Inheritance and Interfaces

- Inheritance models an "is-a" relationship
  - A dog is a mammal, an ArrayList is a List, a square is a shape, ...
- Write general programs to understand the abstraction, advantages?
  ```java
  void execute(Pixmap target) {
    // do something
  }
  ```
- But a dog is also a quadruped, how can we deal with this?

Comparable and Comparator

- Both are interfaces, there is no default implementation
  - Contrast with .equals(), default implementation?
  - Contrast with .toString(), default?
- Where do we define a Comparator?
  - In its own .java file, nothing wrong with that
  - Private, used for implementation and not public behavior
    - Use a nested class, then decide on static or non-static
    - Non-static is part of an object, access inner fields
- How do we use the Comparator?
  - Sort, Sets, Maps (in the future)
- Does hashing (future topic) have similar problems?

Single inheritance in Java

- A class can extend only one class in Java
  - All classes extend Object --- it's the root of the inheritance hierarchy tree
  - Can extend something else (which extends Object), why?
- Why do we use inheritance in designing programs/systems?
  - Facilitate code-reuse (what does that mean?)
  - Ability to specialize and change behavior
    - If I could change how method foo() works, bar() is ok
  - Design methods to call ours, even before we implement
    - Hollywood principle: don't call us, ...

Sets

- Set is an unordered list of items
  - Items are unique! Only one copy of each item in set!
- We will use two different implementations of sets
  - TreeSet
    - A TreeSet is backed up by a tree structure (future topic)
    - Keeps items sorted (+)
    - Slower than HashSets ?? (-)
  - HashSet
    - A HashSet is backed up by a hashing scheme (future topic)
    - Items not sorted – should seem to be in random order (-)
    - Faster than TreeSets ?? (+)
Using Both ArrayList and Sets

- You may want to use a set to get rid of duplicates, then put the items in an ArrayList and sort them!
- **Problem:**
  - Often data comes in the form of an array
  - How do we go from array to ArrayList or TreeSet?
- **Problem:**
  - Often we are required to return an array
  - How do we go from a Collection such as an ArrayList or TreeSet to an array?
- **Can do it the “hard” way with loops or iterators:**
  - one item at a time
- **OR:**

Possible solutions

1. **Use heavy duty data structures (Knuth)**
   - Hash tries implementation
   - Randomized placement
   - Lots’ pointers
   - Several pages
2. **UNIX shell script (Doug McIroy)**
   - `tr -cs "[:alpha:]" "\n*" < FILE | ` sort | ` uniq -c | ` sort -n -r -k 1,1 | ` head -20
   - **Which is better?**
     - K.I.S.?

Data processing example

- Scan a large (~ 10^7 bytes) file
- Print the 20 most frequently used words together with counts of how often they occur
- Need more specification?
- **How do you do it?**

Dropping Glass Balls

- **Tower with N Floors**
- **Given 2 glass balls**
- **Want to determine the lowest floor from which a ball can be dropped and will break**
- **How?**

- **What is the most efficient algorithm?**
- **How many drops will it take for such an algorithm (as a function of N)?**
Glass balls revisited (more balls)

- Assume the number of floors is 100
- In the best case, how many balls will I have to drop to determine the lowest floor where a ball will break?

1. 1
2. 2
3. 10
4. 16
5. 17
6. 18
7. 20
8. 21
9. 51
10. 100

If there are \( n \) floors, how many balls will you have to drop? (roughly)

What is big-Oh about? (preview)

- Intuition: avoid details when they don’t matter, and they don’t matter when input size \( (N) \) is big enough
  - For polynomials, use only leading term, ignore coefficients
    
    \[ y = 3x \quad y = 6x - 2 \quad y = 15x + 44 \]
    
    \[ y = x^2 \quad y = x^2 - 6x + 9 \quad y = 3x^2 + 4x \]

- The first family is \( \mathcal{O}(n) \), the second is \( \mathcal{O}(n^2) \)
  - Intuition: family of curves, generally the same shape
  - More formally: \( \mathcal{O}(f(n)) \) is an upper-bound, when \( n \) is large enough the expression \( cf(n) \) is larger
  - Intuition: linear function: double input, double time, quadratic function: double input, quadruple the time

More on O-notation, big-Oh

- Big-Oh hides/obscures some empirical analysis, but is good for general description of algorithm
  - Allows us to compare algorithms in the limit
    - \( 20N \) hours vs \( N^2 \) microseconds: which is better?
  - \( \mathcal{O} \) notation is an upper-bound, this means that \( \mathcal{N} = \mathcal{O}(N) \), but it is also \( \mathcal{O}(N^2) \); we try to provide tight bounds.
    - Formally:
      - A function \( g(N) = \mathcal{O}(f(N)) \) if there exist constants \( c \) and \( n \) such that \( g(N) < cf(N) \) for all \( N > n \)

Which graph is “best” performance?
Big-Oh calculations from code

- Search for element in an array:
  - What is complexity of code (using O-notation)?
  - What if array doubles, what happens to time?
  ```java
  for(int k=0; k < a.length; k++) {
    if (a[k].equals(target)) return true;
  }
  return false;
  ```
- Complexity if we call $N$ times on $M$-element vector?
  - What about best case? Average case? Worst case?

Some helpful mathematics

- $1 + 2 + 3 + 4 + ... + N$
  - $N(N+1)/2$, exactly $= N^2/2 + N/2$ which is $O(N^2)$ why?
- $N + N + N + ... + N$ (total of $N$ times)
  - $N*N = N^2$ which is $O(N^2)$
- $N + N + N + ... + N + ... + N$ (total of 3$N$ times)
  - $3N*N = 3N^2$ which is $O(N^2)$
- $1 + 2 + 4 + ... + 2^N$
  - $2^{N+1} - 1 = 2 * 2^N - 1$ which is $O(2^N)$
- Impact of last statement on adding $2^N+1$ elements to a vector
  - $1 + 2 + ... + 2^N + 2^{N+1} = 2^{N+2}-1 = 4x2^N-1$ which is $O(2^N)$
    - resizing + copy = total (let $x = 2^N$)

Amortization: Expanding ArrayLists

- Expand capacity of list when add() called
- Calling add N times, doubling capacity as needed

<table>
<thead>
<tr>
<th>Item #</th>
<th>Resizing cost</th>
<th>Cumulative cost</th>
<th>Resizing Cost per item</th>
<th>Capacity After add</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3-4</td>
<td>4</td>
<td>6</td>
<td>1.5</td>
<td>4</td>
</tr>
<tr>
<td>5-8</td>
<td>8</td>
<td>14</td>
<td>1.75</td>
<td>8</td>
</tr>
</tbody>
</table>

2$^{m+1} - 2^{m+1}$

Running times @ $10^6$ instructions/sec

<table>
<thead>
<tr>
<th>$N$</th>
<th>$O(\log N)$</th>
<th>$O(N)$</th>
<th>$O(N \log N)$</th>
<th>$O(N^2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.00001</td>
<td>0.0001</td>
<td>0.000033</td>
<td>0.0001</td>
</tr>
<tr>
<td>100</td>
<td>0.000007</td>
<td>0.00010</td>
<td>0.000064</td>
<td>0.1000</td>
</tr>
<tr>
<td>1,000</td>
<td>0.000010</td>
<td>0.01000</td>
<td>0.010000</td>
<td>1.0</td>
</tr>
<tr>
<td>10,000</td>
<td>0.000013</td>
<td>0.01000</td>
<td>0.132900</td>
<td>1.7 min</td>
</tr>
<tr>
<td>100,000</td>
<td>0.000017</td>
<td>0.10000</td>
<td>1.661000</td>
<td>2.78 hr</td>
</tr>
<tr>
<td>1,000,000</td>
<td>0.000020</td>
<td>1.0</td>
<td>19.9</td>
<td>11.6 day</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>0.000030</td>
<td>16.7 min</td>
<td>18.3 hr</td>
<td>318 centuries</td>
</tr>
</tbody>
</table>