquals(w1, w2.getBoss());
}
```

assertEquals(x, y):

test whether **x** (*expected*) equals **y** (*computed*); print error msg. and stop execution if not equal.

Pg 488 lists other methods that can be used.

Constructor: test ALL fields to be sure class invariant is true.

Click button Test in DrJava to call all "testX methods".   5

A1 possible test cases dealing with the number of children

If w0 is given another child, it's number of children must be increased!!

Class Object: The superest class of them all

A minor mystery: since Worker doesn't extend anything, it seems that it should have only the methods we wrote for it. *But it has some other methods, too.*

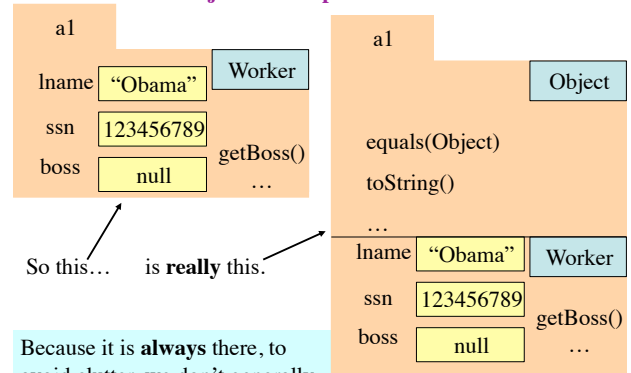**Java feature**: Every class that does not extend another one automatically extends class Object. That is,

public class C { … }

is equivalent to

public class C **extends** Object { …}

7

Class Object: The superest class of them all

So this… is **really** this.

Because it is **always** there, to avoid clutter, we don't generally draw the partition for superclass Object. (A2 will be an exception).
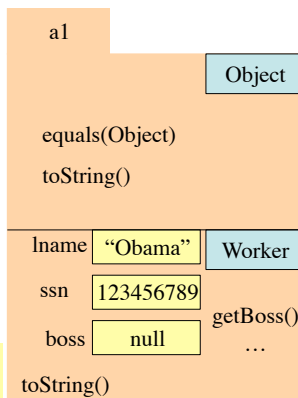
8

Method toString()

Convention: c.toString() returns a representation of folder c, giving info about the values in its fields.

Put following method in Worker.

```
/** = representation of this Worker
* [etc., see full program] */
public String toString() {
    return …;
}
```

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

9

Another example of toString()

```
/** An instance represents a point (x, y) in the plane */
public class Point {
    private int x;  // the x-coordinate
    private int y;  // the y-coordinate

    /** Constructor: An instance for point (xx, yy) */
    public Point(int xx, int yy) {
        …
    }

    /** = a representation of this point in form "(x, y)" */
    public String toString() {
        return …;
    }
}
```

(getter and setter methods not given on this slide)

Fill these in

Example: "(3, 5)"

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

10

A **static method** appears not in each folder but only once, in *the file drawer.*

Make a method static if it doesn't need to be in a folder because it doesn't reference the contents of the "containing" folder.

```
/** = "this object is the c's boss".
    Precondition: c is not null. */
public boolean isBoss(Worker c) {
    return this == c.getBoss();
}
```

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

```
/** = "b is c's boss".
    Precondition: b and c are not null. */
public  static  boolean isBoss(Worker b, Worker c) {
    return b == c.getBoss();
}
```
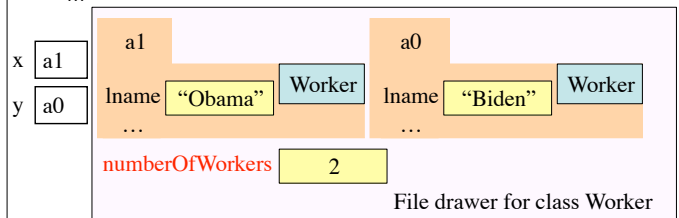
11

A **static** variable appears not in each folder but as a *single entity in the file drawer.* It can be used to maintain information about all the folders.

Declaration: (goes inside class definition, just like field declarations)
    private static int numberOfWorkers; // no. of Worker objects created
    …

numberOfWorkers 2

File drawer for class Worker

Class, not var holding folder name

Reference the variable by Worker.numberOfWorkers.

12