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

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

Classes
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(250.0);**
  - Sets owes to 0
  - **Procedure**; it does something
Class versus Object

Anatomy of a declaration + assignment statement:

```java
int             x   =    2;
Point3d     p   =    new Point3d();
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>object</td>
<td>new Point3d();</td>
</tr>
<tr>
<td>int</td>
<td>x</td>
<td>2</td>
</tr>
<tr>
<td>Point3d</td>
<td>p</td>
<td>new Point3d();</td>
</tr>
</tbody>
</table>

1/28/13

Classes 5
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`

<table>
<thead>
<tr>
<th>var1</th>
<th>@4e0a1</th>
<th>var2</th>
<th>@13fc8</th>
<th>var3</th>
<th>null</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>@4e0a1</td>
<td></td>
<td>@13fc8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Point3d</td>
<td></td>
<td>Point3d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>2.2</td>
<td>x</td>
<td>3.5</td>
<td>x</td>
<td>3.5</td>
</tr>
<tr>
<td>y</td>
<td>5.4</td>
<td>y</td>
<td>-2.0</td>
<td>y</td>
<td>-2.0</td>
</tr>
<tr>
<td>z</td>
<td>6.7</td>
<td>z</td>
<td>0.0</td>
<td>z</td>
<td>0.0</td>
</tr>
</tbody>
</table>

1/28/13  Classes
Class Definition

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

```java
/**
 * Description of what the class is for
 */

public class <class-name> {
    declarations of fields and methods (in any order)
}
```

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

```java
/** specification */
```

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

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

1/28/13
Classes
Field: A Variable in each Folder of a Class

/** 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.

1/28/13
We Write Programs to Do Things

- Methods are the **key doers**

**Method Definition**
- Defines what method **does**

```java
public void setName(String n) {
    lname = 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

**Method Call**
- Command to **do** the method

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

- declaration of parameter `n`

- argument to assign to `n`
/** 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
/** 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
Why Getters and Setters?

- Fields have invariants:

```java
/** 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.
  ```java
  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:**
  ```java
  public void setName(String n) {
      lname = n;
      if (n == null) {
          lname = "";
      }
  }
  ```

### Getters
- Allow “read”, not “write”
- **Example:**
  ```java
  public int getName() {
      return lname;
  }
  ```

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

- **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`

Memorize This! Write it down several times.
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)

• 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

Purpose of the Constructor
• Initialize the fields of a newly created object
• Make sure that the invariants are true

Memorize This! Write it down several times.
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;
}

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 `new 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