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:

```
p = new PublicPoint3d();
p.x = 3.0;
```

- No need for getters/setters

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

```
public Worker(String n, int s, Worker b) {
    /** Constructor: a worker with last name n ("" if none), SSN s, and boss b (null if none). */
    /** Yields: worker's last name */
    /** Yields: last 4 SSN digits w/o leading zeroes. */
    /** Yields: worker's boss (null if none) */
    /** Set boss to b */
}
```

Specifications for Methods in Worker

- Invariant: Property of a field
- Precondition: Property of a method parameter
- Responsibilities of the method call, not method definition
- How you will "enforce" invariants in Assignment 1

How Do Methods Work?

- Method Frame: Formal representation of a method call
- Remember that methods are inside objects (folders)
- Method: a block of code that does something
- Parameters: arguments passed to the method
- Local variables (in the method body)
- Name of the method
- Instruction counter: number of the statement in the method body to execute next

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;
}
```
Defining Static Methods

**Regular Version**

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

**Static Version**

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

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

Point3d's methods:

```java
Point3d hasAZero();
q.hasAZero();
```

Constructors are Instance Methods

1. Make a new object (folder)
   - Java gives the folder a name
   - All fields are defaults (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

Example:

```java
p = new Point3d(1.0, 2.2, 3.3);
```

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"

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

Local Variable Scope

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