Some Loose Ends

packages, more protection, this

packages

Packages

package - group of related classes, *e.g.*, read from web, graphs

Why?

- easy to bundle, distribute
- name clash

How?

The first lines of code in your file:

```
/* some comments here are ok */
package packagename;

public class Whatever {
    ...
```

How?

The first lines of code in your file:

```
/* some comments here are ok */
package packagename;

public class Whatever {
    ...
```

IDE

- package often created automatically
- to make a class part of a package, usually:
  - right click on the package when creating new class
  - drag and drop the class under the package

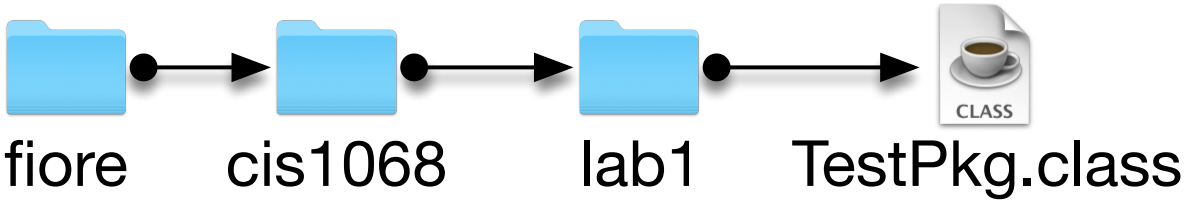
Directory Structure

It matters. Suppose that we have:

```
package fiore.cis1068.lab1;

public class TestPkg {
    public static void main(String args[]) {
        System.out.println("Did this work?");
    }
}
```

Must place resulting class file in:



Name Clash

- me in grade school
- you make a `Math` class

Name Clash

- me in grade school
- you make a `Math` class
- convention. URL (globally unique) in reverse order, *e.g.*,

Name Clash

- me in grade school
- you make a `Math` class
- convention. URL (globally unique) in reverse order, *e.g.*,
  - you own the domain `www.citizensagainstfiore.org`
  - all of your packages begin `org.citizensagainstfiore.www`

Using Classes in Packages

import particular class

```
import java.util.Scanner;
...
Scanner in = new Scanner(System.in);
```

import all classes within package

```
import java.util.*;
...
Scanner in = new Scanner(System.in);
```

import nothing

```
java.util.Scanner in = new java.util.Scanner(System.in);
```

Default Package

- What happens if we don't use the `package` statement?
- Class becomes part of the **default package**
  - all of the classes in the current directory

protection

Remaining Java Protection Levels

public	accessible anywhere
private	accessible only within the class
protected	accessible within class, its descendents, package
no keyword	*package* access

this

OK, but slightly cumbersome

```
public class Point {
    protected int x;
    protected int y;

    public Point(int newX, int newY) {
        x = newX;
        y = newY;
    }

    public void move(int dx, int dx) {
        x += dx;
        y += dy;
    }
   ...
```

```
    ...

    Point p1 = new Point(10, 20);
    Point p2 = new Point(10, 20);

    ...

    p1.move(5,5);
    p2.move(1,1);
```

Same thing, but with this

```
public class Point {
   protected int x;
   protected int y;

   public Point(int x, int y) {
      this.x = x;
      this.y = y;
   }

   public void move(int dx, int dy) {
      this.x += dx;
      this.y += dy;
   }
  ...
```

```
...

Point p1 = new Point(10, 20);
Point p2 = new Point(10, 20);

...

p1.move(5,5);
p2.move(1,1);
```

What If We Didn't Use This? Mistake.

```
public class Point {
   protected int x;
   protected int y;

   /* broken constructor */
   public Point(int x, int y) {
      x = x;  // in here, we're referring
      y = y;  // to the local (i.e., the
              // argument), not the
              // field of the class
              // --> does nothing useful
   }

   public void move(int dx, int dx) {
      this.x += dx;
      this.y += dy;
   }
  ...
```

Two different x's

- one local to the constructor
- one the field of the class
- when we're inside the constructor, x refers to the local

Calling One Constructor from Another

One Way

```
public class Point {
   protected int x;
   protected int y;

   public Point() {
     this.x = 0;
     this.y = 0;
   }

   public Point(int x, int y) {
      this.x = x;
      this.y = y;
   }

  ...
```

Using this

```
public class Point {
   protected int x;
   protected int y;

   public Point() {
     this(0, 0);
   }

   public Point(int x, int y) {
      this.x = x;
      this.y = y;
   }

  ...
```