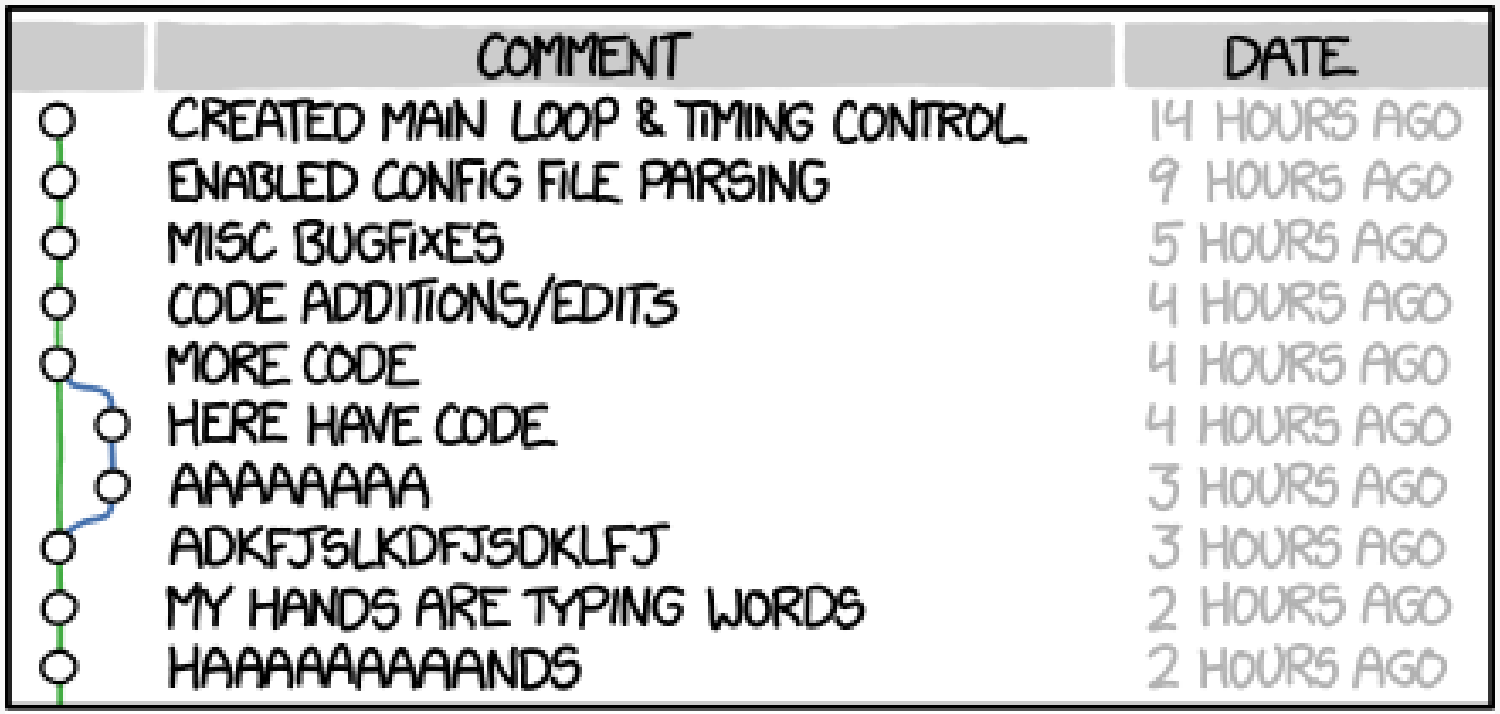

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Section 10:
Design Patterns 2

Slides by Alex Mariakakis

with material from Hal Perkins

List of Design Patterns

We discussed several patterns during the quarter. For reference, here is a list of some of them:

- *Creational: Factory, Singleton, Builder, Prototype, Interning, Flyweight*
- *Structural: Adapter, Composite, Decorator, Flyweight, Proxy*
- *Behavioral: Interpreter, Observer, Iterator, Strategy, MVC, Visitor*

Winter 2012 Q9a

Your program can send content to a printer. You want to ensure that different parts of the program don't command the printer to print output belonging to multiple documents simultaneously, mixed together.

Singleton – ensure that all threads use the same print manager/queue.

We allowed Proxy as an answer if a good explanation was supplied, although most answers were arguing for using a Proxy to implement Singleton.

Winter 2012 Q9b

You are writing a library that allows client code to read and write compressed data for storing in files. You've written several classes that implement encode() and decode() methods of a CompressedFile interface. These classes use different algorithms with different characteristics. Rather than requiring the client to choose among them, you want the client to be able to simply to request a CompressedFile object and be given an instance of the best available implementation for the particular kind of file.

Factory – If the client were to call a constructor directly, he/she would need to know what implementation of CompressedFile he/she wants to use. Instead, what we want is a way for the client to ask for a CompressedFile without knowing what concrete class to use. In other words, we want a static factory method like getCompressedFile() that contains all the logic for deciding what class to construct.

Some people answered Strategy, which we accepted even though it's not quite correct. Because we didn't describe this pattern in much detail we didn't expect people to know the subtle distinctions between Strategy and factory.

Winter 2012 Q9c

You have an application that reads and writes files on the local file system. Without changing the existing application any more than absolutely necessary you would like to modify it so it can read and write files stored at a different location on the network, even though reading and writing data over the network requires using a different API and a different set of methods than reading and writing data stored in local files.

Adapter – create a wrapper class whose API matches what the application already expects for reading/writing locally. The methods in the adapter use the API to read and write data over the network.

We gave partial credit for Proxy. That would be the correct answer if the local object were simply acting as a stand-in for a remote service using the same protocol. But here the object is translating from one API to another, so Adapter is the actual pattern being used.

Decorator was not a correct answer here. That pattern wraps a basic abstraction with an augmented version, rather than translating from one abstraction to another.

Winter 2012 Q9d

You are writing a program that maintains a grid of objects. Although there are several million objects in the grid, there are only a few hundred distinct objects, but there are many copies of each one. Unfortunately each of the objects require a fair amount of storage and creating several million individual objects is using way too much storage.

Interning – create one each of the several hundred distinct objects. When the client requests a new object, give them a reference to the existing object of the same value (or for a lazy implementation, create the object the first time it is requested).

Fall 2012 Q12

a) You have a set of objects that is unlikely to change, but the set of operations on them is likely to change. You want to keep the code for each operation together in a single module.

Visitor/Procedural

b) You have a class with many optional parameters, and want to avoid creating a large number of constructors.

Builder (partial credit for Factory)

c) You have a Point class whose interface uses polar coordinates and you want to provide a wrapper so it can be used to implement a version of Point that uses rectangular coordinates.

Adapter

Fall 2012 Q12

d) You want to wrap a library class that manages data connections so that it logs connection attempts as they occur.

Decorator (partial credit for Proxy)

e) You want to save memory costs of creating many duplicate objects of a class which has only a few distinct abstract values.

Interning (partial credit for Factory, Flyweight)

f) When new objects of your class are created the new objects should actually have a more appropriate subtype.

Factory

Fall 2012 Q8

More generic questions. Suppose we have the following classes defined:

```
class Shape { … }
class Circle extends Shape { … }
class Rectangle extends Shape { … }
```

Now suppose we have a program that contains the following objects and list:

```
Object o;
Shape s;
Circle c;
Rectangle r;
```

Fall 2012 Q8

`List<? extends Shape> les;`

	legal	illegal
`les.add(s);`		X
`les.add(c);`		X
`c = les.get(0);`		X
`s = les.get(0);`	X	
`o = les.get(0);`	X	

`List<? super Shape> lss;`

	legal	illegal
`lss.add(s);`	X	
`lss.add(c);`	X	
`c = lss.get(0);`		X
`s = lss.get(0);`		X
`o = lss.get(0);`	X	

Fall 2013 Q2

```
/** A tic-tac-toe game */
public class TicTacToe {
 // instance variables omitted

 /** Initialize a new game object */
 public TicTacToe() { }

 /** Print the state of the game board on System.out */
 public void printGame() { }

 /** Read the player's next move from the keyboard
 * and update the game to reflect that move. */
 public void getPlayerMove() { }

 /** Calculate the computer's next move and print it on
 * System.out */
 public void computerMove() { }

 /** Calculate the computer and user's current scores
 * and print them on System.out */
 public void printScores() { }

 /** Reset the game back to the same initial state it had
 * when it was created */
 public void resetGame() { }
}
```

Fall 2013 Q2

What is (are) the major design flaw(s) in the above design? A brief couple of sentences should be enough to get the point across.

The major design flaw is that the game logic and user interface have been jumbled together in a single class. These need to be separated, particularly if we ever hope to be able to replace the user interface with a different one without a major rewrite.

Fall 2013 Q2

Describe how you would refactor (change) the initial design to improve it. Briefly sketch the class(es) and methods in your design, and what each of them do. You do not need to provide very much detail, but there should be enough so we can tell how you've rearranged the design, what the major pieces are, and how they interact.

The important thing that the application should be split using the MVC pattern to separate the game logic from the interface

Model: game logic. Operations would include the following from the original code:

- **Create a new game model**
- **Record a new player move**
- **Calculate a new move for the computer**
- **Return the current user and computer scores to the caller**
- **Return the current state of the game board to the caller**
- **Reset the game board to start a new game**

Fall 2013 Q2

Describe how you would refactor (change) the initial design to improve it. Briefly sketch the class(es) and methods in your design, and what each of them do. You do not need to provide very much detail, but there should be enough so we can tell how you've rearranged the design, what the major pieces are, and how they interact.

Viewer/Controller (likely a single module in a console text-based game):

- **Process initialize and reset commands from the user**
- **Read player moves and call the model to update the game state**
- **Display computer moves**
- **Display current contents of the game board and current scores**

The viewer/controller and the model would likely use some form of the Observer pattern to communicate.

Fall 2013 Q3

We would like to modify this code to change it into a generic method that works with any sorted array whose elements have type Comparable<T> (i.e., the elements can be ordered using the compareTo method). Mark the code above to show the changes needed to turn this into a generic method.

```
/** Search a sorted array of integers for a given value
 * @param a Array to be searched
 * @param x Value to search for in array a
 * @return If x is found in a, return largest i such
 * that a[i]==x. Return -1 if x not found in a
 * @requires a!=null & a is sorted in non-decreasing
 * order (i.e., a[0]<=a[1]<=...<=a[a.length-1])
 */
```

Fall 2013 Q3

```
public static int bsearch(int[] a, int x) {
      int L = -1;
      int R = a.length;
      // inv: a[0..L] <= x && a[R..a.length-1] > x &&
      // a[L+1..R-1] not examined
      while (L+1 != R) {
            int mid = (L+R)/2;
            if (a[mid] <= x)
                  L = mid;
            else // a[mid] > x
                  R = mid;
      }
      if (L >= 0 && a[L] == x)
            return L;
      else
            return -1;
 }
```

Fall 2013 Q3

```
public static <T extends Comparable<T>> int bsearch(T[] a, T x) {
      int L = -1;
      int R = a.length;
      // inv: a[0..L] <= x && a[R..a.length-1] > x &&
      // a[L+1..R-1] not examined
      while (L+1 != R) {
            int mid = (L+R)/2;
            if (a[mid].compareTo(x) <= 0)
                  L = mid;
            else // a[mid] > x
                  R = mid;
      }
      if (L >= 0 && a[L].compareTo(x) == 0)
            return L;
      else
            return -1;
}
```

Fall 2013 Q3

```
/** Search a sorted array ~~of integers~~ for a given value
 * @param a Array to be searched
 * @param x Value to search for in array a
 * @return If x is found in a, return largest i such
 * that a[i]==x. Return -1 if x not found in a
 * @requires a!=null & a is sorted in non-decreasing
 * order (i.e., a[0]<=a[1]<=...<=a[a.length-1])
 */
```

Fall 2013 Q4

There's something fishy about this question. Suppose we have the following class hierarchy:

```
class Creature extends Object { }
class Fish extends Creature {
     /** Return the weight of this Fish */
     public float getWeight() { return ...; }
}
class Salmon extends Fish { }
class Haddock extends Fish { }
class Tuna extends Fish { }
```

Class `Creature` *does not have a* `getWeight` *method. Class* `Fish` *implements that method and all of its subclasses inherit it.*

Fall 2013 Q4

Write a `static` *method* `collectionWeight` *whose parameter can be any Java* `Collection` *containing* `Fish` *objects (including objects of any* `Fish` *subclass(es)). Method* `collectionWeight` *should return the total weight of all the* `Fish` *in the Collection, using* `getWeight` *to determine the weight of each individual* `Fish`*.*

Hints: Method `collectionWeight` *will need a proper generic type for its parameter. Java* `Collections` *are things like* `Lists` *and* `Sets` *that implement the* `Iterable` *interface. They do not include things like* `Maps`*, which are not simple collections of objects.*

Fall 2013 Q4

Using wildcards:

```
public static float collectionWeight(Collection<? extends Fish> f) {
     float totalWeight = 0;
     for (Fish fish: f)
          totalWeight += fish.getWeight();
     return totalWeight;
}
```

Or without wildcards:

```
public static <T extends Fish> float collectionWeight(Collection<T> f)
```

WRAP UP