CS1110  10 Sept. 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)

Quiz 2 on Tuesday 15 Sept
Purpose of a constructor (slide 5)
Evaluating a new expression (slide 6)

• 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).

Assignment A1 out, due Friday 18 Sept
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.

Worker name ...
ssn ...
boss ...

Next time:
Testing using JUnit.
Object: the superest class of them all. pp 153–154.
Function toString.
Static components Sec. 1.5 (p. 47).

/** 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)
}

We generally make fields private instead of public, so methods that are outside the class can’t reference them.
(Slightly confusingly at first, you can access them in the DrJava interactions pane.)

public class Worker {
    /** An instance is a worker in a certain organization. */
    public Worker(String n, int s, Worker b) {
        name=n;
        ssn=s;
        boss=b;
    }
    public void setName(String t) {
        name=t;
    }
    public String getName() {
        return name;
    }
    public int getSSN() {
        return ssn;
    }
}

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.

The name of a constructor: the name of the class.
Do not put a type or void here

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.

1. \( w_1 = \text{new Worker("Obama", 1, null); } \)
   Name should be: “Obama”; SSN: 1; boss: null.

2. \( w_2 = \text{new Worker("Biden", 2, w_1); } \)
   Name should be: “Biden”; SSN: 2; boss: w_1.

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.

```java
/** 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

  }

  public void testConstructor() {

    Worker w1 = new Worker("Obama", 123456789, null);
    assertEquals("Obama", w1.getName());
    assertEquals(123456789, 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());

  }

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

  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.
```