

Lecture 11
CSE11 – Fall 2013
Java Interfaces and File Differences

Looking at differences in Files

- Very often, it is desirable to list the differences between two files
 - Code stopped working and you have an older working version, what did you do?
 - How “close” are two files to discover if code is being needlessly replicated
- Two tools
 - diff
 - vimdiff

diff

- How to learn about a unix command
 - google search
 - man — online manual
- Diff has many options
 - diff <file1> <file2>
 - This is standard diff
 - diff -c <file1> <file2>
 - This is called a context diff because it prints lines of code that surround the identified difference

Look at code from Last Lecture

```
$ diff FallingLogo.java FallingLogoCallback.java
4c4
< public class FallingLogo extends ActiveObject {
---
> public class FallingLogoCallback extends ActiveObject {
16c16,19
<     public FallingLogo(Image logo, Location initialLocation, DrawingCanvas aCanvas) {
---
>     private LogoControllerCallback myController;
>
>     public FallingLogoCallback(LogoControllerCallback master, Image logo,
> Location initialLocation, DrawingCanvas aCanvas) {
18a22
>     myController = master;
26a31
>     myController.atBottom(this);
```

Context

```
$ diff -c FallingLogo.java FallingLogoCallback.java
*** FallingLogo.java    2013-04-15 11:13:11.000000000 -0700
--- FallingLogoCallback.java  2013-04-15 11:24:54.000000000 -0700
*****
*** 1,7 ***
    import objectdraw.*;
    import java.awt.*;

! public class FallingLogo extends ActiveObject {

    // the delay between successive moves of the ball
    private static final int DELAY_TIME = 33;
--- 1,7 ----
    import objectdraw.*;
    import java.awt.*;

! public class FallingLogoCallback extends ActiveObject {

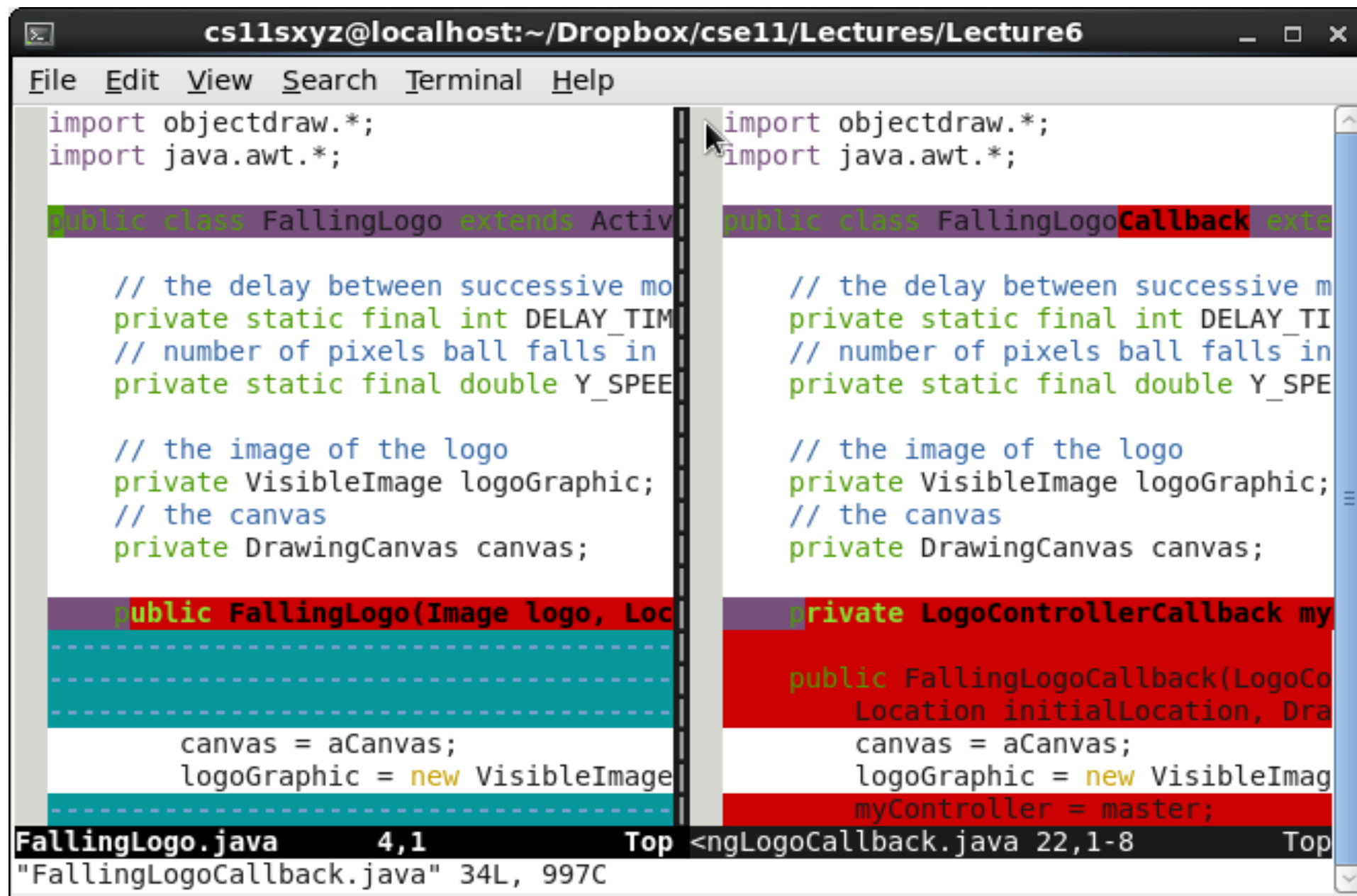
    // the delay between successive moves of the ball
    private static final int DELAY_TIME = 33;
*****
*** 13,21 ***
```

If you do this long enough...

- You get used to the format of diffs
- Even with practice can get “muddled” when the diffs become complicated, (deletions, insertions, small changes)
- Discover a tool called “patch” that takes the original file + diff file and produce the new file
- There are more visual tools
 - Let's look at `vimdiff`

vimdiff

```
$ vimdiff FallingLogo.java FallingLogoCallback.java
```



```
cs11sxyz@localhost:~/Dropbox/cse11/Lectures/Lecture6
File Edit View Search Terminal Help

import objectdraw.*;
import java.awt.*;

public class FallingLogo extends Activ

// the delay between successive mo
private static final int DELAY_TIM
// number of pixels ball falls in
private static final double Y_SPEE

// the image of the logo
private VisibleImage logoGraphic;
// the canvas
private DrawingCanvas canvas;

public FallingLogo(Image logo, Loc

-----
        canvas = aCanvas;
        logoGraphic = new VisibleImage

-----

FallingLogo.java 4,1 Top
"FallingLogoCallback.java" 34L, 997C

import objectdraw.*;
import java.awt.*;

public class FallingLogoCallback exte

// the delay between successive m
private static final int DELAY_TI
// number of pixels ball falls in
private static final double Y_SPE

// the image of the logo
private VisibleImage logoGraphic;
// the canvas
private DrawingCanvas canvas;

private LogoControllerCallback my

public FallingLogoCallback(LogoCo
        Location initialLocation, Dra
        canvas = aCanvas;
        logoGraphic = new VisibleImag
        myController = master;
```

Detailing the Differences

- Differences Between FallingLogo and FallingLogoCallback
- 2 lines changed:
 - class name changed
 - class constructor changed
- 4 lines added
 - Keep track of controller for callback
 - Perform the callback
- ==> These are very very close

Differences Between The LogoControllerCallback[Timed]

```
$ diff LogoControllerCallback.java LogoControllerCallbackTimed.java
4c4
< public class LogoControllerCallback extends WindowController {
---
> public class LogoControllerCallbackTimed extends WindowController {
9c9
<     private FallingLogoCallback droppedLogo; // the falling Logo
---
>     private FallingLogoCallbackTimed droppedLogo; // the falling Logo
27c27
<         droppedLogo = new FallingLogoCallback(this, logo, point, canvas);
---
>         droppedLogo = new FallingLogoCallbackTimed(this, logo, point, canvas);
31c31
<     public void atBottom(FallingLogoCallback logo)
---
>     public void atBottom(FallingLogoCallbackTimed logo)
38c38
<         new LogoControllerCallback().startController(400,600);
---
>         new LogoControllerCallbackTimed().startController(400,600);
```

Five Differences

- Every difference is an object type
 - `LogoControllerCallback --> LogoControllerCallbackTimed`
 - `FallingLogoCallback --> FallingLogoCallbackTimed`
 - `FallingLogoCallback droppedLogo; --> FallingLogoCallback droppedLogo;`
 - signature of `atBottom()` changed
 - `FallingLogoCallback logo --> FallingLogoCallbackTimed logo`
 - `main()` method changed because of class name change

Why do we need different versions of very similar programs?

- Java is strongly typed
 - A logo controller is defined to drop only a particular object type
 - a FallingLogo (w/callback) is defined to only callback to a particular kind of controller
- But... A logoController with callback is performing the same function as any other logoController (w/callback)
 - Isn't there a way to make a controller support ANY logo object that will properly callback (e.g. calls `atBottom()`)?
 - Isn't there a way make a FallingLogo (w/callback) support ANY controller that has an `atBottom()` method

Java Interfaces

- Interfaces are one method to helping us write code that supports “more generic (or abstract)” Objects
- An Interface is a “contract” -- A class that declares that it implements an interface. It “promises” to implement specific methods with specific signatures
 - This is one way to get around the fact that java does not support multiple inheritance (later this quarter)
-
-

Using an Interface Reference

- An “interface” is a specification of a set of methods that a class (or object) supports
- This means that a program can declare a variable to have the type of the interface
 - `Drawable2DInterface sqOrOval;`
 - `sqOrOval` could refer to a `Filled/Framed Rectangle`, or a `Filled/Framed Oval`.
 - A variable of that is an interface type, cannot be constructed it is simply a reference
 - Only classes can be constructed

Example Method

```
void printHeight(Drawable2DInterface someGraphic)
{
    System.out.println("Height is " + someGraphic.getHeight());
    Object general = someGraphic;
    System.out.println("The real class is: " +
        general.getClass().getCanonicalName());
}

// someGraphic must be a constructed object that implements the
// Drawable2DInterface
// --any-- object that implements that interface will do. Even
// those that aren't defined by objectdraw or inherited from
// objectdraw
```

Two interfaces

FallingObject.java:

```
public interface FallingObject {  
    public void run();  
}
```

ControllerCallback.java:

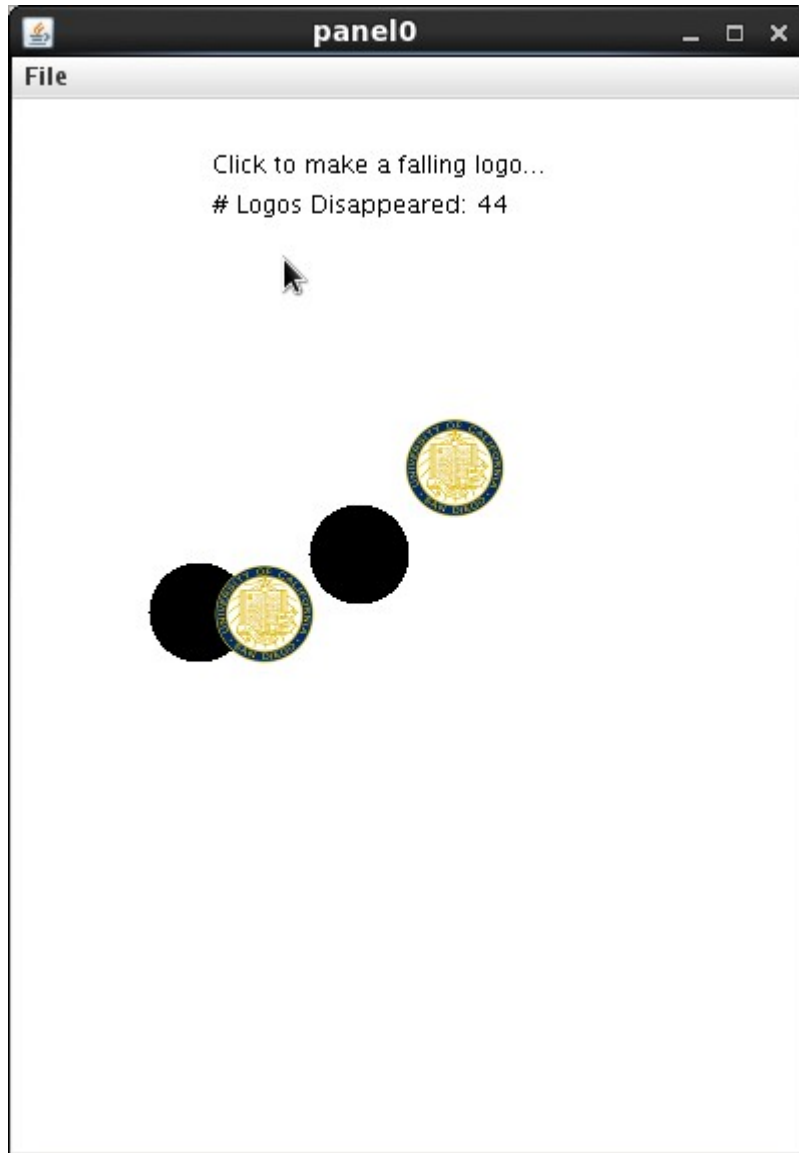
```
public interface ControllerCallback {  
    public void atBottom(FallingObject drop);  
}
```

Interfaces only have public methods and constants

How to think about this

- A controller can drop any class that implements the FallingObject interface
- Any falling object can call back to any class that implements the ControllerCallback interface

LogoControllerCallback (Revised with Interfaces)



- Drop Logos or Ovals
-

LogoController Implements

- `public class LogoControllerCallback
extends WindowController implements
ControllerCallback {`
- Now the Object to be dropped is Generic
- `private FallingObject
droppedObject; // the falling
object logo or oval`

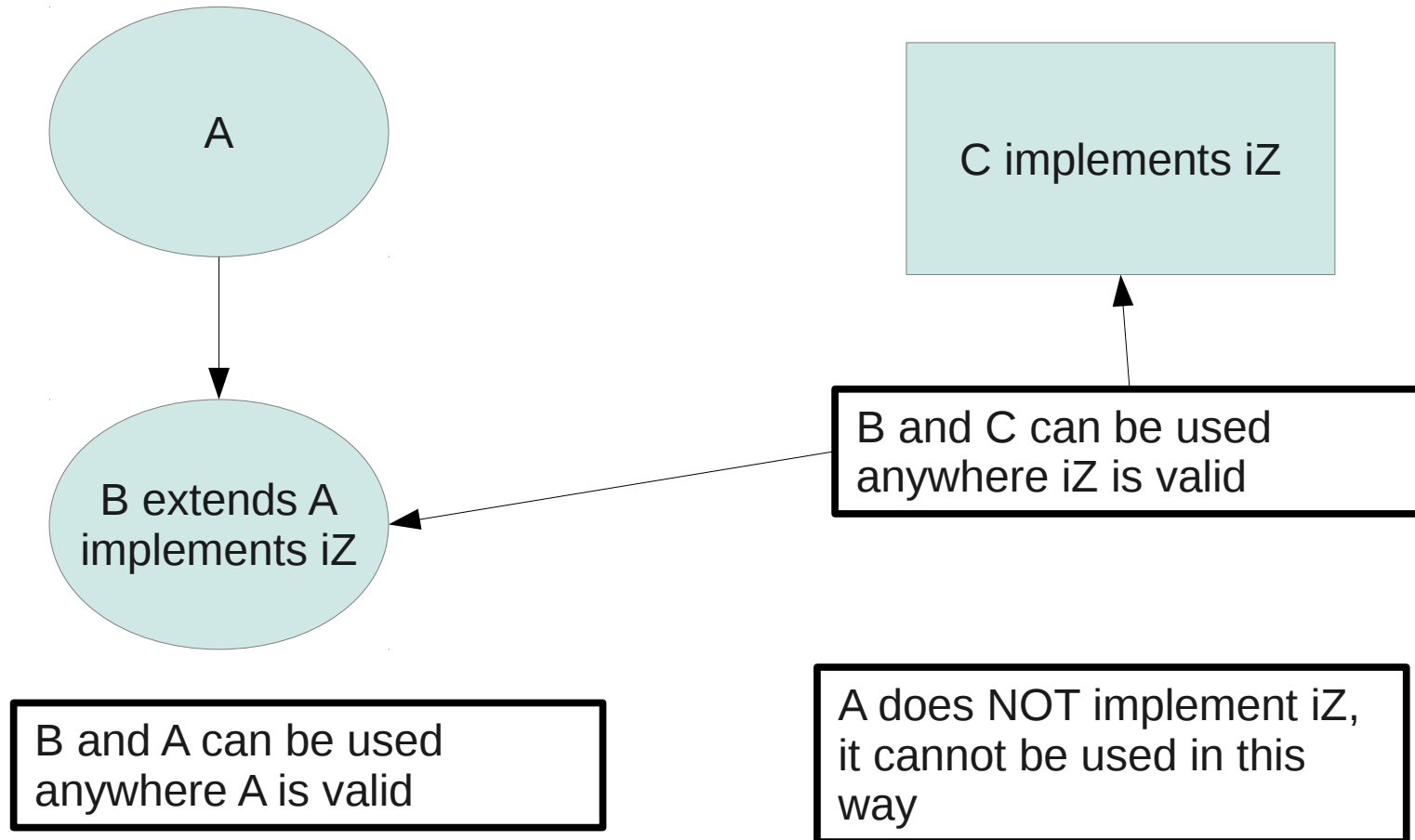
Two Falling Objects

- FallingLogoCallback
 - `public class FallingLogoCallback extends ActiveObject implements FallingObject {`
- FallingOvalCallback
 - `public class FallingOvalCallback extends ActiveObject implements FallingObject {`
- Can be used with any controller implements ControllerCallback
 - `private ControllerCallback myController;`

Some things to keep in mind

- An interface exposes certainly functionality of an object.
 - Objects can *have **more** methods* than defined by the interface
 - The *must implement **all** the methods* defined in the interface
 - All method declarations must be public
- interfaces are a kind of “inheritance”
 - We've seen inheritance with extends
 - Programs using extends `WindowController`
 - Can be used any place `WindowController` is valid
-

Extends vs. Implements



Implement multiple interfaces?

- YES! a class can implement multiple interfaces
 - Allows it to be used in a number of different ways
- implements Interface1, Interface2, ..., InterfaceN
- Implements is an interface “contract”, objects should do roughly the same thing for each defined method.

Extend and Interface?

- YES!
- Suppose interface J extends interface K
- Any class that implements K must interface all of the methods defined in J, too.
- K can only add new methods to J
 - New methods OR
 - existing method names with new signatures.

Some Objectdraw Interfaces

- DrawableInterface
 - Drawable2DInterface extends DrawableInterface
 - Resizable2DInterface extends Drawable2DInterface
- VisibleImage implements Resizable2DInterface
 - ==> DrawableInterface , Drawable2DInterface
- Same is true for FilledRects, FilledOvals ...

Define Constants in an Interface?

- YES!
- If a class implements an interface and the interface defines a constant (public static final ...)
 - the class includes the constants