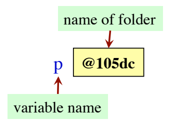
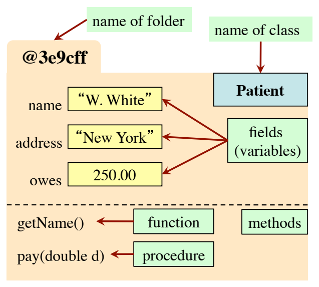
Extended Review From Last Time

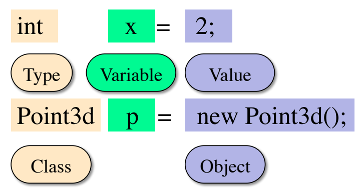


- p.getName()
  - Has value "W. White"
  - **Function**; gives value
- p.pay(250.0);
  - Sets owes to 0
  - **Procedure**;
    it does something



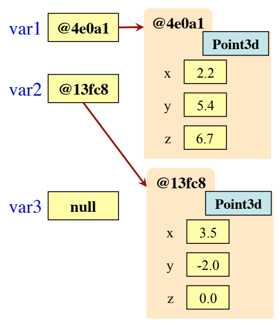
Class versus Object

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.

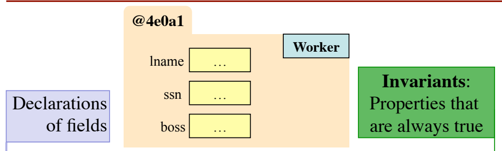
```
/**
 * Description of what the class is for
 */
public class <class-name> {

    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	Method Call
• Defines what method **does**	• Command to **do** the method

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

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

Method **Body** (inside {})

declaration of **parameter** n

Method **Header**

argument to assign to n

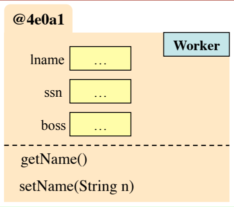
- **Parameter**: variable that is declared within the parentheses of a method header.
- **Argument**: a value to assign to the method parameter when it is called

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

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;
}
```

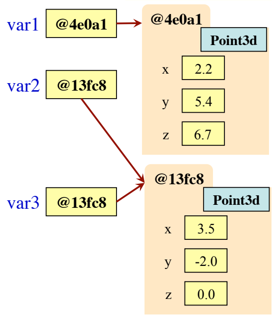
`w.getName() = null; // Illegal!`

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(); ← lname is **null** *violates* invariant
  - 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(); ← l
  - 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

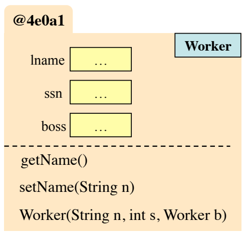
- Initialize the fields of a newly created object
- Make sure that the invariants are true

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!



@4e0a1
Worker
lname …
ssn …
boss …
getName()
setName(String n)
Worker(String n, int s, Worker b)