Lecture 07a
Object Oriented Programming
Oct. 3rd, 2017

Announcements
- Quiz #6 due Tuesday (before class)
- PA4 is due next Tuesday (counts double)
- Recitations this week: mandatory
  - Valgrind (profiling)
  - Next week: GoogleTest
- Midterm #1 was returned last Thursday
  - I am tossing out the gamma questions, but…
  - If you got points, you got to keep them
  - With curve, no change

PA4
- Go to assignment…
- Answer any questions

PA4 Hints
- PA4 is complex
  - No single rule/definition/etc. is that complex
  - But their combination is challenging
  - Programming is the art of managing complexity

- Hint #1: No method longer than 15 lines
  - Including parameters, brackets, etc.
  - Avoids spaghetti code
  - Manageable chunks

- Hint #2: Lots of (short) methods
  - Each has a meaningful name
    - Most important of all forms of documentation
    - If you don’t know how to name it, it’s not a good abstraction…
  - Manipulate parameters (mostly)
    - Many const methods (accessors)
    - Can access class fields
  - My code has:
    - A method for every definition
    - A method for every stemming step
    - And more….

- Hint #3: Define Classes
  - Multiple types of data:
    - Files
    - Strings (words, punctuation)
    - Documents (sequences of strings)
    - Frequencies
  - Encapsulation & Abstraction
  - I have three classes:
    - Document (sequence of strings, frequencies, file reader)
    - Lexeme (strings)
    - Stemmer (because otherwise, document too big)
Welcome to CS253, Part 2

- **Object Oriented Programming**
  - Encapsulation
    - Group data & code
    - The heart of object oriented design
  - Abstraction via inheritance
    - Inheritance as union
    - How are objects implemented?
    - Multiple inheritance
  - Dispatch
  - Testing starts with recitation on GoogleTest
  - Next week

---

**Object Oriented Programming**

- Why do we use objects?
  - Encapsulation
    - Collect related data & code together
  - Polymorphism
    - Goal: abstraction
      - Quaggas have properties
      - Some properties shared by all Equines
      - Some shared by all Mammals
      - Or all Animals
    - Write methods at most abstract level possible
      - Avoids repeating code

---

**OO: C++ vs Java**

- In C++, objects are optional
  - In Java, all functions are methods of a class
  - In C++, they don't have to be

  - When should you create an object?
    - Complex data supports unique operations
      - Encapsulation
    - When it shares properties/code with a more abstract form
      - Polymorphism
    - Efficiency should not be a consideration

---

**OO: C++ vs Java (cont.)**

- Java has an inheritance tree
  - Every class has exactly one parent
  - Object is the root of all classes
- C++ has an inheritance forest
  - No unique root
  - Many classes have no parent
  - A few classes have multiple parents
  - Inheritance relations form a directed acyclic graph

---

**When do you write stand-alone functions?**

- Not manipulating state
  - Parameters are the only data touched
- No side-effects
  - Output value is the only result
  - No other state is changed (including parameters)
- Example: sqrt
- Functions support parallelism
  - Move function to parameters (no other state)
  - Apply two functions to same parameters (no side effects)

---

**Syntax of Inheritance**

```cpp
class Quagga : public Equine 
{
  ...
};
```

- Choice of three inheritance levels:
  - Public: everything has the same access privileges in the child as the parent
  - Protected: public fields of parent are protected in child
  - Private: everything is private to the parent
- Always use public inheritance
  - Haven't found many uses for the rest
What is inheritance?

- If class B inherits class A, what does that mean?
  - Inheritance is union
    - Every field in A is included in B
    - Every method in A is included in B
  - B then adds its own fields and methods
    - Note: you cannot inherit part of a parent
- Inheritance is mandatory in Java
  - Every class has a parent
  - Every object has extra data
- Inheritance is optional in C++
  - Most classes have no parent
  - Makes inheritance “lighter”
  - Makes run-time typing harder

Inheritance vs Inclusion (Example)

- I have a lexeme class
  - Stores a string (word or punctuation)
  - Also, other values (count, type)
  - Should it inherit string?
  - Or just include a field of type string?
  - Why?
- Remember: inheritance is union
  - Can I concatenate two lexemes?
  - It might not have a type (word vs punctuation)
  - In this case, inclusion seems better