CS1110 4 February. Customizing a class & testing

- Fields; getter & setter methods. Secs 1.4.1 (p. 45) & 3.1 (pp. 105–110 only)
- Constructors. Sec. 3.1.3 (p. 111–112)
- Testing methods. Appendix I.2.4 (p. 486)

Next time:

Testing using JUnit.

Object: the superest class of them all. pp 153–154.

Function toString.

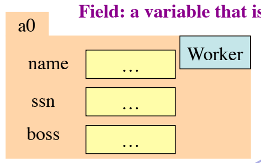
Static components Sec. 1.5 (p. 47).

Quiz 2 on Tuesday 9 February
Purpose of a constructor (slide 5)
Evaluating a new expression (slide 6)

Assignment A1 out, due Friday 13 February
Writing and testing a class definition

Labs and one-on-ones (schedule yours on CMS) will help you with it.

Field: a variable that is in each folder of a class.



Declarations of fields

```
/** An instance is a worker in a certain organization. */
public class Worker {
   private String name; // Last name (null if unknown/none)
   private int ssn; // Social security #: in 0..999999999
   private Worker boss; // Immediate boss (null if none)
}
```

Usually, fields are **private**, so methods that are outside the class can't reference them. Slightly confusing: you *can* access them in the DrJava interactions pane if preferences are set appropriately.

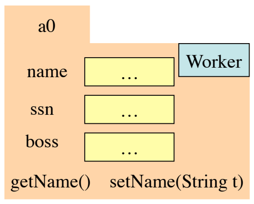
Getter and setter methods

In the definition of Worker
(we post our code on the website):

```
/** = worker's last name*/
public String getName() {
   return name;
}

/** Set worker's last name to n */
public void setName(String n ) {
   name= n;
}
```

/** = last 4 SSN digits, as an int*/
(Try writing it yourself.
Should there also be a setter? What about for boss?)



Getter methods (functions) **get** or retrieve values from a folder.

Setter methods (procedures) **set** or change fields of a folder

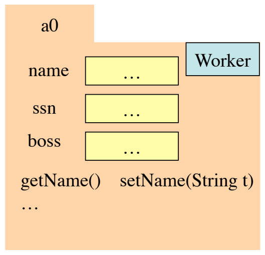
Initialize fields when a folder is first created

We would like to be able to use something like

new Worker("Obama", 1, **null**)

to create a new Worker, set the last name to "Obama", the SSN to 000000001, and the boss to **null**.

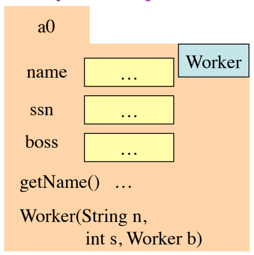
For this, we use a new kind of method, the **constructor**.



Purpose of a constructor: To initialize (some) fields of a newly created object

In the class definition of Worker:

```
/** Constructor: an instance with last
name n, SSN s (an int in 0..999999999,
and boss b (null if none) */
public  Worker(String n, int s,
               Worker b) {
   name= n;
   ssn= s;
   boss= b;
}
```



The name of a constructor: the name of the class.

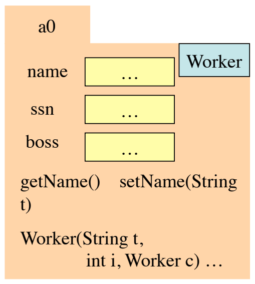
Do not put a type or **void** here

New description of evaluation of a new-expression

new Worker("Obama", 1, **null**)

1. Create a new folder of class Worker, with fields initialized to default values (e.g. 0 for **int**) –of course, put the folder in the file drawer.
2. Execute the constructor call

   Worker("Obama", 1, null)
3. Use the name of the new object as the value of the new-expression.



Memorize this new definition! Today! Now!

Testing —using JUnit

Bug: Error in a program.

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

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

Debugging: Process of finding a bug and removing it.

Get in the habit of writing test cases for a method from the method's specification --- even before you write the method's body.

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

7

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.

Here are two test cases

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.

8

```
/** A JUnit test case class.
 * Every method starting with "test" will be called when running
 * the test with JUnit. */
public class WorkerTester extends TestCase {

    /**  A test method.
     * (Replace "X" with a name describing the test.  Write as
     * many "testSomething" methods in this class as you wish,
     * and each one will be called when testing.) */
    public void testX() {
    }
}
```

One method you can use in testX is

assertEquals(x,y)

which tests whether expected value x equals y

9

A testMethod to test constructor (and getter methods)

```
/** Test first constructor and getter methods getName,
    getSSN4, and getBoss*/
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());
    }
```

first test case

second test case

assertEquals(x,y):

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

x: expected value,
y: actual value.

A few other methods that can be used are listed on page 488.

Every time you click button Test in DrJava, this method (and all other testX methods) will be called.

10