

Topdown v Bottomup

- Programming is changing our world
 - Empowering, liberating, equalizing,...
- Everything is a bit: all 0's and 1's
 - From jpg to mp3 to ...
- It's about problems! It's about details!
 - Should we think about problems to get to the details?
 - Should we master details before grand thinking?
- See [Wikipedia on topdown v bottomup design](#)

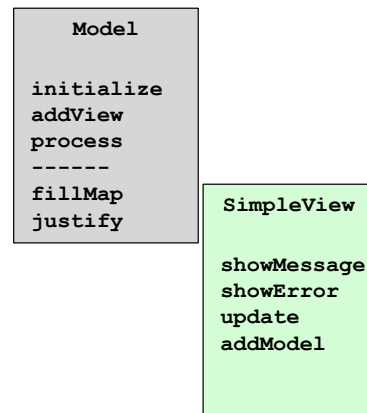


Conventions in Compsci 100(e) projects

- We want you to concentrate on algorithms and data structures
 - Not on rendering fonts, interacting with users
 - This is important! But not what this course is about
- We try to build GUIs or views that facilitate projects
 - You write the brains, we build the skull/manikin
 - Our GUI communicates with your code
 - Requires following conventions in interacting code
- GUI libraries are similar across languages, but...
 - Deeper Java specific details than HashMap

MVC Example, Markov

- User loads file
 - Where? Communicate to?
 - What changes in model?
 - What happens in view?
- User chooses word
 - Process in Model
 - Alternatives?
 - Generate context, display
 - How to show in any view?



KWIC main program/class

```
public class MarkovMain {
    public static void main(String[] args){
        IModel model = new MarkovModel();
        SimpleViewer view =
            new SimpleViewer("Compsci 100e Markov",
                "k count>");
        view.setModel(model);
    }
}
```

- What changes in above, e.g., for other assignments?
 - How can view communicate with *any* model?
 - View doesn't change, model does!
 - Requires using a Java interface to capture commonality

Model View Controller, MVC

- **Gui is the View and often the controller**
 - Separate user-interaction from updates to data
- **User loads file, chooses word, ...**
 - Model notified, computes, updates view
- **Model has all the state and knows when it changes**
 - Communicates changes to views (via controller)
 - Must be initialized, updated, etc.
- **Very common *Design Pattern***
 - Capture common solutions to problems in a context
 - Iterator, Composite, Decorator seen in Compsci 100e

Convention Summary

- **Classes start with capital letter and then we have:**
 - They're public, except nested class? Protected means ...
 - camelCaseForMethods and ForClasses
 - Ivars, fields, instance variables, mySize, myMap, ...
 - Constants (public static) are ALL_CAPS
- **Interfaces are IModel, IView, and so on**
 - Not true for standard Java classes, yes for Compsci 100
 - Don't need to label methods as abstract, but can
- **Supply AbstractDefault implements IThing**
 - Constructor, some state, some common behavior: extend!

Methods, Interfaces, Inheritance

- **A method by any other name would smell as sweet**
 - Method in OO languages, functions, procedures in others
 - Parameters and return value: communication
 - Do objects or methods communicate?: OO v procedural
- **Static : `Math.sqrt`, `Character.isUpperCase`, ...**
 - Don't belong to an object, invoked via class (clue above?)
 - Java API helpful here
- **Interface: implement class with required, related methods**
 - HashMap, TreeMap
 - ArrayList, LinkedList, Vector

Interfaces continued

- **In the beginning**
 - Make code work, don't worry about generalizing
 - But, if you write code using `Map` rather than `TreeMap`
 - Can swap in a new implementation, coded generally!
- **Don't know how to optimize: space or time**
 - Facilitate change: use interface rather than concrete class
 - My DVD connects to my TV, regardless of brand, why?
 - How do you turn on a Nokia cell phone? Motorola? But!
- **Interfaces facilitate code refactoring**
 - Don't add functionality, change speed or memory or ...