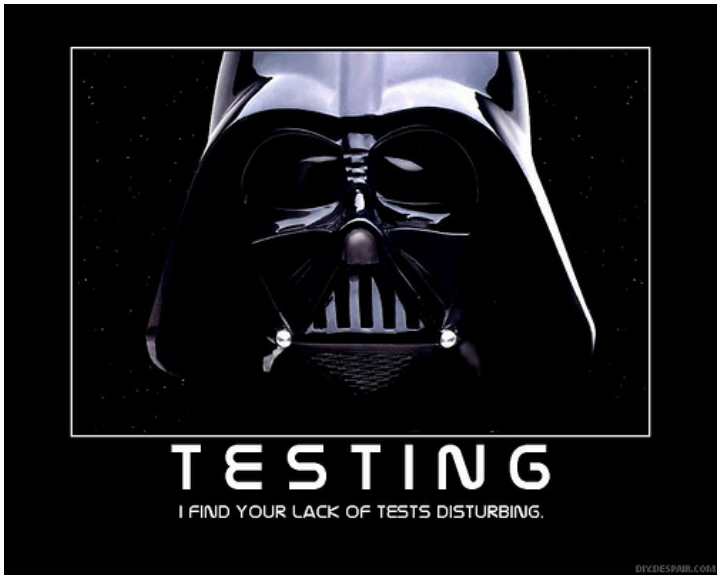
Lecture 5

Specifications & Testing

Important For This Lecture

Readings

- Testing with Junit
  - Appendix I.2.4
- Function toString
  - pp. 112—113



Announcements

- Assignment 1 is live
  - Posted on web page
  - Due Tuesday, Feb. 14th
- 1-on-1s for next 2 weeks
  - Slots still available
  - Schedule through CMS
- Recall Lab Schedules
  - 12:20-2:15 in ACCEL
  - 2:30-4:25 in Phillips 318

Quiz

- Get out a blank piece of paper.
- Write your LAST name, FIRST name, and Cornell NetID (not your Cornell ID. My NetID is wmw2)
- **Write down the purpose of a constructor.**
- **Write down the steps in evaluating the new expression** in the following assignment statement (you should not need to know what the class definition looks like):

```
t = new Book("Truth is All", 163845);
```

- Please be literal-minded and to write precise statements.

Public vs. Private

- Recall our convention
  - Fields are private
  - Everything else public
- Private means "hidden"
  - Public fields can be accessed **directly**
- **But this is a bad idea!**
  - Cannot control how other programmers use them
  - They might violate our **invariants** (and get bugs)

```
public class PublicPoint3d {
  public double x;
  public double y;
  public double z;
}
```

- Type in Interactions Pane:
  - > PublicPoint3d p = new PublicPoint3d();
  - > p.x = 3.0;
  - > p.x
- No need for getters/setters

Public vs. Private

- Recall our convention
  - Fields are private
  - Everything else public
- Priv
  - P
    a
- **But**
  - Cannot control how other programmers use them
  - They might violate our **invariants** (and get bugs)

```
public class PublicPoint3d {
  public double x;
  public double y;
  public double z;
```

Invariants must always be true. **Always.**

The Role of Getters and Setters

- Make sure that the invariants are true

> p.x = 3.0;

> p.x

- No need for getters/setters

Aside: Private is a Class Property!

- Private means hidden to objects of other classes!
  - Does not apply to two objects of same class
  - Methods can access fields in object of same class
- Example: Point distance
- Useful in Assignment 1
  - **Hint**: What field does not have getters or setters?

```
public class Point3d {
  private double x;
  private double y;
  private double z;
  …
  /** Yields: Distance to q */
  public double
     distanceTo(Point3d q) {
       return Math.sqrt(
           (x-q.x)*(x-q.x)+
           (y-q.y)*(y-q.y)+
           (z-q.z)*(z-q.z));
     }
}
```

We Write Programs to Do Things

Memorize These!

Write them down several times.

- Methods are the **key doers**

Method Definition

- Defines what method **does**

```
public void setName(String n) {
    lname= n;
}
```

Method **Body** (inside {})

declaration of **parameter** n

Method **Header**

Method Call

- Command to **do** the method

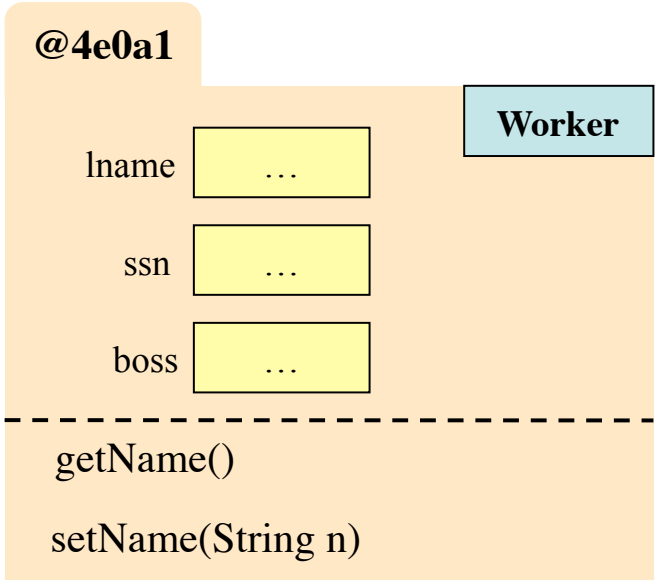
```
var.setName("Bob");
```

argument to assign to n

- **Parameter**: variable that is declared within the parentheses of a method header.
- **Argument**: a value to assign to the method parameter when it is called

Invariants vs. Preconditions

- Both are properties that **must be true**
  - **Invariant**: Property of a field
  - **Precondition**: Property of a method parameter
- Preconditions are a way to "**pass the buck**"
  - Responsibility of the method call, not method definition
  - How you will "enforce" in




```
/** Set worker's last name to n
 * Precondition: n cannot be null
or "Bob"
 */
public void setName(String n) {
        lname = n;
}
```

- Recall lname invariant
- Precondition ensures invariant is true

toString(): A Very Special Method

- We use interactions pane to see object "tab name"
- Interactions pane is really showing off a string
  - String that represents object
  - By default: **the tab name**
- But we can change this!
  - Add toString() to your class
  - That String will be used in place of the tab name
- Will see usage later

```
public class Point3d {
  …
  /**Yields: String (x,y,z)*/
  public String toString() {
    return "("+x+","+y+","
              +z+")";
  }
}
```

- Type in Interactions Pane:
  - > Point3d p = new Point3d();
  - > p
- Remove toString() & repeat

toString(): A Very Special Method

- We use interactions par
  to see object "tab name
- **Without** toString():
  Point3d@7b751a9e
- **With** toString():
  (0,0,0)
  - That String will be used in place of the tab name
- Will see usage later

Assigned by Java
Always Different

```
Point3d {

String (x,y,z)*/
ing toString() {
  return "("+x+","+y+","
         +z+")";
 }
}
```

- Type in Interactions Pane:
  > Point3d p = new Point3d();
  > p
- Remove toString() & repeat

Specifications for Methods in Worker

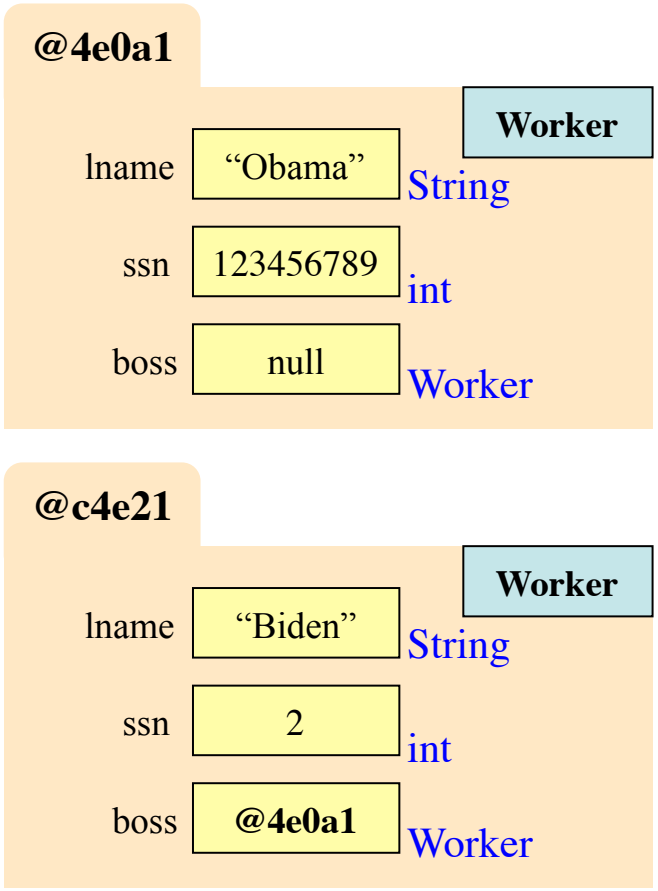
```
/** Constructor: a worker with last name n
 * ("" if none), SSN s, and boss b (null if none).
 * Precondition: n is not null,  s in
 * 0..999999999 with no leading zeros.*/
public Worker(String n, int s, Worker b)

/** Yields: worker's last name */
public String getLname()

/** Yields: last 4 SSN digits w/o leading zeroes. */
public int getSSN()

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

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



Test Cases: Finding Errors

- **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.

```
/** Yields: number of vowels in word w.
 * Precondition: w contains at least one letter and only letters */
public int numberOfVowels(String w) {
  // (nothing here yet!)
}
```

Test Cases: Finding Errors

- **Bug**: Error in a program. (Always
- **Debugging**: Process of finding bug
- **Testing**: Process of analyzing, runn
- **Test case**: A set of input values, to

Get in the habit of writing test case
specification —even *before* writing

Some Test Cases

- numberOfVowels("Bob")
  Answer should be 1
- numberOfVowels("Aeiuo")
  Answer should be 5
- numberOfVowels("Grrr")
  Answer should be 0

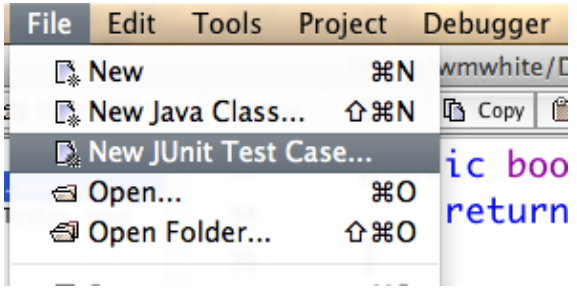
```
/** Yields: number of vowels in word w.
 * Precondition: w contains at least one letter and only letters */
public int numberOfVowels(String w) {
  // (nothing here yet!)
}
```

Test Cases for a Constructor in Worker

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
- Could use interactions pane, but this is time-consuming.

- To create a testing framework
  - Select menu File item new Junit Test Case….
  - At prompt, put in class name WorkerTester
  - Save it in same directory as class Worker



- This imports junit.framework.TestCase; has tools 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)

- It tests whether expected value x equals computed value y.

Method to Test Constructor (& 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());
```

2nd 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):

- Tests if x (expected) equals y (computed)
- If they are not equal, print an error message & stops
- Other testing procedures on p. 488 of the text

Special: called w/o object