CONCLUSION

History

Programming and computers:
Momentous changes since the 1940s—or since even the use of punch cards and attempt at automation …

Punch cards

Mechanical loom invented by Joseph Marie Jacquard in 1801.
Used the holes punched in pasteboard punch cards to control the weaving of patterns in fabric.
Punch card corresponds to one row of the design.
Based on earlier invention by French mechanic Falcon in 1728.

Jacquard loom

Loom still used in China

Charles Babbage designed a “difference engine” in 1822

Compute mathematical tables for log, sin, cos, other trigonometric functions.

No electricity

The mathematicians doing the calculations were called computers

Oxford English Dictionary, 1971

Computer: one who computes; a calculator, rekoner, spec. a person employed to make calculations in an observatory, in surveying, etc.

1664: Sir T. Browne. The calendars of these computers.
1704. T. Swift. A very skillful computer.
1744. Walpole. Told by some nice computers of national glory.
1855. Brewster Newton. To pay the expenses of a computer for reducing his observations.

The mathematicians doing the calculations were called computers

Charles Babbage planned to use cards to store programs in his Analytical engine. (First designs of real computers, middle 1800s until his death in 1871.)

First programmer was Ada Lovelace, daughter of poet Lord Byron.
Privately schooled in math. One tutor was Augustus De Morgan.

The Right Honourable Augusta Ada, Countess of Lovelace
Herman Hollerith.  His tabulating machines used in compiling the 1890 Census.  Hollerith’s patents were acquired by the Computing-Tabulating-Recording Co.  Later became IBM.

The operator places each card in the reader, pulls down a lever, and removes the card after each punched hole is counted.

Hollerith 1890 Census Tabulator

Computers, calculating the US census

History of computers

1935-38. Konrad Zuse - Z1 Computer
1944. Howard Aiken & Grace Hopper Harvard Mark I Computer
1946. John Presper Eckert & John W. Mauchly ENIAC I Computer 20,000 vacuum tubes later ...
1947-48 The Transistor, at Bell-labs.
1953. IBM. the IBM 701.

1959. Took his only computer course. Senior, Queens College.


Programmed in Fortran and IBM 7090 assembly language

\[
\begin{align*}
\text{if (SEX == 'M') MALES = MALES + 1;} \\
\text{else FEMALES = FEMALES + 1;}
\end{align*}
\]

CLI SEX=M Male?
BNO IS_FEM If not, branch around
L 7 MALES Load MALES into register 7;
LA 7,1(7) add 1;
ST 7 MALES and store the result
B GO_ON Finished with this portion
IS_FEM L 7 FEMALES If not male, load FEMALES into register 7;
LA 7,1(7) add 1;
ST 7 FEMALES and store
GO_ON EQU *

1960: Big Year for Programming Languages

LISP (List Processor): McCarthy, MIT (moved to Stanford). First functional programming language. No assignment statement. Write everything as recursive functions. (take 3110)

COBOL (Common Business-Oriented Language). Became most widely used language for business, data processing.

ALGOL (Algorithmic Language). Developed by an international team over a 3-year period. McCarthy was on it, John Backus was on it (developed Fortran in mid 1950’s). Gries’s soon-to-be PhD supervisor, Fritz Bauer of Munich, led the team.
1959. Took his only computer course. Senior, Queens College.
   John Backus, FORTRAN, mid 1950’s: 30 people years
   Today, CS compiler writing course: 2 students, one semester
1963-66 Dr. rer. nat. in Math in Munich Institute of Technology
1966-69 Asst. Professor, Stanford CS
1969- Cornell!

### Programming languages. Dates approximate

<table>
<thead>
<tr>
<th>Year</th>
<th>Major languages</th>
<th>Teach at Cornell</th>
</tr>
</thead>
<tbody>
<tr>
<td>1956’s</td>
<td>Fortran</td>
<td></td>
</tr>
<tr>
<td>1960</td>
<td>Algol, LISP, COBOL</td>
<td></td>
</tr>
<tr>
<td>1970</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>1972</td>
<td>Pascal</td>
<td></td>
</tr>
<tr>
<td>1980’s</td>
<td>Smalltalk (object-oriented)</td>
<td>Pascal (1980’s)</td>
</tr>
<tr>
<td>1980’s</td>
<td>(late) C++</td>
<td></td>
</tr>
<tr>
<td>1996</td>
<td>Java</td>
<td>C and C++</td>
</tr>
<tr>
<td>2008</td>
<td>Java / Matlab</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td>Python / Matlab / Java</td>
<td></td>
</tr>
</tbody>
</table>

### Late 1960s
Write programs on IBM “punch cards. Deck of cards making up a program trucked to Langmuir labs by the airport 2-3 times a day: get them back, with output, 3-4 hours later

### About 1973. BIG STEP FORWARD
1. Write program on punch cards.
2. Wait in line (20 min) to put cards in card reader in Upson basement
3. Output comes back in 5 minutes

### About 1979. Teraks
Prof. Tim Teitelbaum sees opportunity. He and grad student Tom Reps develop "Cornell Program Synthesizer". Year later, Cornell uses Teraks in its prog course.

### 1983-84
Switched to Macintosh in labs

### 1980s
CS began getting computers on their desks.

### 2014
Moved into Gates Hall!
Java is not the Only OO Language

- Usability
- Performance
- Security

Performance

- Java has a reputation for being slow
- Early versions were slow
- Java programs start up slow

Compiling in Java

- Compiler converts source code (*.java) into platform-neutral bytecode (*.class)
- JVM runs bytecode using just-in-time compilation
- JIT performs dynamic code optimization

Garbage Collection

- What happens to objects after you are done with them?
- Why don’t you run out of memory?
- JVM implements garbage collection. It detects and frees objects that are no longer needed

Reachable Objects

- An object is reachable if it is referenced anywhere in the call stack
  - local variables
  - method parameters
  - global variables
- An object is reachable if it is referenced by a reachable object
  - fields
  - array elements

Mark-and-Sweep

- Each object has an extra 1-bit field that is reserved for garbage collecting use
- Garbage Collector (GC) operates in two phases:
  - mark: GC does a tree traversal of reachable objects from the stack and sets the GC field
  - sweep: GC scans all memory from start to finish and frees all objects that do not have the GC field set
# Optimized Garbage Collection

- Concurrent mark-and-sweep
- Generational management
- Garbage-First garbage collector

## Performance

![C++ vs Java Graph]

## Shortcomings of Java

- Java has no separation between specification and implementation
- Writing correct concurrent programs in Java is hard and/or inefficient
- People continue to develop new languages (e.g., Rust) that address some of these shortcomings
  - Steeper learning curve
  - Longer compile times

## tl;dr;

- Modern compiled, OO languages have similar performance
- Different companies use different languages for historical, philosophical, or legal reasons
- The concepts you learned in this class apply to any language
  - Abstraction
  - Isolation
  - Inheritance
  - Incremental development & testing

## Object-Oriented Design

- Problem: how to design a large program
- Design considerations:
  - How easy to make changes? (Flexible)
  - How easy to reuse? (Reusable)
  - How easy to maintain? (Maintain)

## Object-Oriented Design

1. What classes do you need?
2. What is the relationship between those classes?
3. What classes should do what?
4. How should objects interact?
Example: Dice Game

Application domain: Play a dice game. Players requests to roll the dice. System presents results: If the dice face values sum to seven, player wins; otherwise player loses.

Example: Dice Game

1. What classes do you need?
2. What is the relationship between those classes?
3. How should objects interact?
4. What classes should do what?

Design Patterns

- Design patterns are general, re-usable solutions to commonly recurring problems
- OO design patterns typically show relationships and interactions between classes or objects
- Not a magic solution; blindly applying design patterns can overcomplicate your code

Example: MVC

- Model-View-Controller is a common pattern for developing applications with GUIs

CS2110

- Object-oriented programming, reasoning about complex problems
- Testing; Reasoning about correctness
- Algorithmic complexity, analyzing algorithms,
- Data structures: linked lists, trees, hash tables, graphs, etc.
- Programming paradigms: recursion, parallel execution