CS1110 lecture 4 8 Sept. Customizing a class & testing

- Classes: fields; getter & setter methods. Secs 1.4.2 (p. 45) & 3.1 (pp. 105–110 only)
- Constructors. Sec. 3.1.3 (p. 111–112)
- Testing methods. Appendix I.2.4 (p. 486)

Organizational tip #652:
For classes with a lot of handouts (like CS1110), get a 3-ring binder and a 3-hole punch. Punch holes in the handouts and store them in the binder; this makes accessing them *much* easier. You can easily interleave other notes and papers, too.

Next time:

More testing using JUnit.

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

Function toString (pg. 112).

Static components Sec. 1.5 (p. 47).

A "must see" about academic integrity (on youtube):
http://tinyurl.com/35ltf4n

Quiz 2 on Tuesday 13 Sept
Purpose of a constructor (slide 6); Evaluating a new expression (slide 8)

Assignment A1 out today, due Saturday 17 September on the CMS.
Submit A1 earlier if you can so that we can start the iterative feedback process going.

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

Collaboration rules for assignment A1

- **Work alone or with *one* partner** —partners "group themselves" on the CMS *well before* submission; only one person submits the files.

Partners must do the work together, sit next to each other, with each taking turns "driving" (handling the mouse and keyboard). It is against the rules for one partner to develop code and later show it to the other.

- **Never** look at someone else's code or show yours to someone else.
- **Never** be in possession of someone else's code (except your partner).

One-on-One Sessions (optional)

Next 1.5 weeks, we are holding 1/2-hour one-on-one sessions on a computer with each student in CS1110.

Purpose: Help you develop a class as preparation for A1, give you a chance to ask questions. Not counted in final course grade.

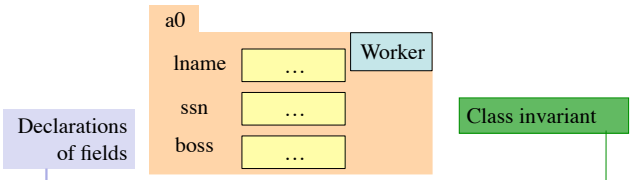
Sign up on the course CMS (http://cms.csuglab.cornell.edu): Click on assignment One-on-one, find the schedule of times/instructors. Choose one.

Bring to the 1-on-1: the book; laptop w/ DrJava if you have one.

Students with little or no programming experience report that these sessions are extremely helpful!

Office hours: http://www.cs.cornell.edu/courses/cs1110/2011sp/staff.html
Gries, Marschner (Tu/Th 10:10-10:55 Hollister 202)
Consultants and TAs: see web page

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




```
/** An instance is a worker in a certain organization. */
public class Worker {
    private String lname; // Last name ("" if none; never null)
    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.

(Exception: private fields *are* accessible in the DrJava Interactions pane)

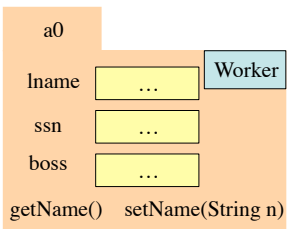
Getter and setter methods

In the definition of Worker (full code on the website):

```
/** = worker's last name*/
public String getName() {
    return lname;
}
/** Set worker's last name to n
    (can't be null, can be "")*/
public void setName(String n) {
    lname= 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

Initializing fields of a new instance (folder)

Creating a new Worker is now a multi-step process:

```
Worker w = new Worker();
w.setName("Obama");
…
```

← Ack! w.lname is **null** —contradicts class invariant!

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
This initialization should make the class invariant true.

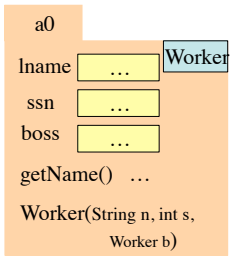
Example constructor

In the class definition of Worker:

```
/** Constructor: an instance with last name n
(can't be null, can be ""), SSN s (an int in
0..999999999), and boss b (null if none) */
public  Worker(String n, int s, Worker  b) {
    lname= n;
    ssn= s;
    boss= b;
}
```

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

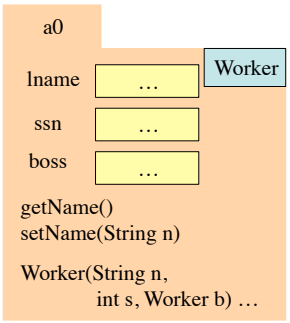
Do not put a type or **void** here



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 new definition! Today! Now!

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.

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

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.

Test case template created by DrJava

```
/** 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 *computed* value y.

A testMethod to test constructor (and getter methods)

```
/** Test first constructor (and getter methods getName,
    getSSN4, and getBoss) */
public void testConstructor() {
```

first test case

```
    Worker w1= new Worker("Obama", 123456789, null);
    assertEquals("Obama", w1.getName(), );
    assertEquals(6789, w1.getSSN4());
    assertEquals(null, w1.getBoss());
```

second test case

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

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

assertEquals(x,y):

Test whether **x** (*expected*) equals **y** (*computed*). If they are not equal, print an error message and stop the method.

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