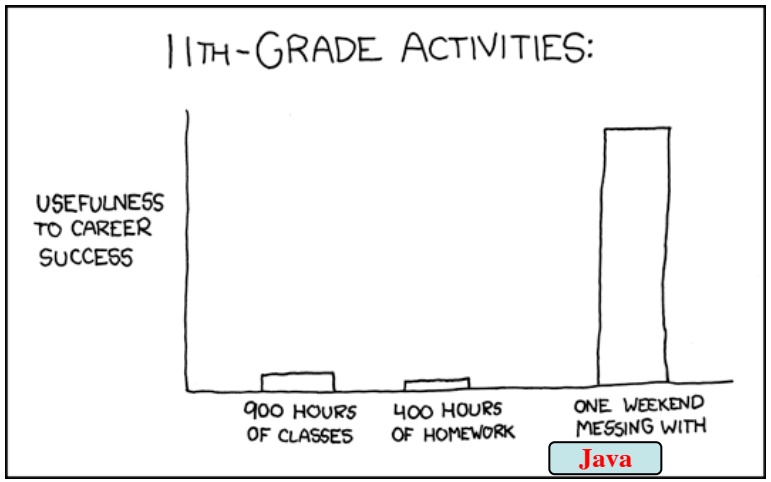
Lecture 2

Classes

Announcements for the Class

Readings

- Section 1.4, 1.5 in text
- Section 3.1 in text
- Optional: PLive
  - CD that comes with text
  - References in text



Apologies to XKCD

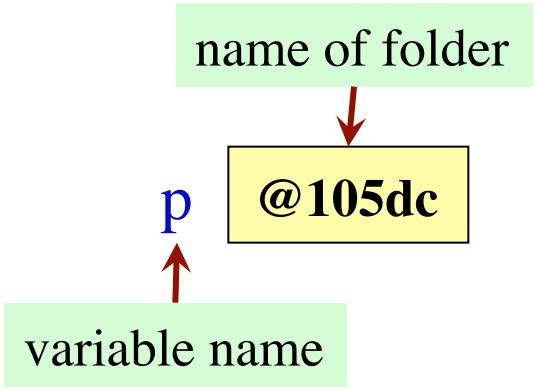
Assignment

- Assignment 1 due next week
  - Due **Friday, February 8th**
  - By 11:59PM to CMS
- Graded for mastery
  - Keep submitting until correct
  - But must **make progress**
- Visit consultant hours!
  - Sunday – Thursday 4:30-9:30

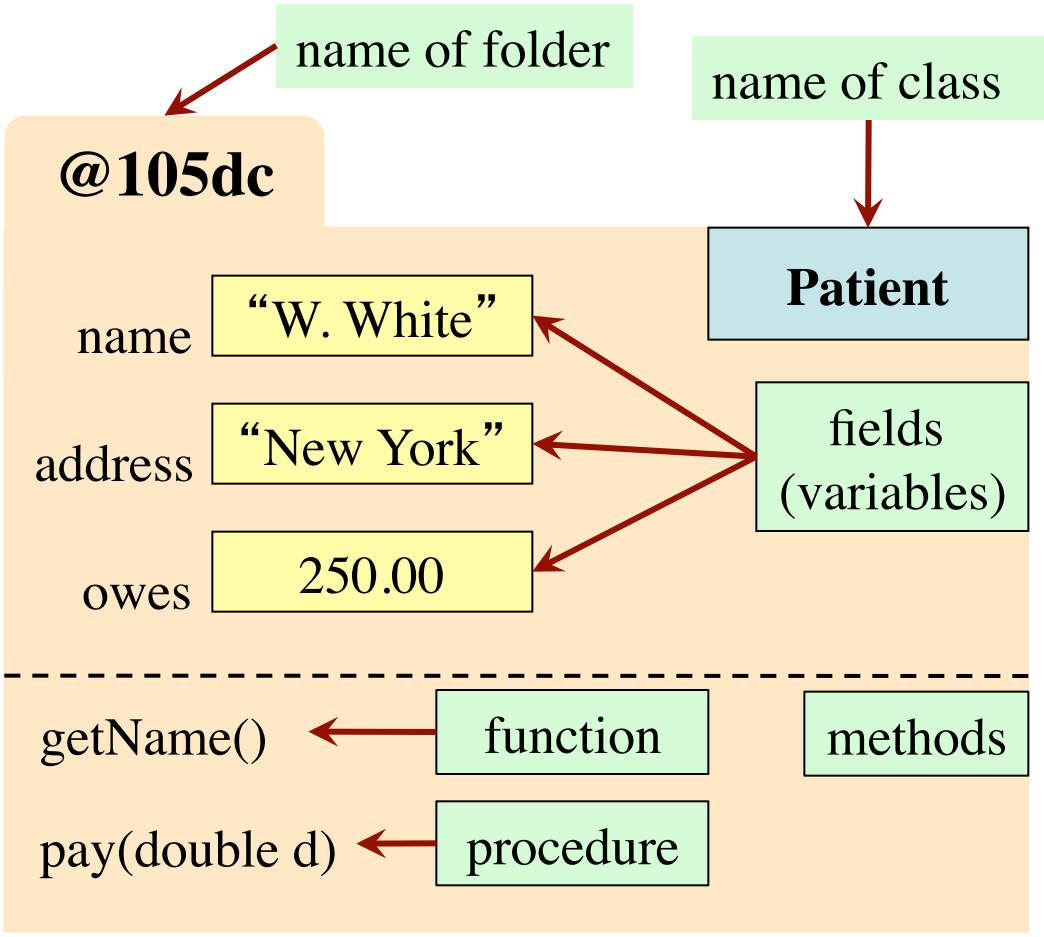
Assignment Details

- This is a very simple assignment
  - Just the basics of OO programming
  - Show you know enough to start a "real" program
- Work alone or with **one partner**
  - Partners "group themselves" on the CMS
  - Only one person submits the files.
  - Partners must do the work together, sit next to each other, with each taking turns "driving" (writing the code)
- **Academic Integrity**
  - Never look at someone's code or show yours to someone else
  - Never possess someone else's code (except your partner)

Extended Review From Last Time

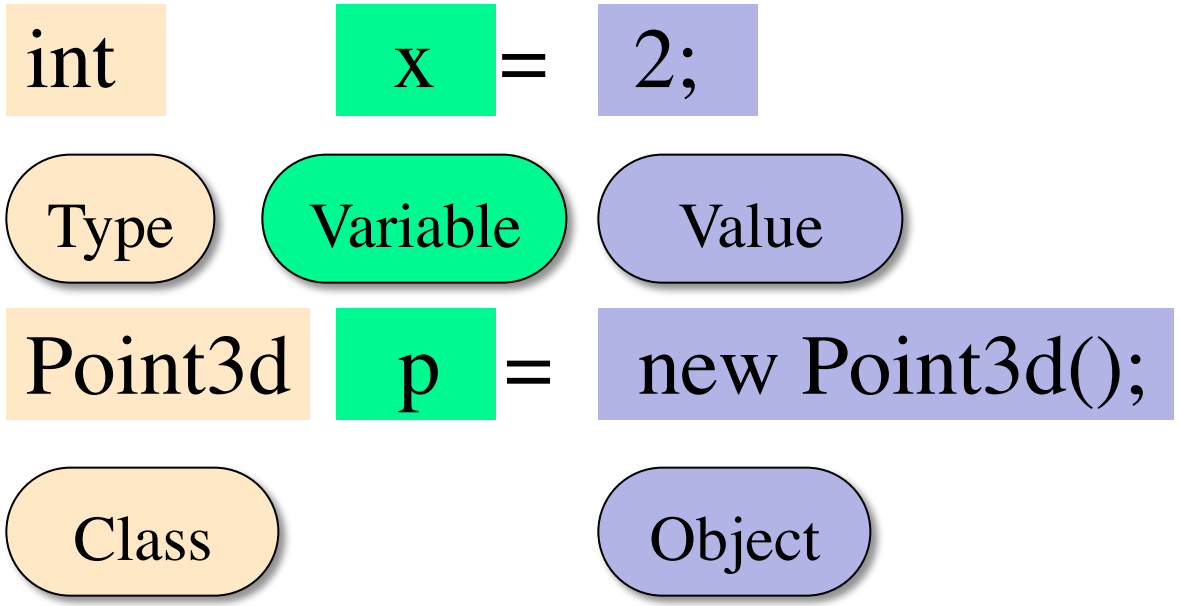


- p.getName()
  - Has value "W. White"
  - **Function**; gives value
- p.pay(250.00);
  - Sets owes to 0
  - **Procedure**;
    it does something



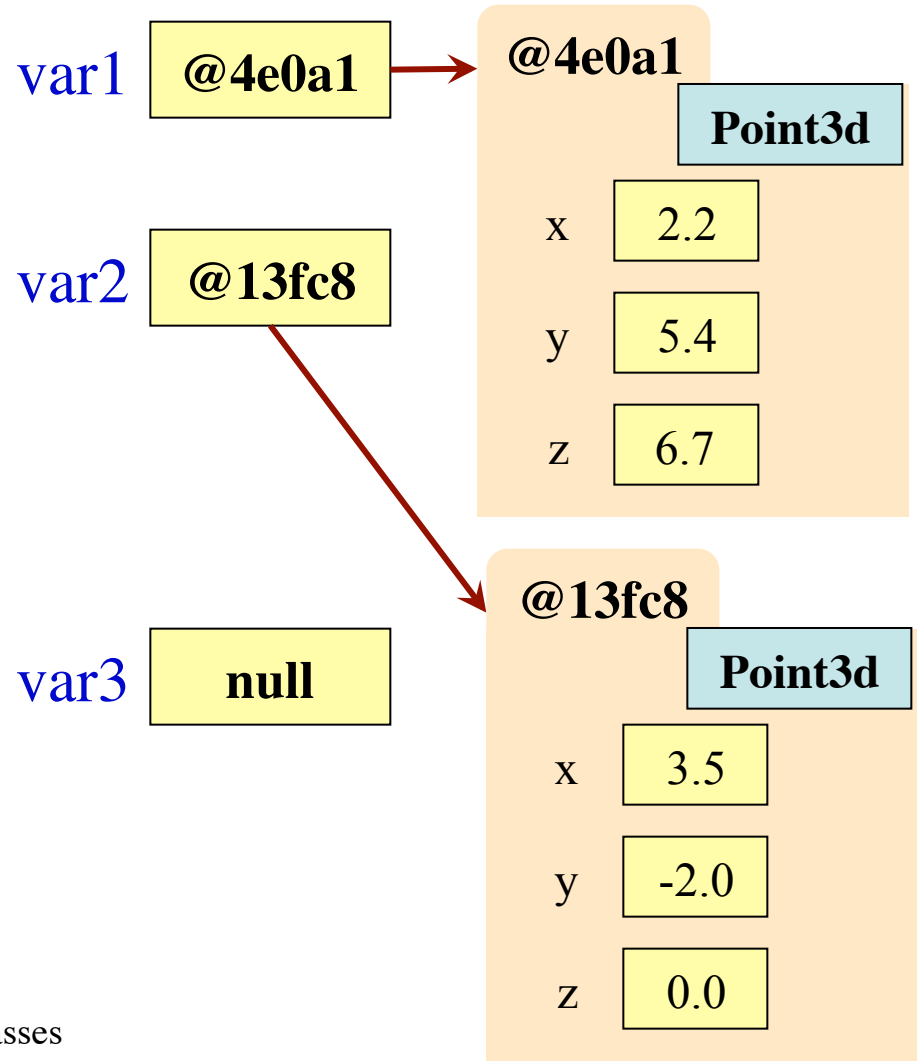
Class versus Object

Anatomy of a declaration + assignment statement:



The Value `null`

- You can declare a class variable w/o using new
  - Example: Point3d var3;
- Value in variable is **null**
  - **null**: Absence of a name
- var3.getX() gives error!
  - There is no name in var3
  - Does not know which Point3d to access
  - **NullPointerException**



Class Definition

- Describes the format of a folder (instance, object) of the class.

```
/**
 * Description of what the class is for
 */
public class <class-name> {

        declarations of fields and methods (in any order)
}
```

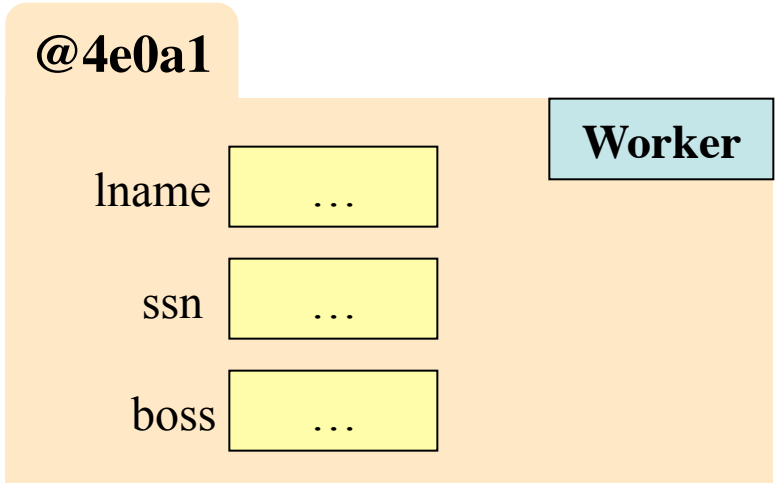
This is a **comment**
It does nothing.
It is a note to yourself

- The class and every method has a comment of the form

  / specification */**

- **This is a Javadoc comment** (Part of Lab next week).

Field: A Variable in each Folder of a Class



Declarations of fields

Invariants: Properties that are always true

```
/** 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)
}
```

Note the **private** and **public** keywords.
They are important but we will explain them later.

We Write Programs to Do Things

- Methods are the **key doers**

Method Definition

- Defines what method **does**

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

declaration of **parameter** n

Method **Header**

Method **Body** (inside {})

Method Call

- Command to **do** the method

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

argument to assign to n

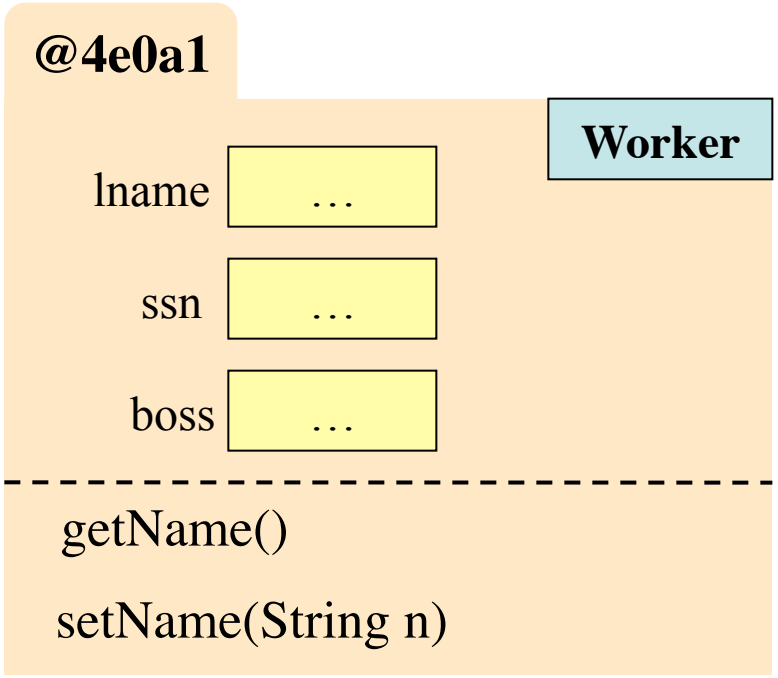
- **Parameter**: variable that is declared within the parentheses of a method header.
- **Argument**: a value to assign to the method parameter when it is called

Getter and Setter Methods

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

```
/** Yields: last 4 SSN digits, as int *
```

- *Try writing it yourself.*
- Full code on website



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

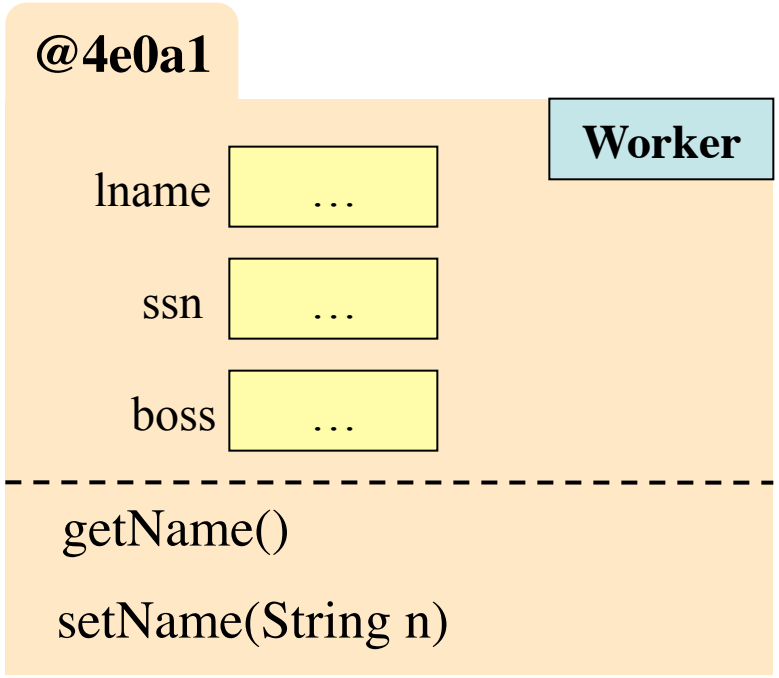
Setter methods (procedures) **set** or change fields of a folder

Getter and Setter Methods

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

value of function

procedure; no value




```
/** Yields: last 4 SSN digits, as int *
```

- *Try writing it yourself.*
- Full code on website

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

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

Why Getters and Setters?

- Fields have invariants:

```
/** 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)
}
```

- Allowing direct access to a field is bad.

```
w.lname = null; // Violates invariant!
```

- Sometimes want a field to be **read-only**
  - Can use in expressions, but not assignments
  - How do we do this?

Why Getters and Setters?

Setters

- Protect field invariants
- **Example**:

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

Invariant preserved

Getters

- Allow "read", not "write"
- **Example**:

```
public int getName() {
        return lname;
}
```

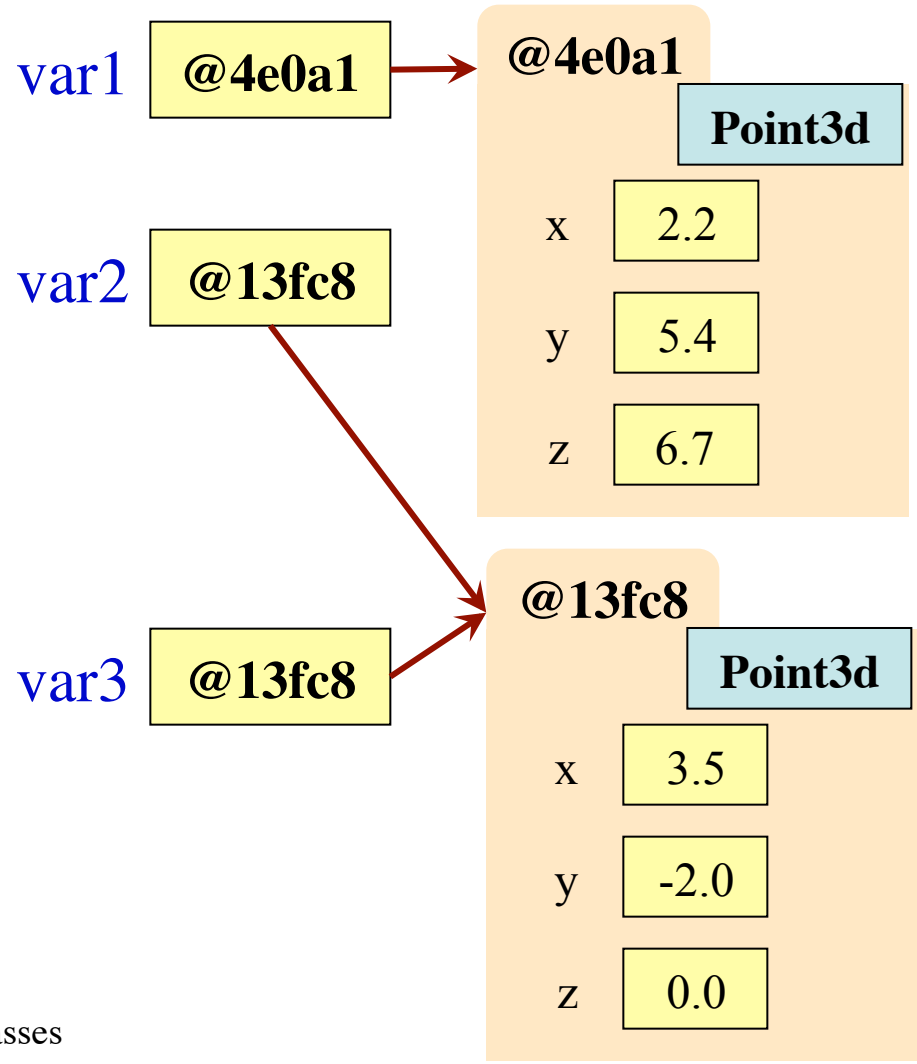
```
w.getName() = null; // Illegal!
```

How Methods Work

Memorize This!

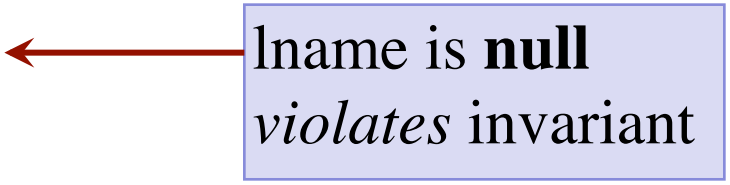
Write it down several times.

- **Example**: var1.getX()
  - Gets object (folder) name from the variable
  - Searches class (file drawer) for object (folder)
  - Executes commands inside the method on that object
- Methods apply to the **object** (folder), not the variable!
  - Execute var2.setX(8.2);
  - Makes var3.getX() == 8.2



Initializing the Fields of an Object (Folder)

- Creating a new Worker is now a multi-step process:
  - Worker w = new Worker();
  - w.setName("White");
  - …



- We would like to be able to use something like

  Worker w = new Worker("White", 1, null);

  - Create a new Worker, sets the last name to "White", the SSN to 0000000001, and the boss to **null**.
  - Need a special kind of method: **the constructor**

Initializing the Fields of an Object (Folder)

Memorize This!

Write it down several times.

- Creating a new Worker is now a mult
  - Worker w = new Worker();
  - ant
  - W
  - e",
    the SSN to 0000000001, and the boss to **null**.
  - Need a special kind of method: **the constructor**

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

Purpose of the Constructor

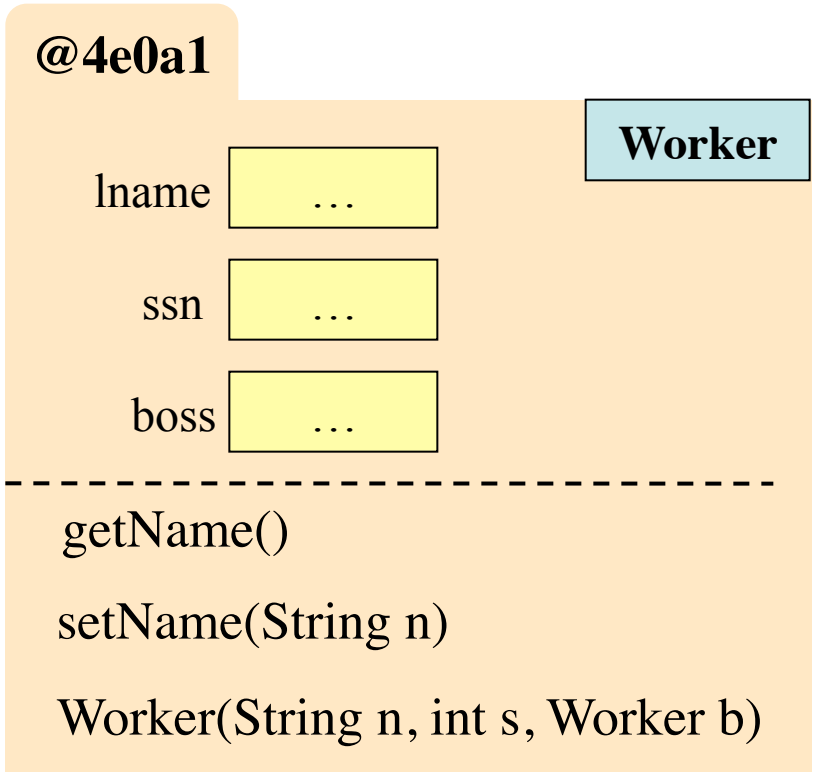
- Initialize the fields of a newly created object
- Make sure that the invariants are true

Example Constructor

```
/**
 * 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;
}
```

name of constructor = name of class

no void or type!



How "new" Is Evaluated

- Create a new object (folder) of class Worker
  - Initializes fields to default values
  - e.g. 0 for int, null for String
- Put the folder in file drawer
- Execute the constructor call Worker("White", 1, null)
  - Executes commands in body
  - Primarily to initialize fields
- Gives **the name** of the object as the final value of this expression

new Worker("White", 1, null)