CompSci 100E

Dietolf (Dee) Ramm

http://www.cs.duke.edu/courses/cps100e/spring05

http://www.cs.duke.edu/~dr
What is Computer Science?

What is it that distinguishes it from the separate subjects with which it is related? What is the linking thread which gathers these disparate branches into a single discipline? My answer to these questions is simple --- it is the art of programming a computer. It is the art of designing efficient and elegant methods of getting a computer to solve problems, theoretical or practical, small or large, simple or complex.

C.A.R. (Tony) Hoare
Programming != Computer Science

- What is the nature of intelligence? How can one predict the performance of a complex system? What is the nature of human cognition? Does the natural world 'compute'?

- It is the interplay between such fundamental challenges and the human condition that makes computer science so interesting. The results from even the most esoteric computer science research programs often have widespread practical impact. Computer security depends upon the innovations in mathematics. Your Google search for a friend depends on state-of-the-art distributed computing systems, algorithms, and artificial intelligence.

Efficient *design, programs, code*

Using the language: Java (or C++, or Python, or ...), its idioms, its idiosyncracies

Know data structures and algorithms. Trees, hashing, binary search, sorting, priority queues, greedy methods, ...

Object-oriented design and patterns. Software design principles transcend language, but ...

Engineer, scientist: what toolkits do you bring to programming? Mathematics, design patterns, libraries --- standard and Duke CPS
Course Overview

- **Lectures, Labs, Quizzes, Programs**
  - Lectures based on readings, questions, programs
    - Online quizzes used to motivate/ensure reading
    - In-class questions used to ensure understanding
  - Programs
    - Theory and practice of data structures and OO programming
    - Fun, practical, tiring, …
    - Weekly programs and longer programs
  - Labs based on current work
    - Get in practical stuff
    - Become familiar with tools

- **Exams/Tests (closed book)**
  - Two “midterms”
  - Final
Questions

If you gotta ask, you’ll never know
Louis Armstrong: “What’s Jazz?”

If you gotta ask, you ain’t got it
Fats Waller: “What’s rhythm?”

What questions did you ask today?
Arno Penzias
Tradeoffs

Simple, elegant, quick, efficient: what are our goals in programming? What does XP say about simplicity? Einstein?

Fast programs, small programs, run anywhere-at-all programs. Runtime, space, your time, CPU time...

How do we decide what tradeoffs are important? Tension between generality, simplicity, elegance, …
OO design in code/wordcount

- Count number of different words in an array, how can we accommodate more than one approach?

```java
public interface UniqueCounter {
    public int uniqueCount(String[] list);
}
```

- Three (or more) approaches:
Fast, cheap, out-of-control?

- This is valid and correct Java code, questions?

```java
import java.util.*;

public class SetUniqueCounter
    implements UniqueCounter {

    public int uniqueCount(String[] list) {
        TreeSet set = new TreeSet();
        set.addAll(Arrays.asList(list));
        return set.size();
    }

}
Some Java / Matlab Differences

- **Compile & Execute vs Interactive**
  - In Java, *compile*, then run (*execute*) – like `.m` files
  - Matlab executes as you type in program
- **Java requires declaration of variables**
  - Need to tell about the variable before creating
  - Declaration is distinct from Definition (creation)
- **Java is not matrix oriented**
  - Operators (+, -, *, /, %), do not work on matrices
  - You must write code with loops for matrix operations
  - - or use functions (methods)
Some Java / Matlab Differences

- **No exponentiation operator**
  - Cannot say $x^3$ for $X^3$
  - Use $x*x*x$ or a function

- **Syntax differences**
  - Use of braces, `{ ... }`, in place of `xxx ... end`
  - Semicolon has somewhat different meaning
  - Use quotes, " ... ", for strings not ’ ... ’
  - Loops and if require parentheses ( ... )

- **You’ll find many more differences**
  - Will be an annoying, but transient problem
Some Java Vocabulary and Concepts

- **Java has a huge standard library**
  - Organized in *packages*: `java.lang`, `java.util`, `javax.swing`, ...
  - API browseable online, but Eclipse IDE helps a lot

- **Java methods have different kinds of access inter/intra class**
  - Public methods ...
  - Private methods ...
  - Protected and Package methods ...

- **Primitive types** (`int`, `char`, `double`, `boolean`) are not objects but everything else is literally an *instance* of class `Object`
  - `foo.callMe();`
Basic data structures and algorithms

- Arrays are typed and fixed in size when created
  - Not like vector in C++
  - Don't have to fill the array, but cannot expand it
  - Can store int, double, String, Foo, ...

- ArrayList (and related class Vector and interface List) grows
  - Stores objects, not primitives
  - Accessing elements can require a downcast
  - ArrayList objects grow themselves intelligently

- java.util package has lots of data structures and algorithms
  - Use rather than re-implement, but know how do to do both
Tracking different/unique words

- We want to know how many times ‘the’ occurs
  - Do search engines do this? Does the number of occurrences of “basketball” on a page raise the priority of a webpage in some search engines?
    - Downside of this approach for search engines?

- Constraints on solving this problem
  - We must read every word in the file (or web page)
  - Search for the word? Avoid counting twice? Store?

  - Are there fundamental limits on any of these operations? Where should we look for data structure and algorithmic improvements?
What does it try to do? Why is it wrong?

```java
public class SlowUniqueCounter implements UniqueCounter{

    public int uniqueCount(String[] list) { 
        int count = 0;
        int diffSize = list.length;
        for(int k=0; k < diffSize; k++){
            String word = list[k];
            count++;
            for(int j=k+1; j < diffSize; j++){
                if (list[j].equals(word)){
                    list[j] = list[diffSize-1];
                    diffSize--;
                }
            }
        }
        return count;
    }
}
```
Search: measuring performance

- How fast is fast enough?

```java
/** pre: a contains a.size() entries
* post: return true if and only if key found in a
*/
boolean search(ArrayList a, String key)
{
    for(int k=0; k < a.size(); k++)
        if (a[k].equals(key)) return true;
    return false;
}
```

- Java details: parameters? Return values? ArrayLists?
- How do we measure performance of code? Of algorithm?
  - Does processor make a difference? G5? Itanium? 64-bit?
Tradeoffs in processing and counting

- Read words, then sort, determine # unique words?
  - frog, frog, frog, rat, tiger, tiger, tiger, tiger

- If we look up words one-at-a-time and bump counter if we haven't seen a word, is this slower than previous idea?
  - How do we look up word, how do we add word

- Are there kinds of data that make one approach preferable?
  - What is best case, worst case, average case?
Benefits of inheritance, interfaces

Consider new algorithm for determining unique word count

```java
public static void test(UniqueCounter uc, String[] list) {
    double start = System.currentTimeMillis();
    int count = uc.uniqueCount(list);
    double end = System.currentTimeMillis();
    System.out.println(count + " unique words");
    System.out.println((end - start) / 1000 + " seconds");
}
```

Why can we pass different kinds of objects to `test`?

- Why is this an advantage?
- Inheritance and late/dynamic binding
Why inheritance?

- Add new shapes easily without changing much code
  - Shape s1 = new Circle();
  - Shape s2 = new Square();

- Interface/abstract base class:
  - interface or abstraction
  - Function called at runtime

- concrete subclass
  - All abstract functions implemented
  - Later we'll override

- “is-a” view of inheritance
  - Substitutable for, usable in all cases as-a

User’s eye view: think and program with abstractions, realize different, but conforming implementations,
don’t commit to something concrete until as late as possible
Example of inheritance

- What is behavior of a shape?

```java
void doShape(Shape s) {
    System.out.println(s.area());
    System.out.println(s.perimeter());
    s.expand(2.0);
    System.out.println(s.area());
    System.out.println(s.perimeter());
}

Shape s1 = new Circle(2);
Shape s2 = new Square(4);
Shape s3 = new Rectangle(2,5);
doShape(s1); doShape(s2); doShape(s3);
```
Inheritance (language independent)

- **First view: exploit common interfaces in programming**
  - Iterators in Java or C++/Tapestry
  - Implementation varies while interface stays the same

- **Second view: share code, factor code into parent class**
  - Code in parent class shared by subclasses
  - Subclasses can *override* inherited method
    - Subclasses can override and call

- **Polymorphism/late(runtime) binding (compare: static)**
  - Function actually called determined when program runs, not when program is compiled
Who is Alan Perlis?

- It is easier to write an incorrect program than to understand a correct one
- Simplicity does not precede complexity, but follows it
- If you have a procedure with ten parameters you probably missed some
- If a listener nods his head when you're explaining your program, wake him up
- Programming is an unnatural act
- Won first Turing award

http://www.cs.yale.edu/homes/perlis-alan/quotes.html