CS1110 lecture 5 14 Sept 2010

Testing; the 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), the 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've had to gloss over so far.

A1 (still) due Saturday Sept 18 on CMS; group yourselves by Wed. Ignore "Extended Until" on CMS

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

Email re: lab 03, quiz 2, etc. was sent on Saturday. Bouncing emails: cabooserwar, blacktora4546, jfk54, tariq.mozaini, lukeg432, dc.mcmurtry10, khyjhcho.

Organizing and streamlining the testing process

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

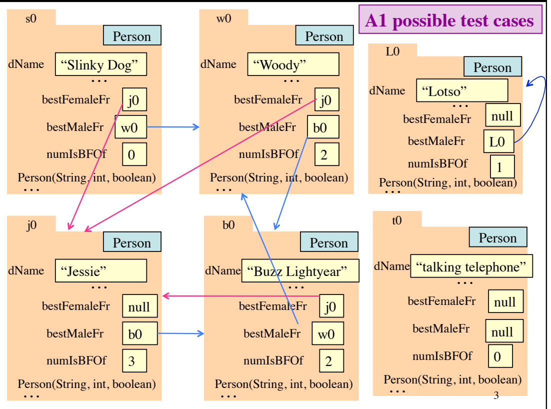
Test case: A set of input values, together with the expected output.

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!)
}
```

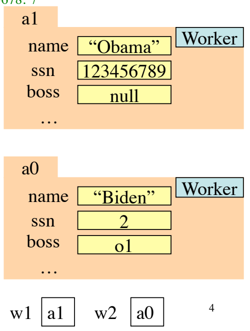
Test cases "need", "rhythm" reveal vagueness in the spec!
Developing test cases first, in "critique" mode, can prevent wasted work.

A1 possible test cases



Specifications and headers for methods in class Worker, plus test cases

```
/** 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,
 * so SSN 012-34-5678 should be given as 12345678.*/
public Worker(String n, int s, Worker b)

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

/** Set worker's last name to n ("" if none).
     * Precondition: n is not null.*/
public void setName(String n)

/** = last 4 SSN digits without leading zeroes. */
public int getSSN4()

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

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



A testMethod to test a constructor (and getter methods)

File->new JUnit test case … [save in same directory as Worker.java] imports junit.framework.TestCase, with key methods.

```
/** Test constructor and getters*/
public void testConstructor() {
 Worker w1= new Worker("Obama", 123456789, null);
  assertEquals("Obama", w1.getName());
  assertEquals(6789, w1.getSSN4());
  assertEquals(null, w1.getBoss());

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

 // should test last name "", too
```

first test case

second test case

assertEquals(x,y):

test whether **x** *(expected)* equals **y** *(computed)*; print an error message and stop the method if they are not equal.

Pg 488 lists some other methods that can be used.

Every time you click button Test in DrJava, all "testX methods" are called, in some order.

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.*

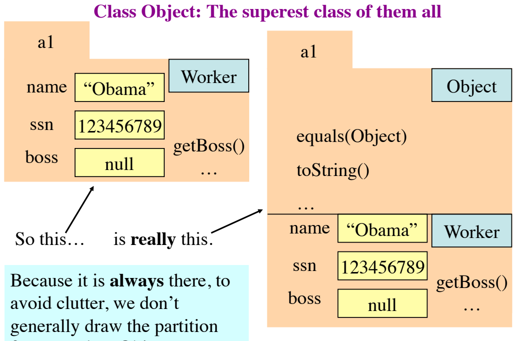
Java feature: Every class that does not extend another one automatically extends class Object. That is,

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.

When a String is expected (or in the Interactions pane), the expression c is evaluated as c.toString() .

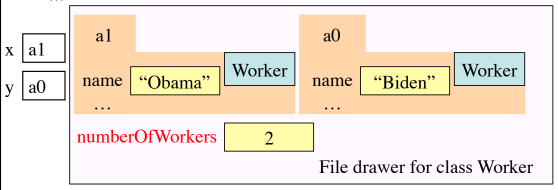
```
/** = e.g., "Obama, XXX-XX-6789, boss null",
        "Biden, XXX-XX-2, boss Obama" [see posted code for full spec]*/
public String toString() {return ??; } // bad scrunched style, slides are tiny☹
```

(A) "lname" + ", XXX-XX-" + "getSSN4()" + ", boss " + "boss.getName()"
(B) lname + ", XXX-XX-" + getSSN4() + ", boss " + boss.getName()
(C) getName() + ", XXX-XX-" + getSSN4() + ", boss " + getBoss().getName()
(D) lname + ", XXX-XX-" + getSSN4() + ", boss " + getBoss()
(E) none of the above

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 multiple folders.

Declaration:

```
private static int numberOfWorkers; // no. of Worker objects created
…
```



Class, not var holding folder name

Reference the variable by Worker.numberOfWorkers.

A **static** method is also in the drawer, not in individual folders.

Make a method static if it doesn't need to be in all the folders because:
it wouldn't reference the contents of the "containing" folder;
equivalently, its actions/results would be the exactly the same no matter which folder it were in.

getName() should not be static, but the following should:

```
/** = "there are more than 100 workers". */
public static boolean moreThan100() {
    return numberOfWorkers > 100;
}
```

Class, not var holding a folder name

Sample call: Worker.moreThan100()

Should the following function be static?
(A) yes (B) no

```
/** = "a and b are not null and the last 4 digits of their SSNs
        are the same". */
/* (This notation means that the function yields
 * the truth value of the quoted statement.
 * So, a and b are allowed to be null!) */
public [static??] boolean clashingSSNs(Worker a, Worker b) {
   return a != null && b != null
        && a.getSSN4() == b.getSSN4();
}
```