Lecture 3

Methods & Constructors

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

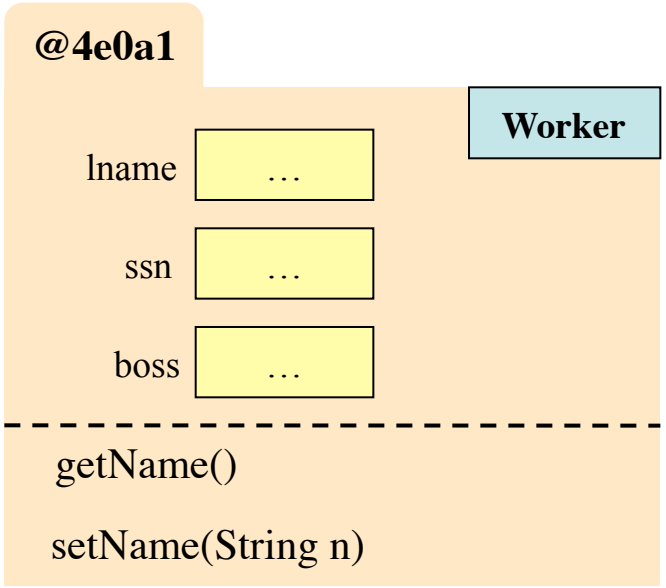
Aside: Private is a Class Property!

- 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));
     }
}
```

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" invariants in Assignment 1

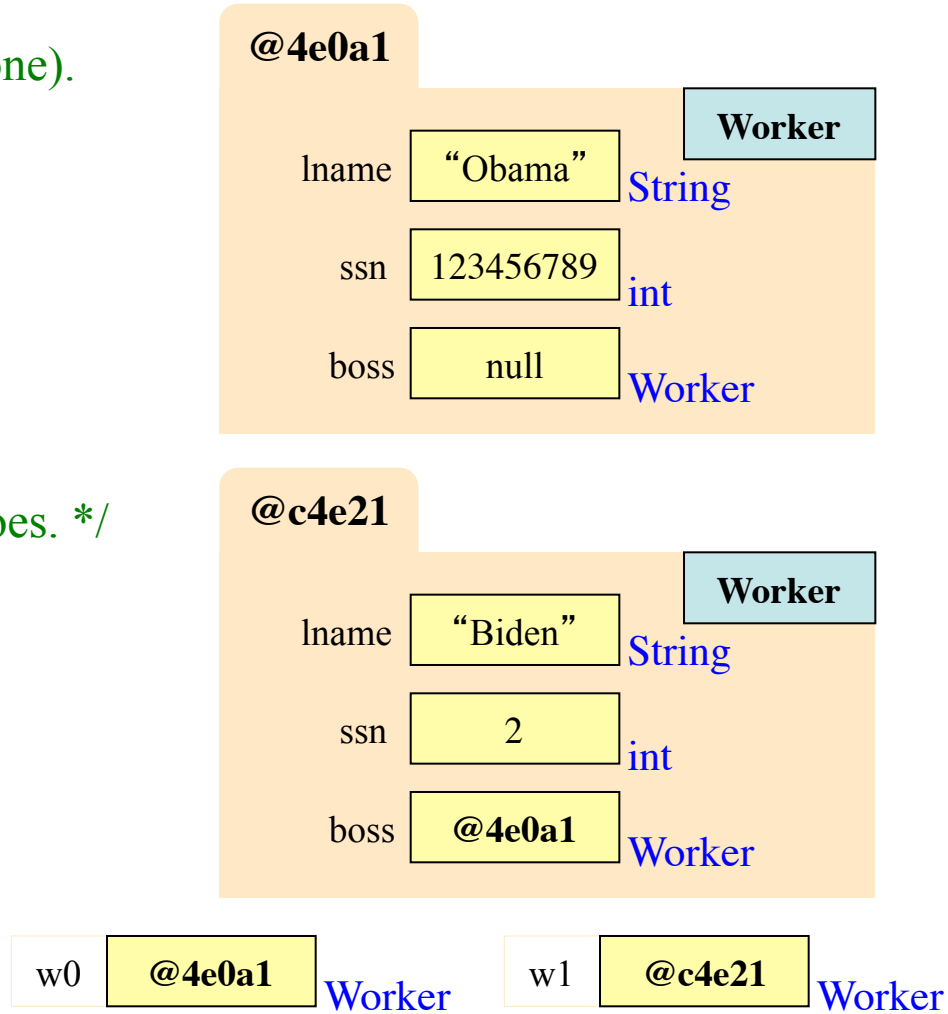



```
/** Set worker's last name to n
 * Precondition: n cannot be null
 *  or "Bob"
 */
public void setName(String n) {
    lname = n;
}
```

- Recall lname invariant
- Precondition ensures invariant is true

Specifications for Methods in Worker

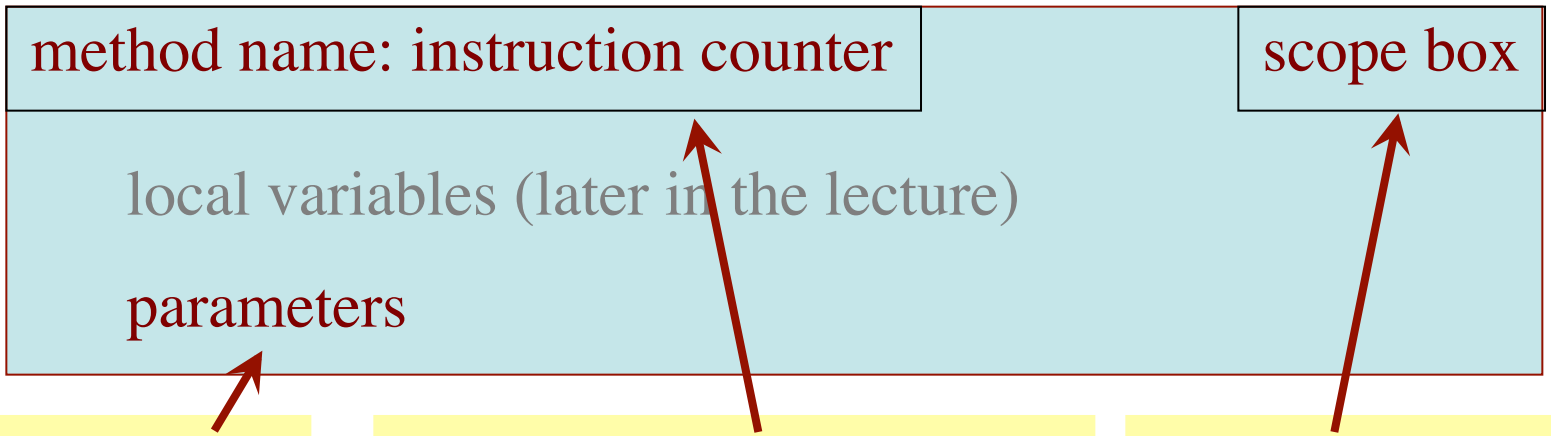
```
/** Constructor: a worker with last name n
  * ("" if none), SSN s, and boss b (null if none).
  * Precondition: n is not null,  s in
  * 0..999999999 with no leading zeros.*/
public Worker(String n, int s, Worker b)

/** 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)
```



How Do Methods Work?

Draw template on a piece of paper

- **Method Frame**: Formal representation of a method call
- *Remember* that methods are inside objects (folders)



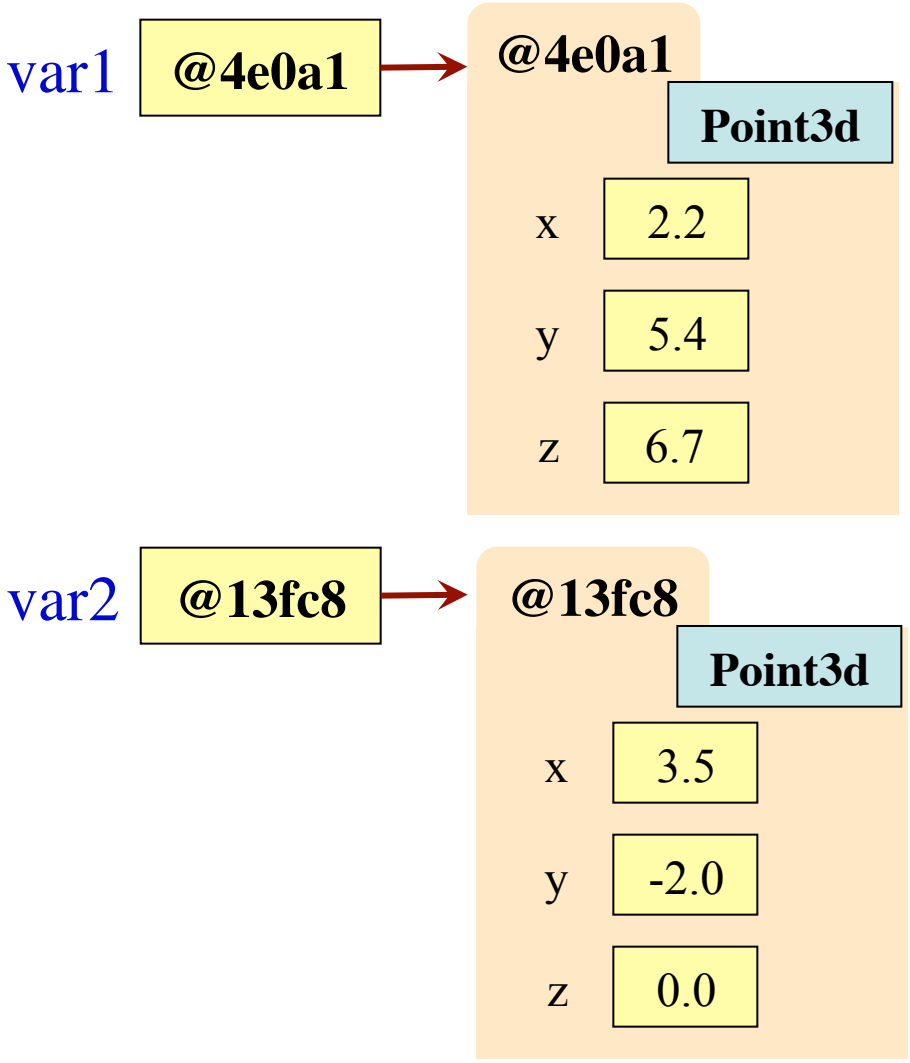
Draw parameters as variables (e.g. boxes)

- Number of the statement in method body to execute next
- **Starts with 1**
- Helps you keep track of where you are

- Contains the name of entity associated with the method
- Typically, the object in the method call

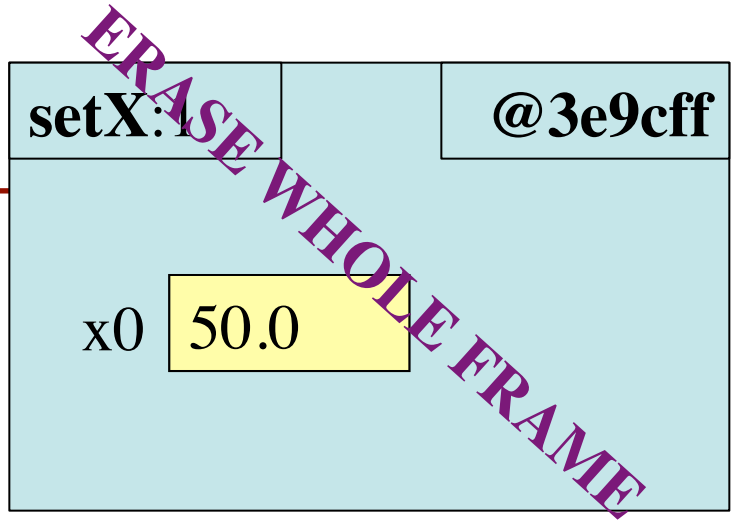
The Scope Box

- Most methods are attached to an object (folder)
  - Result depends on the object (folder) you use it on
- Example:
  - var1.getX() is 2.2
  - var2.getX() is 3.5
- Object (folder) you use for the method call is the **scope**
  - Goes in the **scope box**
  - Helps us keep track of "current" object



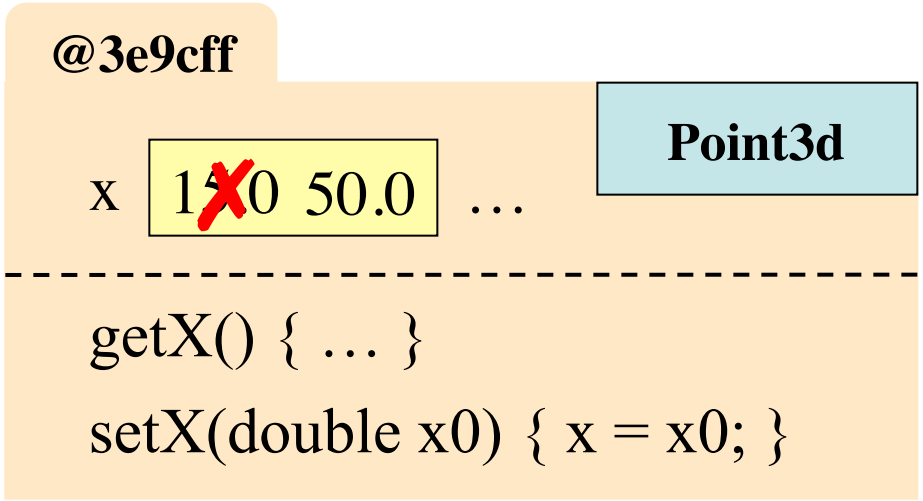
Example: p.setX(50.0);

1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Execute the method body
   - Look for variables in the frame
   - If not there, look in folder given by the scope box
4. Erase the frame for the call

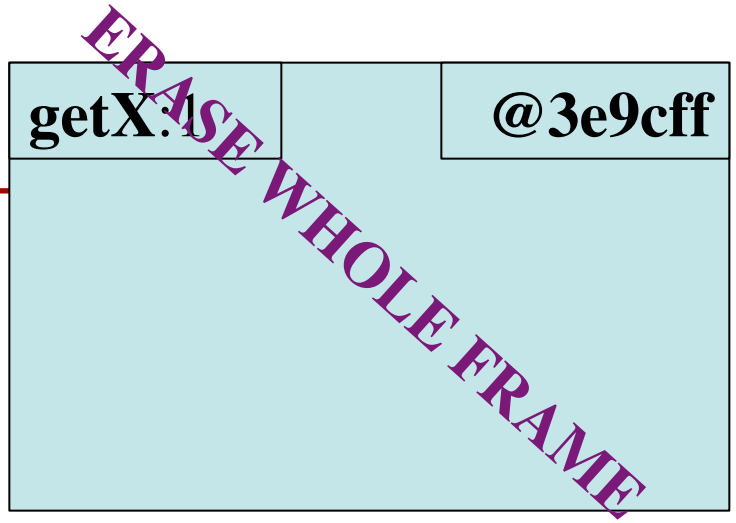



```
public void setX(double x0) {
    x = x0;
}
```

p **@3e9cff** Point3d

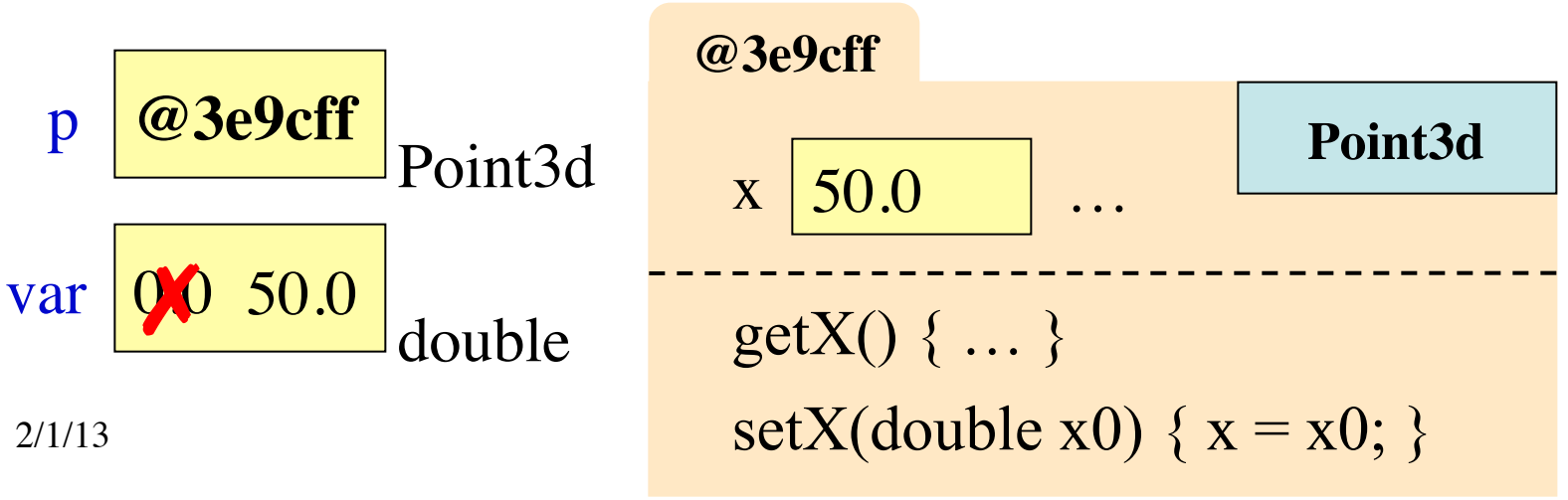


Example: var = p.getX();



1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Execute the method body
   - Look for variables in the frame
   - If not there, look in folder given by the scope box
4. Erase the frame for the call

```
public double getX() {
    return x;
}
```

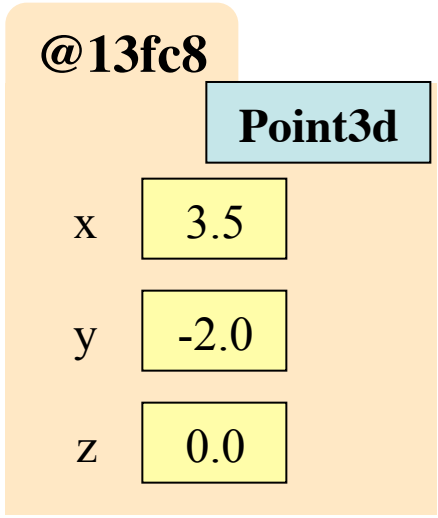


Static Methods

- **Static** methods are tied to a class (e.g. file drawer)
- They must not access the fields!
  - Fields are in the folders
  - Folders have different field values
- Their method calls are different:
  - <Class-Name>.<Method-Call>
- **Example**: Math methods in lab
  - Math.ceil(5.6);
  - Math.min(1,2);
  - Math.sqrt(5);

Class

Object

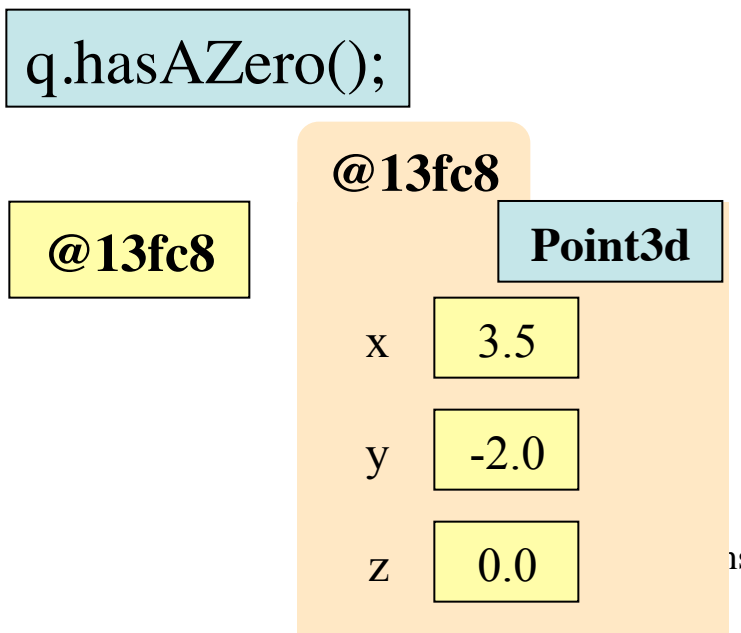


Defining Static Methods

Regular Version

```
/** Yields: "at least one of the
 * coordinates of this point is 0" */
public boolean hasAZero() {
    return x == 0 || y == 0 || z == 0;
}
```

Call: `q.hasAZero();`



Static Version

```
/** Yields: "at least one of the
 * coordinates of the point q is 0" */
public static boolean
        hasAZero(Point3d q) {
    return q.x == 0 || q.y == 0
                    || q.z == 0;
}
```

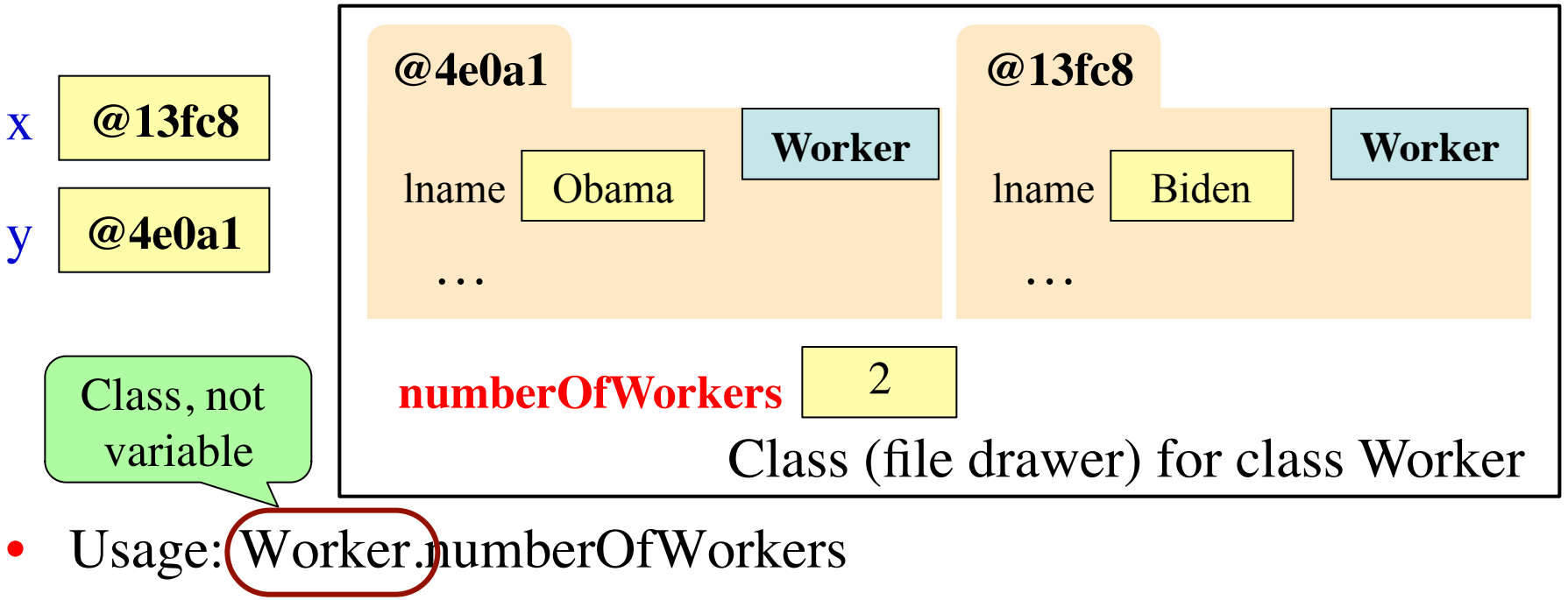
Call: `Point3d hasAZero(q);`

Goes in the **scope box**

Static Variables

- Static variable is a *single entity in the class*
  - Used to hold information about all objects
- Declare it just like a field declaration

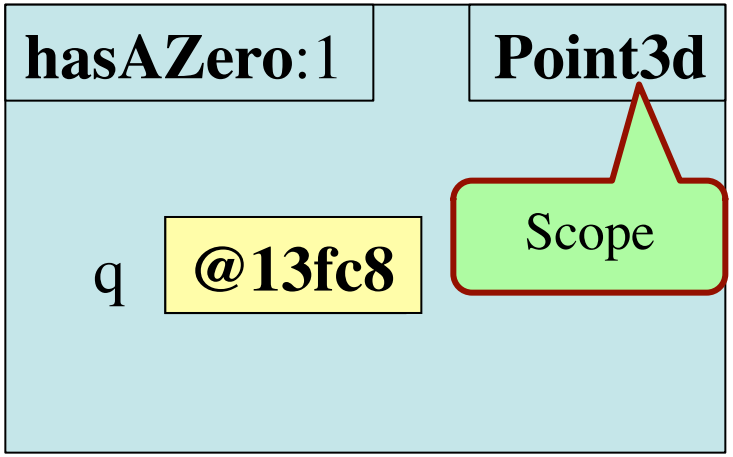
```
public static int numberOfWorkers;  // no. of Worker objects created
```



- Usage: Worker.numberOfWorkers

Method Model for Static Methods

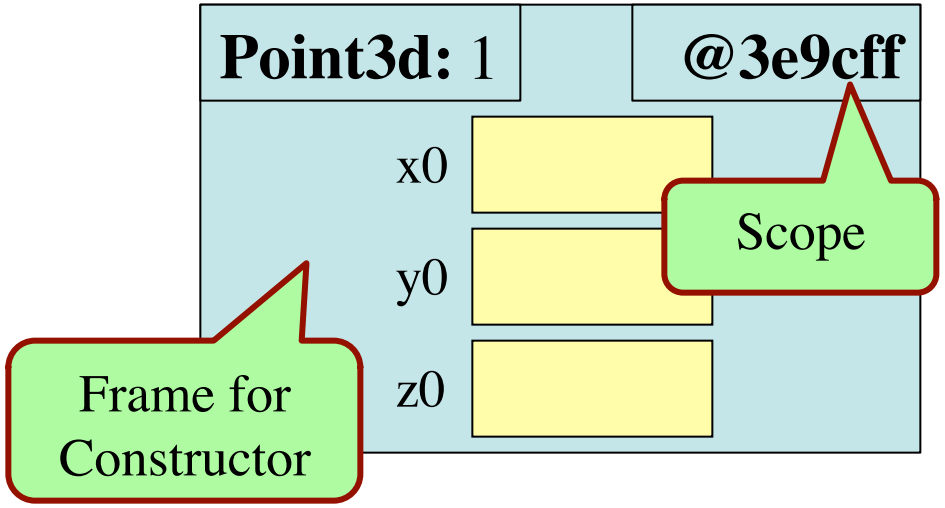
1. Draw a frame for the call
   - Scope box contains class!
2. Assign the argument value to the parameter (in frame)
3. Execute the method body
   - Look for variables in the frame
   - If not there, look in **static variables** in **class** in scope box
4. Erase the frame for the call




```
public static boolean
        hasAZero(Point3d q) {
    return q.x == 0 || q.y == 0
                    || q.z == 0
}
```

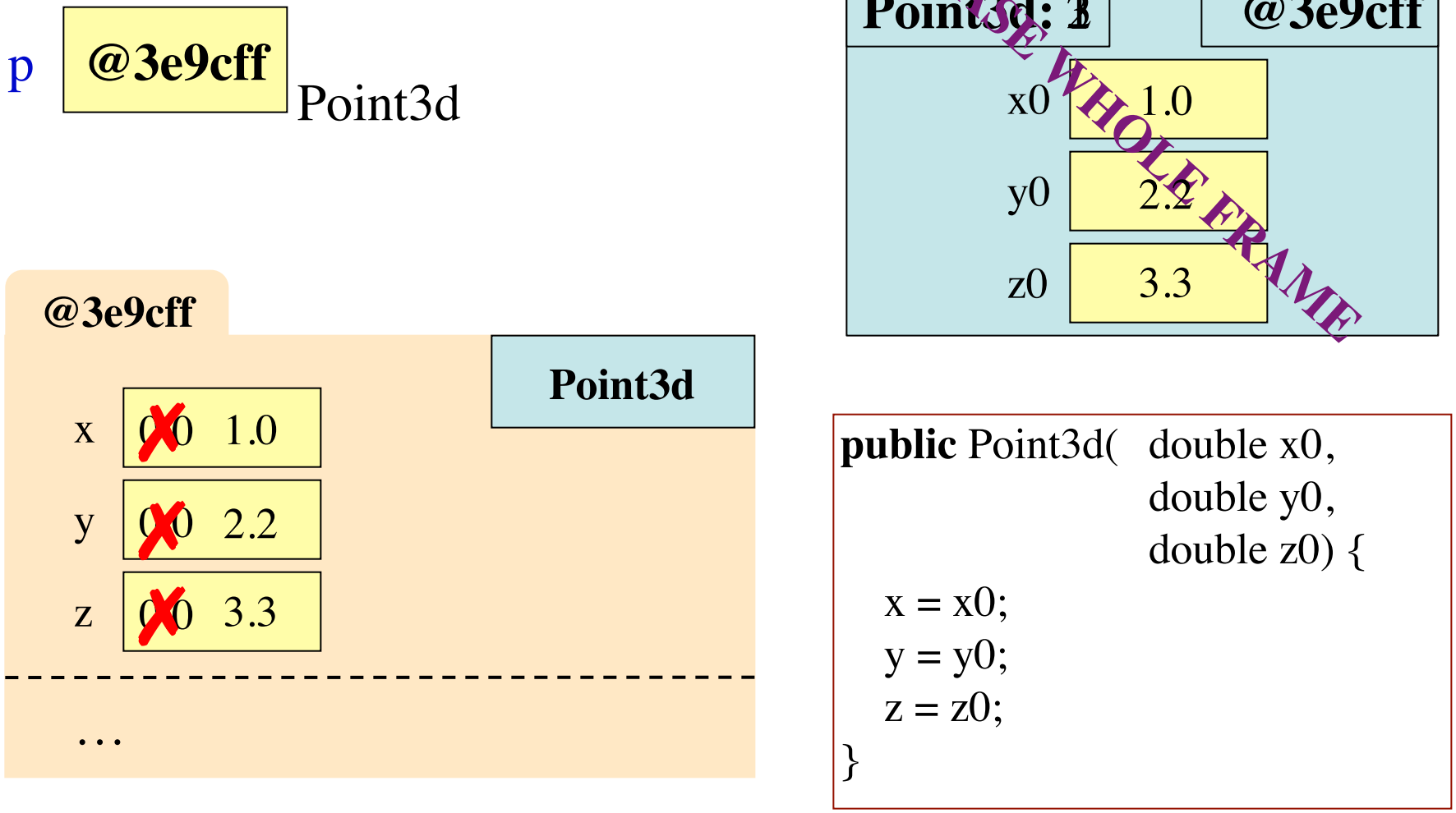
Constructors are Instance Methods

1. Make a new object (folder)
   - Java gives the folder a name
   - All fields are defauls (0 or null)
2. Draw a frame for the call
3. Assign the argument value to the parameter (in frame)
4. Execute the method body
   - Look for variables in the frame
   - Execute statements to initialize the fields to non-default values
   - Give the folder name as the result
5. Erase the frame for the call




```
public Point3d(  double x0,
                 double y0,
                 double z0) {
   x = x0;
   y = y0;
   z = z0;
}
```

Example: p = new Point3d(1.0, 2.2, 3.3);

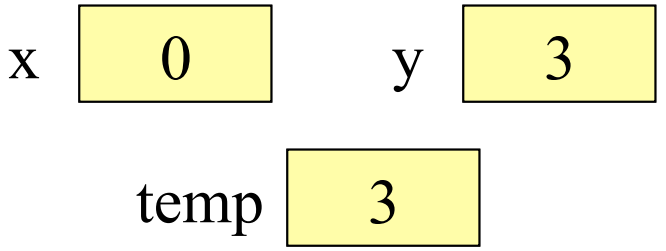



```
public Point3d(   double x0,
                  double y0,
                  double z0) {
   x = x0;
   y = y0;
   z = z0;
}
```

Local Variables

- **Local variable**: declared inside a *method body*
- Four types of variables:
  - Fields (in folders)
  - Parameters (method header)
  - Static (in file drawer)
  - Local (method body)
- Local variables are very useful with if-statements
  - Hold temporary values
  - "Scratch computation"

```
// swap x, y
// Put the larger in y
if (x > y) {
   int temp;
   temp = x;
   x = y;
   y = temp;
}
```



Local Variable Scope

```
/** Yields: the max of x and y */
public static int max(int x, int y) {
    // Swap x and y
    // Put the max in x
    if (x < y) {
        int temp;
        temp= x;
        x= y;
        y= temp;
    }

    return x;
}
```

scope of temp

Cannot use temp down here.
You will get an error!

- **Scope of local variable**: the places it can be used
- Only inside a "block"
  - Following the declaration
  - Inside of the braces {}