CS1110 lecture 5 8 Feb 2010
Testing; class Object; toString; static variables/methods

Keep your iClickers and a sheet of paper out.

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).

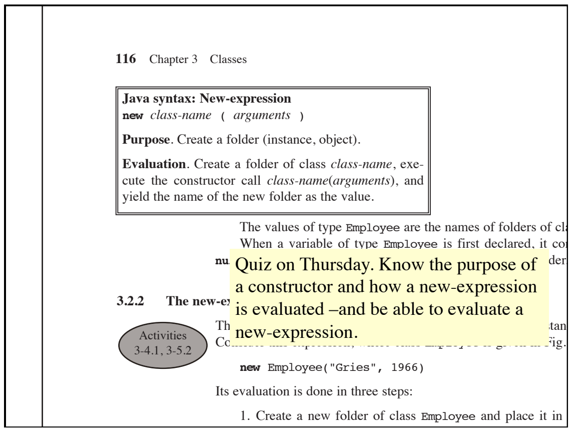
Reading for next two lectures: Executing method calls, if-statements, the return statement in a function, local variables. Chapter 2 except 2.3.8 and 2.3.9.

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

A1: due Sat, Sept 12, on CMS; form groups by Wed.
Ignore "Extended Until" on CMS

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

1



116 Chapter 3 Classes

Java syntax: New-expression
new *class-name* (*arguments*)

Purpose. Create a folder (instance, object).

Evaluation. Create a folder of class *class-name*, execute the constructor call *class-name*(*arguments*), and yield the name of the new folder as the value.

The values of type Employee are the names of folders of cl
When a variable of type Employee is first declared, it co
nu der

3.2.2 The new-ex

Activities 3-4.1, 3-5.2

Th tan
Co Fig.

```
new Employee("Gries", 1966)
```

Its evaluation is done in three steps:

1. Create a new folder of class Employee and place it in

Quiz on Thursday. Know the purpose of a constructor and how a new-expression is evaluated –and be able to evaluate a new-expression.

Organizing and streamlining the testing process

Testing: Process of analyzing, running program, looking for bugs (errors).

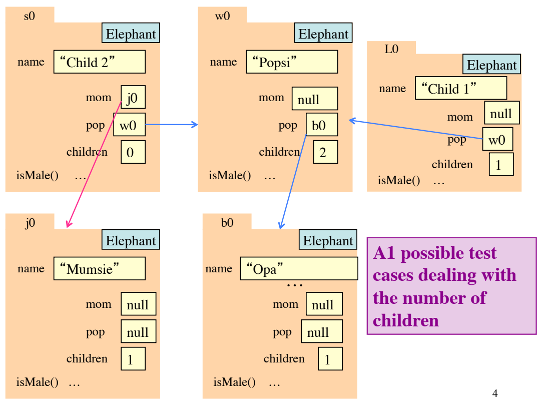
Test case: A set of input values, together with the expected output.

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

```
/** = 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!)
}
```

Developing test cases first, in "critique" mode, can prevent wasted work.

3

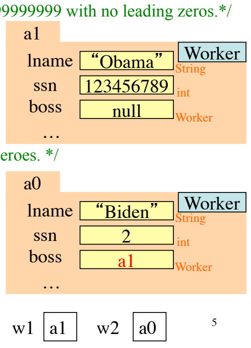


A1 possible test cases dealing with the number of children

4

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)
```



5

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 mess. and stop execution if they are not equal.

Pg 488 lists some other methods that can be used.

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

6

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.*

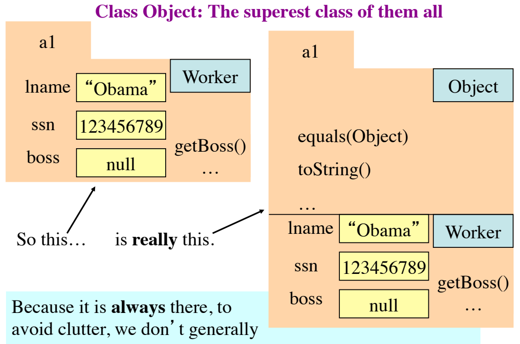
Java feature: Every class that does not extend another one automatically extends class Object. That is,

```
public class C { … }
```

is equivalent to

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

Class Object: The superest class of them all



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

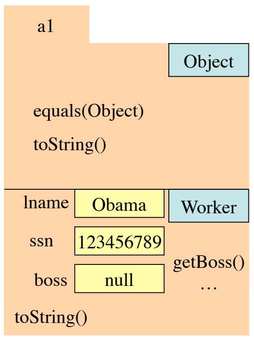
Method toString()

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()



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 are 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.

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.

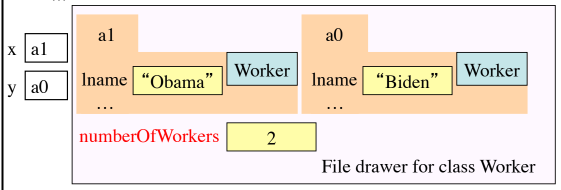
```
/** = "this object is the c's boss".
      Precondition: c is not null. */
public boolean isBoss(Worker c) {
    return this == c.getBoss();
}

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

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

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
…
```



Class, not var holding folder name

Reference the variable by Worker.numberOfWorkers.