CompSci 100E
http://www.cs.duke.edu/courses/spring07/cps100e

Dietolf (Dee) Ramm
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

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

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

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

Programming, design, algorithmic, data-structural

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

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

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 \( x \times x \times x \) or a function

- Syntax differences
  - Use of braces, \{ \ldots \}, in place of \( xxx \ldots \text{end} \)
  - Semicolon has somewhat different meaning
  - Use quotes, " \ldots "', for strings not ' \ldots '
  - Loops and if require parentheses ( \ldots )

- 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();
Java Basics *(ala Goodrich & Tamassia)*

- Everything is in a class
- A minimal program:

```java
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello Computer Science");
    }
}
```

- Output?
- Where?
- Do colors mean something?

- Identify the pieces . . .
Java Basics

Objects

- Every object is an *instance* of a class (which defines its type)
- Objects contain data (state) and function (methods)

State

- Stored in *instance variables* (fields, members)
- Can be base types (e.g., integers)
  or instances of (objects) of other classes (e.g., String)

Function

- Expressed as *methods* (subroutines, functions, procedures…)
- These define the behavior of objects of this class
Java Basics

- Declaring a Class:
  ```java
  public class Counter {
    protected int count;
    Counter() {
      count = 0;
    }
    public int getCount() {
      return count;
    }
    public void incCount() {
      count = count + 1;
    }
    public void decCount() {
      count = count - 1;
    }
  }
  ```

- Identify the methods by kind
  - Constructor
  - Accessor
  - Mutator (modifier)

- Note Syntax from this and previous examples
  - Braces
  - Semicolons
  - Parentheses
  - Identifiers
  - . . .
Java Basics

- **Class Modifiers**
  - Abstract, final, public, default

- **Reserved Words**
  - May not be used as identifiers
  - Shown in red by Eclipse and in many of our examples
  - See table in text (p4) or in any Java text

- **Comments**
  - For *human consumption*: ignored by compiler
  - Inline comments: //
    - Effective for rest (to end) of current line
  - Block comments: /* */
    - Effective between *start* and *stop* groups
Java Basics

- **Primitive Types (base types)**
  - Built-in data types; native to most hardware
  - Note: *not objects* (will use mostly first four)
    - `boolean` (1 bit)
    - `int` (4 bytes)
    - `double` (8 bytes)
    - `char` (2 bytes)
- **Constants/Literals (by example):**
  - `boolean f = false;`
  - `int i = 32769;`
  - `double d = 0.333333;`
  - `char c = 'x';`
  - `byte b = 33;`
  - `short s = 21;`
  - `long l = 289L;`
  - `float f = 3.141592F;`
Java Basics

- Creating and Using Objects (Example)

```java
public class Example {
    public static void main(String[] args) {
        Counter c; // Counter defined on a previous slide
        Counter d = new Counter();
        c = new Counter();
        System.out.println("c = " + c.getCount()
                + " d = " + d.getCount());
        c.incCount();
        d.decCount();
        System.out.println("c = " + c.getCount()
                + " d = " + d.getCount());
        d = c;       // what does this really mean???
        c.incCount();
        d.incCount();
        System.out.println("c = " + c.getCount()
                + " d = " + d.getCount());
    }
}
```
Java Basics

- **String Objects**
  - string is a sequences of characters (`char`)
    - Unicode (16 bit)
  - `String` is a built-in class
    - Constants: "this is an example"

- **String Concatenation (+)**

```java
String s = "Happy birthday to you.";
s = s + "\n" + s;
System.out.println(s); // what ?
```
Java Basics

- **Object References**
  - When creating object with `new`, get *location* or *address* of new object
  - Typically assign this to a reference variable:
    ```java
    Counter c = new Counter();
    ```
  - Every object reference variable refers to object or `null`
  - Null is an important value that indicates object not created or not available.
  - Can have multiple references to same object
  - Access members of class using *dot operator* (" . ").
    ```java
    Counter c = new Counter();
    c.incCount();
    ```
  - May have multiple methods with same name but different *signature*: e.g.: `c.incCount(); c.incCount(5);`
Java Basics

- **Instance Variables**
  - Classes have 0 or more instance variables
    - Also called *fields*
    - Keep state of object
  - May be primitive type
    - E.g. `int, double`
  - May be reference type (object)
    - E.g., String, Counter, (an array), . . .
  - If public can:
    - Access or alter reference variables using dot operator
      ```java
      Counter c = new Counter();
      System.out.println(c.count + " = " +
                         c.getCount());
      ```
Java Basics

- Variables Modifiers: scope
  - public
    - Anyone can access
  - protected
    - Only subclass or same package may access
  - private
    - Only methods of same class may access
  - (omitted) default
    - Anyone in same package may access

- Other Variable Modifiers
  - static
    - Associated with whole class, shared among instances
  - final
    - Must be initialized, then not changed: CONSTANT
Java Basics - Methods

- **Methods**
  - Like functions, procedure, subroutines, ...
  - Has *header* and *body*
  - Syntax:
    ```java
    modifiers type name(parameter_declarations){
    method_body
    }
    ```
  - **Modifiers** like those of variables:
    - `public, private, protected, static, final`
  - **Type** is return type and give type of information being passed back
  - **Name** is any valid Java identifier name
  - **Parameters** define type of info being passed into method
Java Basics - Methods

❖ Method modifiers
  ❙ **public**: anyone can invoke (call)
  ❙ **protected**: only called from subclass of same package
  ❙ **private**: only called from same class
  ❙ (omitted) (default): only called from same package

  ❙ **abstract**: has no code (must be dealt with in subclass)
  ❙ **final**: cannot be overridden (modified) in subclass
  ❙ **static**: associated with class, not with instance

❖ Return types
  ❙ Use **void** is no information to be returned (*procedure*)
  ❙ Use actual type of information to be returned (*function*)
    o requires **return** statement(s)
    o only **one** item returned (may be compound object, e.g., array)
Java Basics - Methods

- **Parameters**
  - Parameter list may be empty (`parentheses` still **required**).
  - Parameter list consists of comma separated pairs of types and parameter names.
    ```java
    public void setAge(String name, int age){...}
    ```

- **Constructors**
  - Used to initialize new objects
  - Has **same name as class** and **no return type**
    ```java
    public Counter() {
        count = 0;
    }
    
    public Professor(String aName, String aDept){
        name = aName;
        department = aDept;
    }
    ```
Java Basics

❖ Using a Constructor
  ❑ Invoked using a new operator
    ○ Examples:
      ```java
      Professor compSciProf = 
        new Professor("Jeff Chase", "Computer Science");
      Counter tally = new Counter();
      ```
  ❑ Class may have multiple constructors as long as signatures are different
  ❑ If class has no constructors defined, then a default constructor is used that does not initialize anything
Java Basics - Methods

- **The main Method**
  - Required for an *Application*
    - This is a stand-alone Java program
    - Typically invoked from a command line
    - Must include the following code:
      ```java
      public static void main(String[] args) {
        // main body of the main method
      }
      ```
    - (The parameter name `args` can actually be any name you choose.)
    - Argument may be used to pass command line arguments to the program.
Java Basics - Methods

- **Blocks and Local Variables**
  - Body of a method is a **block**: a sequence of statements and declarations enclosed in braces (`{ }`);
    - Blocks may have blocks nested inside
    - Variables declared with a block are known **only** in that block
    - These variables are called **local variables**
    - (We say their **scope** is limited to that block.)
    - (Method parameters are also local to that method.)
  - Examples:
    ```java
    public static int sumThree(int a, int b, int c){
      int sum;
      int partsum = a + b;
      sum = partsum + c;
      return sum;
    }
    ```
    - `a`, `b`, `c`, `sum`, and `partsum` are all **local** to that method