Most people find the concept of programming obvious, but the doing impossible.

It is easier to write an incorrect program than understand a correct one.

There are two ways to write error-free programs; only the third one works.

To understand a program you must become both the machine and the program.

To understand a program you must become both the machine and the program.

It is easier to change the specification to fit the program than vice versa.

The first seven verses printed by your program must exactly reproduce the output at left. This includes identical wording, spelling, spacing, punctuation, and capitalization. However, to encourage creativity, the last verse of your song (the final held part in 3 or 4 measure) may print any text you like. Creative verses submitted may be shown in class.

Two goals of software system building

Barry Boehm

Building the right system

- Does the program meet the users’ needs?
- Determining if this is true is usually called validation

Building the system right

- Does the program meet the specification?
- Determining if this is true is usually called verification

Two goals of software system building

Barry Boehm

Building the right system

- Does the program meet the users’ needs?
- Determining if this is true is usually called validation

Building the system right

- Does the program meet the specification?
- Determining if this is true is usually called verification

In CSE331, the second goal is the focus – that is, we focus almost only on creating a correctly functioning artifact.

It can be surprisingly hard to specify, design, implement, test, debug and maintain even a simple program.

A familiar kind of specification

A CSE 143 assignment

// Interface Queue defines a set of operations for manipulating a FIFO // (First In First Out) structure that can be used to store elements // of type E.

Program Description:
This program tests your understanding of static methods and print statements. Write a Java class called

public interface Queue<E> { // post: given value inserted at the end of the queue
  public void enqueue(E value); // pre: value is inserted at the end of the queue
  // post: removes and returns the value at the front of the queue
  public E dequeue();
  // post: returns true if the queue is empty, false otherwise
  public boolean isEmpty();
  // post: returns the current number of elements in the queue
  public int size();
}

Unity is the glue that makes anything pasted together stick. This is the glue that makes anything pasted together stick. This is the glue that makes anything pasted together stick. This is the glue that makes anything pasted together stick.
### Specification Jeopardy: Hello World

- Prints "Hello, World"
- Without quotation marks
- Prints any string starting with "H"
- Does anything as long as it terminates
- ... 

- Every program meets (we'll usually say satisfies) an unbounded number of specifications
- It can be tricky to accurately specify the "right" amount of information – it takes experience

- Does anything

- Prints any string with 12 characters in it

- Prints any string without quotation marks

- Prints "Hello, World"

- Prints any integer

- Prints twice the first input argument OR throws java.lang.NumberFormatException

- Prints twice the first input argument otherwise throws java.lang.NumberFormatException

- … don't forget MIN_VALUE ...

### Specification Jeopardy: Double

```java
public static void main(String[] args) {
    System.out.println(args[0] + 2);
}
```

- Prints twice the first input argument
- Prints any integer
- Print argument ≥ 0
- Pre: input argument ≥ 0
- Post: prints any non-negative integer
- Post: prints twice the first input argument
- Post: if input argument * 2 ≤ java.lang.Integer.MAX_VALUE
- Otherwise throws java.lang.NumberFormatException
- ... don't forget MIN_VALUE ...

### And more

```java
public static void main(String[] args) {
    System.out.println(Integer.parseInt(args[0]) * 2);
}
```

- No input argument?
- Non-integer input argument?
- Should it throw java.lang.NumberFormatException?
- More than one input argument?

- Again, it's tricky, requiring careful case analysis – imaginable inputs? desired outputs? what is really intended? how is it intended to be used? ...
- And, again, it takes experience (which should not be mistaken for intelligence)

### Specifications: Q&A instead of A&Q

- Is it easier to simply write down specifications before the program exists?
- What would a post-condition be for a method that sorts an integer array of length N in non-decreasing order?
- How do you precisely describe when an array is sorted?

**In small groups, spend 2-3 minutes sketching a sorting post-condition (any syntax is OK)**

### A flaw

- Most groups probably found a post-condition like:
  ```java
  [i,j] \in [0,N-1] \land i < j \Rightarrow A[i] \leq A[j]
  ```
- That is, for any two elements in the array, the one with the lower index must not be greater than the one with the higher index.
- But "undesired" satisfying programs are also allowed:
  ```java
  for(int i=0; i<N; i++)
    A[i] := i;
  ```
- Sorting means reordering the original elements, not simply creating a sorted array.
- So, we need to add a clause to the post-condition that says:
  ```java
  A is a permutation of A_orig
  ```

### "Incomplete" specifications: sqrt

```java
double sqrt(double x, double epsilon)
pre: x ≥ 0
post: abs(return - return - x) < epsilon
```
Getting to “correct”

- There is no precise notion of a “correct” specification.
- However, there can be incorrect specifications – ones that cannot be implemented, ones that are inconsistent, and ones that are just bad…
- This is really a validation question, “Does the specification meet the needs of the users?”
- This is because there is no precise way to assess user needs – although there is a lot known about this, it is far beyond what we can cover in 331.
- So we will focus on a precise notion – does a given program (implementation) satisfy a given specification?

Why is writing satisfying programs hard?

- Many ways to satisfy a specification – have to find and choose a “good” way.
  - “Goodness” is an ill-defined mix of customer needs, business needs, technologies, etc.
- Software systems are complex.
  - Many difficult decisions and structures
  - Requires teams that effectively communicate and coordinate.
- Customer needs evolve.
  - Programs must change to meet spec changes
  - …and more, much more…!
- “Software entities are more complex for their size than perhaps any other human construct, because no two parts are alike…” If they are, we make the two similar parts into one… In this respect software systems differ profoundly from computers, buildings, or automobiles, where repeated elements abound.” Fred Brooks: “No Silver Bullet — Essence and Accidents of Software Engineering”, IEEE Computer 20 (4): 10–19 (1987).

The Tao (道) of CSE331

- Programming (in Java)
  - Control (loops, conditionals, methods, parameter passing, recursion, etc.)
  - Variables
  - Abstract data types (ADTs):
  - Stacks, linked lists, …
  - Interfaces, inheritance and encapsulation
  - Basics of complexity and performance tradeoffs
  - Using off-the-shelf components from Java Collections

Designing and implementing more realistic software (in Java, but more general)
- Abstraction and specification
- Writing, understanding and reasoning about code
- Program design and documentation: process and tools
- What makes a design good or bad?
- Pragmatic considerations
- Testing
- Debugging and defensive programming
- Software management issues

An observation

- Some of you are eyeballing others, worrying that “they know stuff I don’t know”
- This is likely true but
  - You also know stuff they don’t and
  - Nobody was born knowing about Java generics or version control or Eclipse or such… (who knows it) had to learn it sometime.

If you are having trouble…

- “Ask” yourself what’s going on – programming by permutation rarely succeeds, so think first!
- Look for information – it can be hard, but learning how to do this effectively is a great investment.
- Ask others for help – course staff, friends, students in the class, etc.
- The one epic fail is to stay stuck on something for a long time

DON’T DO THIS! Really. I’m totally serious. Really.

Performance

- How fast a program runs can be important – and we may at times talk about performance.
- But it is not even close to the primary focus of 331 – correctness is much more important.
- These quotations about performance are from people with extraordinary experience and insight

Michael Jackson
- Rule 1: Don’t do it.
- Rule 2 (for experts only): Don’t do it yet.

Bill Wulf
- More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason – including blind stupidity.

Donald Knuth
- We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil.
An old conversation, oft-repeated

You lost points because the program isn’t correct

But it’s fast!

Extra credit if you can identify either or both of these people

Prerequisite: Java at the 142/143 level

Examples

- Sharing
  - Distinction between == and equals()
  - Are two objects the same object, or do they have equal values?
  - And what does “equal” mean?
- Aliasing (multiple references to the same object)
- Subtyping
  - Varieties: classes, interfaces
  - Inheritance and overriding
- Object-oriented dispatch
  - Expressions have a compile-time type
  - Objects/values have a run-time type
- The first two assignments will largely focus on making sure you’re (back) up to speed on this kind of material

Logistics

- http://www.cs.washington.edu/courses/cse331
  - There’s useful information in there … if you can’t find what you’re looking for, make sure to ask!
  - Especially the calendar
  - We’ll also be using piazza.com for information and Q&A threads, etc.
  - And Catalyst as well, although not for posting

Collaboration policy

- Discussion is permitted … indeed, encouraged!
- Representing someone else’s work as your own is not permitted
- Familiarize yourselves with the CSE, COE and UW policies on academic honesty – we rely on them heavily – we will pursue cases of academic dishonesty
- If you have a question about what is allowed, ask!
- Please apply the Gilligan’s rule
  - if you watch mindless TV (Hulu?) for 30-60 minutes after a discussion with classmates, and
  - you can still reproduce the materials from memory (no notes, no email, etc.), then
  - you can consider it your work/knowledge
- Loopholes are not loopholes – spirit of the law applies over the letter of the law

Next steps

- Assignment 0: on the web now, due Friday 11:59PM
  - Objective: get your Java environment configured and running, get a couple of tiny Java programs running, a survey, etc.
  - Warning: this can be quite complicated, with different operating systems (variants of Linux, Windows, Mac OS), different versions of Java, different versions of Eclipse, and more — we have some expertise, but sometimes it’s hard for us, too
  - Sections tomorrow: in the lab, with the staff there, bouncing from student-to-student to help you do this
  - TODAY: Make sure you can login to UW CSE machines! Both Linux and Windows (which can have different passwords!)
  - Forgot CSENetID? Email support@cs (they reset it, you retrieve it at reception)
  - Forgot Windows password but remember CSENetID? Click here
- Assignment 1: on the web before Friday lecture; due next Wednesday 11:59PM
  - Lectures: Specifications (F), testing/JUnit (M), equality (W)