Plan for the week

- Classes and Objects
- Model-View-Controller
- Maps and Hashing

- Upcoming
  - Towards Algorithm analysis
- APTs:
  - IsomorphicWords, MedalTable, and Maps
- Assignment
  - Markov
- Lab: Understanding Markov

What can an Object do (to itself)?

  - Look at java.lang.Object
  - What is this class? What is its purpose?

- toString()
  - Used to print (System.out.println) an object
  - overriding toString() useful in new classes
  - String concatenation: String s = "value " + x;
  - Default is basically a pointer-value

What else can you do to an Object?

- equals(Object o)
  - Determines if guts of two objects are the same, must override, e.g., for using a.indexOf(o) in ArrayList a
  - Default is ==, pointer equality

- hashCode()
  - Hashes object (guts) to value for efficient lookup

- If you're implementing a new class, to play nice with others you must
  - Override equals and hashCode
  - Ensure that equal objects return same hashCode value

Objects and values

- Primitive variables are boxes
  - think memory location with value
- Object variables are labels that are put on boxes
  - String s = new String("genome");
  - String t = new String("genome");
  - if (s == t) {they label the same box}
  - if (s.equals(t)) {contents of boxes the same}

What's in the boxes? "genome" is in the boxes
Objects, values, classes

- For primitive types: int, char, double, boolean
  - Variables have names and are themselves boxes (metaphorically)
  - Two int variables assigned 17 are equal with ==

- For object types: String, ArrayList, others
  - Variables have names and are labels for boxes
  - If no box assigned, created, then label applied to null
  - Can assign label to existing box (via another label)
  - Can create new box using built-in new

- Object types are references/pointers/labels to storage

Conventions in CompSci 100 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

Anatomy of a class

```java
public class Foo {
    private int mySize;
    private String myName;
    public Foo(){
        // what's needed?
    }
    public int getSize(){
        return mySize;
    }
    public double getArea(){
        double x;
        x = Math.sqrt(mySize);
        return x;
    }
}
```

- What values for vars (variables) and ivars (instance variables)?

KWIC: Key word in Context

Arise, fair sun, and kill the envious moon, Who
I. Yet I should kill thee with much cherishing.
shortly, for one would kill the other. Thou! why,
those twenty could but kill one life. I beg
wherefore, villain, didst thou kill my cousin? That villain
mean, But 'banished' to kill me - 'banished'? O friar,
thou happy. Tybalt would kill thee, But thou slewest
cell there would she kill herself. Then gave I
heaven finds means to kill your joys with love!

- Read file, find word and it’s context, print
  - Can find all words, but how do we get context?
  - Loop and look: create context line for each occurrence
  - See ContextModel.java
Use KWIC example to motivate study

- Dissect and inspect KWIC code to understand conventions
  - Understand Model and View interaction
  - Facilitates doing RSG and Markov text-generation

- Review some basic coding idioms and ideas
  - Avoiding recomputing same value, readability, modifiability, ...

- Errors: possible for a method to fail but still work?
  - See KWIC code when same key occurs twice on a line!

MVC Example, key-word-in-context

- 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?

Key Word in Context Explained

- For every different word, store where it occurs
  - *love* is the 1st, 3rd, 50th, and 1237th word in the file

- This data is kept in a map, key is word, value is ??
  - How do we generate the data in the map?

- Given a word, how do we find its context? How do we format?
  - All words are in an array, in order
  - Memory concerns?
  - Original KWIC paper by Parnas as comparison

Code Interlude

- Examine ContextModel.process
  - Called when user enters word, parameter is the word
  - If file already read, we don’t need map, where is this?
  - Error checking? When and when happens
  - How does Model communicate back to View?

- Examine ContextModel.justify
  - What is method doing
  - What is parameter, where was it constructed, issues?
  - What about ‘magic numbers’, e.g., 30?
  - What about comments, should we add some?
**KWIC main program/class**

```java
public class ContextMain {
    public static void main(String[] args){
        ContextModel model = new ContextModel();
        SimpleViewer view =
            new SimpleViewer("Compsci 100e KWIC", "word");
        view.setModel(model);
    }
}
```

- What changes in above, e.g., for Markov assignment?
  - 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

**Tomato and Tomato, how to code**

- `java.util.Collection` and `java.util.Collections`
  - one is an interface
    - add(), addAll(), remove(), removeAll(), clear()
    - toArray(), size(), iterator()
  - one is a collection of static methods
    - sort(), shuffle(), reverse(), max(), min()
    - frequency(), binarySearch(), indexOfSubList()

- `java.util.Arrays`
  - Also a collection of static methods
    - sort(), fill(), copyOf(), asList()

**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
  - Map: HashMap, TreeMap
  - List: ArrayList, LinkedList, Stack, 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 ...

What does Object-Oriented mean?

- Very common method of organizing code
  - Design classes, which encapsulate state and behavior
  - Some classes can be similar to, but different from their parent class: inheritance
    - Super class, subclass
  - Inherit behavior, use as is or modify and use or both

- Complex to design a hierarchy of classes, but important
  - More of this in Compsci 108 or on-the-job training
  - We’re solving simple problems, not designing re-usable libraries

- Simple does not mean straight-forward!

Inheritance and Interfaces

- Interfaces provide method names and parameters
  - The method signature we can expect and thus use!
  - What can we do to an `ArrayList`? To a `LinkedList`?
  - What can we do to a `Map` or `Set` or `PriorityQueue`?
  - `IAutoPlayer` is an interface

- Abstract classes can implement core, duplicated code
  - If we can add one object to a `[set,map,list]`, can we add an entire list of objects? `java.util.AbstractCollection`
  - If we can iterate can we remove? Convert to array? Obtain size?
  - `Abstract` is an interface

Miscellany

- Inheritance and overriding
  - Inheritance is the process by which a Class assumes the properties of its Superclasses
  - An object checks its own methods before consulting the Superclass thus overriding the Superclass methods
  - Polymorphism: many classes respond to some common message.

- Access Control
  - `public`: accessible anywhere
  - `private`: accessible only within methods of this class
  - `protected`: accessible to this class or subclasses
  - `No modifier`: accessible within class and package
Convention Summary

- Classes start with capital letter and then we have:
  - They’re public, except nested class? Protected means ...
  - camelCaseForMethods and ForClasses
  - 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!

Documentation

- Standard identifiers
  - i, j, k: integer loop counters
  - n, len, length: integer number of elements in collection
  - x, y: cartesian coordinates (integer or real)
  - head, current, last: references used to iterate over lists.

- Variable name guidelines
  - Use nouns that describe what value is being stored
  - Don’t reiterate the type involved

- Comments for methods and classes
  - Abstraction: What does it do?
  - Implementation: How does it do it?

- Inline comments as needed

From Comparable to Comparator

- When a class implements Comparable then ...
  - Instances are comparable to each other
    - “apple” < “zebra”, 6 > 2
    - Sorting Strings, Sorting WordPairs, ...
      - Method compareTo invoked when ...
    - Comparable<..> types the parameter to compareTo
  - Return < 0, == 0, > 0 according to results of comparison

- Suppose we want to change how Strings compare
  - Or change class Foo implements Comparable<Foo>
  - What if we need more than one way to compare Foo’s?

java.util.Comparator

- How does sorting work in general and in Java?
  - Characteristics of Java library sort methods
  - What can be sorted?
  - How do you change how sorting works?

- APT ClientsList: example to explore Comparator
  - Creating new Comparator: nested class
    - Should it be public? Private? Matter?
  - Comparator could anonymous, but then issues.

- What does it mean to implement Comparable?
  - Other Java interfaces: cloneable, serializable, ...

From Comparable to Comparator

- When a class implements Comparable then ...
  - Instances are comparable to each other
    - “apple” < “zebra”, 6 > 2
    - Sorting Strings, Sorting WordPairs, ...
      - Method compareTo invoked when ...
    - Comparable<..> types the parameter to compareTo
  - Return < 0, == 0, > 0 according to results of comparison

- Suppose we want to change how Strings compare
  - Or change class Foo implements Comparable<Foo>
  - What if we need more than one way to compare Foo’s?
Goldilocks and the Hashtable

- A hashtable is a collection of buckets
  - Find the right bucket and search it
  - Bucket organization?
    - Array, linked list, search tree

Structuring Data: The inside story

- How does a hashtable work? (see ArrayListHash.java)
  - What happens with `put(key, value)` in a HashMap?
  - What happens with `getValue(key)`?
  - What happens with `remove(key)`?

```
ArrayList<ArrayList<Combo>> myTable;
public void put(String key, int value) {
    int bucketIndex = getHash(key);
    ArrayList<Combo> list = myTable.get(bucketIndex);
    if (list == null) {
        list = new ArrayList<Combo>();
        myTable.set(bucketIndex, list);
    }
    list.add(new Combo(key, value));
    mySize += 1;
}
```

Hashing: Log (10^{100}) is a big number

- Comparison based searches are too slow for lots of data
  - How many comparisons needed for a billion elements?
  - What if one billion web-pages indexed?
- Hashing is a search method: average case O(1) search
  - Worst case is very bad, but in practice hashing is good
  - Associate a number with every key, use the number to store the key
    - Like catalog in library, given book title, find the book
- A hash function generates the number from the key
  - Goal: Efficient to calculate
  - Goal: Distributes keys evenly in hash table
Hashing details

- There will be collisions, two keys will hash to the same value
  - We must handle collisions, still have efficient search
  - What about birthday "paradox": using birthday as hash function, will there be collisions in a room of 25 people?
- Several ways to handle collisions, in general array/vector used
  - Linear probing, look in next spot if not found
    - Hash to index h, try h+1, h+2, ..., wrap at end
    - Clustering problems, deletion problems, growing problems
  - Quadratic probing
    - Hash to index h, try h+1^2, h+2^2, h+3^2, ..., wrap at end
    - Fewer clustering problems
  - Double hashing
    - Hash to index h, with another hash function to j
    - Try h, h+j, h+2j, ...

Chaining with hashing

- With n buckets each bucket stores linked list
  - Compute hash value h, look up key in linked list table[h]
  - Hopefully linked lists are short, searching is fast
  - Unsuccessful searches often faster than successful
    - Empty linked lists searched more quickly than non-empty
    - Potential problems?
- Hash table details
  - Size of hash table should be a prime number
  - Keep load factor small: number of keys/size of table
  - On average, with reasonable load factor, search is O(1)
  - What if load factor gets too high? Rehash or other method

Hashing problems

- Linear probing, hash(x) = x, (mod tablesize)
  - Insert 24, 12, 45, 14, delete 24, insert 23 (where?)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>24</td>
<td>45</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Same numbers, use quadratic probing (clustering better?)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>24</td>
<td>14</td>
<td>45</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- What about chaining, what happens?

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>24</td>
<td>14</td>
<td>45</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What about hash functions

- Hashing often done on strings, consider two alternatives

```java
public static int hash(String s)
{
    int k, total = 0;
    for(k=0; k < s.length(); k++){
        total += s.charAt(k);
    }
    return total;
}
```

- Consider total += (k+1)*s.charAt(k), why might this be better?
  - Other functions used, always mod result by table size
- What about hashing other objects?
  - Need conversion of key to index, not always simple
  - Ever object has method hashCode()!