Data Structures & Algorithms

Adil M. Khan
Professor of Computer Science
Innopolis University

Agenda

- Course Outline
- Motivation
- Data, algorithms, and Software development

Course Outline

- **Motivation & Preview**
  - The importance of algorithms & data structures
- **Algorithm Analysis** – Performance of algorithms
  - Time and space tradeoff
  - Worst case and average case performance
  - Big O, Big Ω (Omega), and Big Θ (Theta) notations

Course Outline

- **Elementary Data Structures**
  - Abstract Data Types (ADTs)
  - List, Stacks and Queues
    - ADTs specification
    - Array Implementation
    - Linked Implementation

Course Outline

- **Hash Tables**
  - Hash functions
  - Compression functions
  - Collision handling
  - Other Set and Map ADTs implementation

Course Outline

- **Algorithmic strategies**
  - Various strategies (Brute-force, Divide-and-conquer, Greedy etc.)
  - **Recursion**
    - Time complexity of recursive algorithms
    - Pitfalls of recursion

Course Outline

- **Searching Algorithms**
  - Linear search
  - Binary search

- **Sorting Algorithms**
  - Insertion sort
  - Selection sort
  - Bubble sort
  - … and faster sort

Course Outline

- **Trees**
  - Binary Trees
  - Binary Search Trees
  - AVL & Red Black Trees
  - B-Trees

Course Outline

- **Priority Queues**
  - Priority queue data structure
  - Binary heap
  - Heapsort

Course Outline

- **Graphs**
  - Graph Representations
  - Graph Traversals
  - Minimum spanning tree
  - Shortest path algorithms
  - Topological sorting
  - Maxflow Algorithms

Motivation

The Importance of Data Structures and Algorithms

Example: Von Neumann Architecture principles

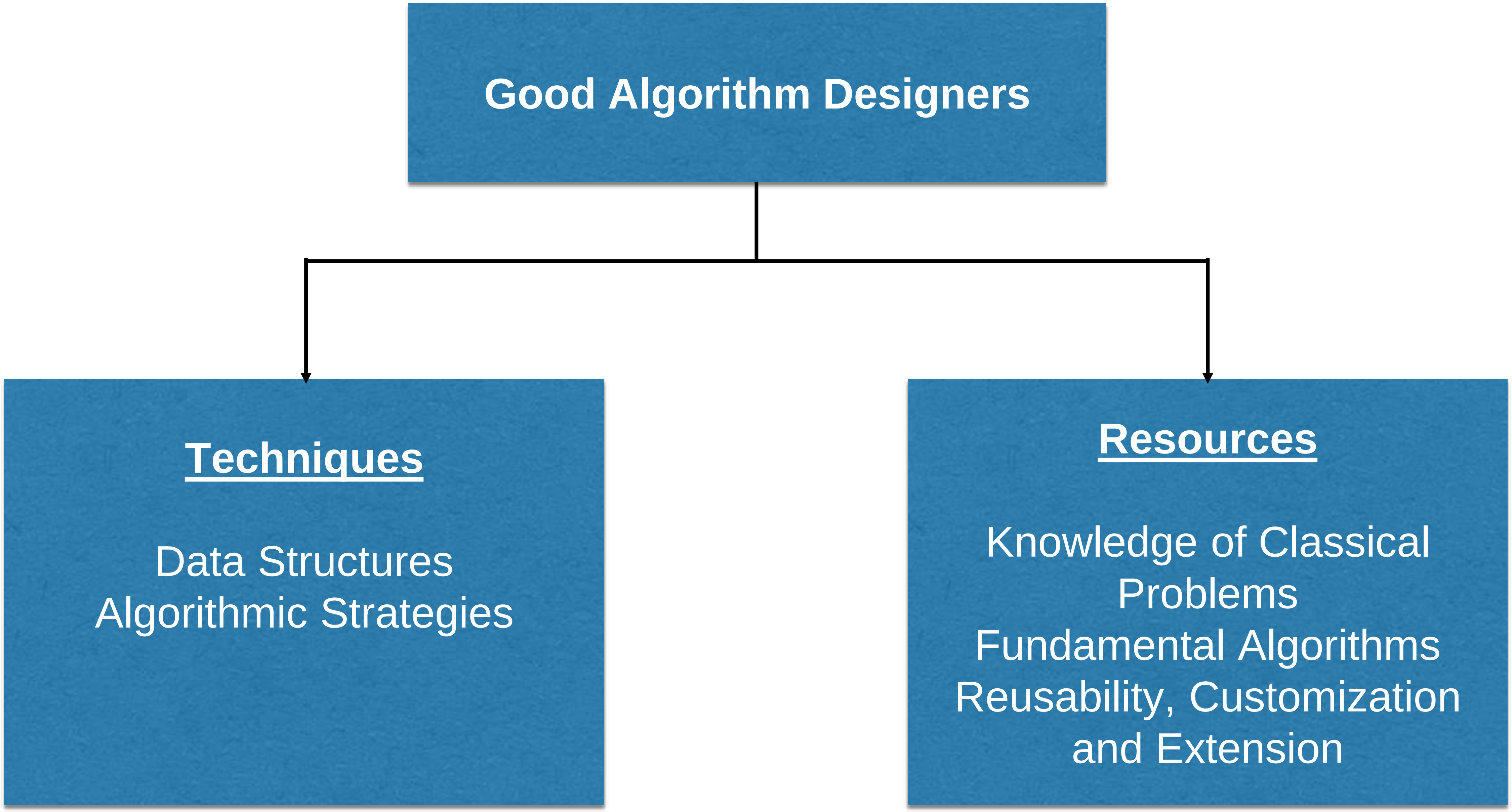
- **Memory** is homogenous. Code and data are stored together.
  - There's no way to distinguish **code** from **data**. (This makes it different from Harvard Architecture)
- Addressing principle: **data** can be referenced by address
- Control flow
  - Sequential* execution of the **code**
- Binary encoding
  - **Data** and **code** and represented as binary words

Data Structures + Algorithms = Programs

- **Algorithms**: a procedure that accomplishes a certain task or solves a "*general and well-specified"* problem – they go hand in hand with **data structures**

  - For systems to be economical the data must be organized (into data structures) in such a way as to support efficient manipulation (by algorithms).

- Choosing the wrong algorithms and data structures makes a program slow at best and unmaintainable and insecure at worst.



Good Algorithm Designers

Techniques

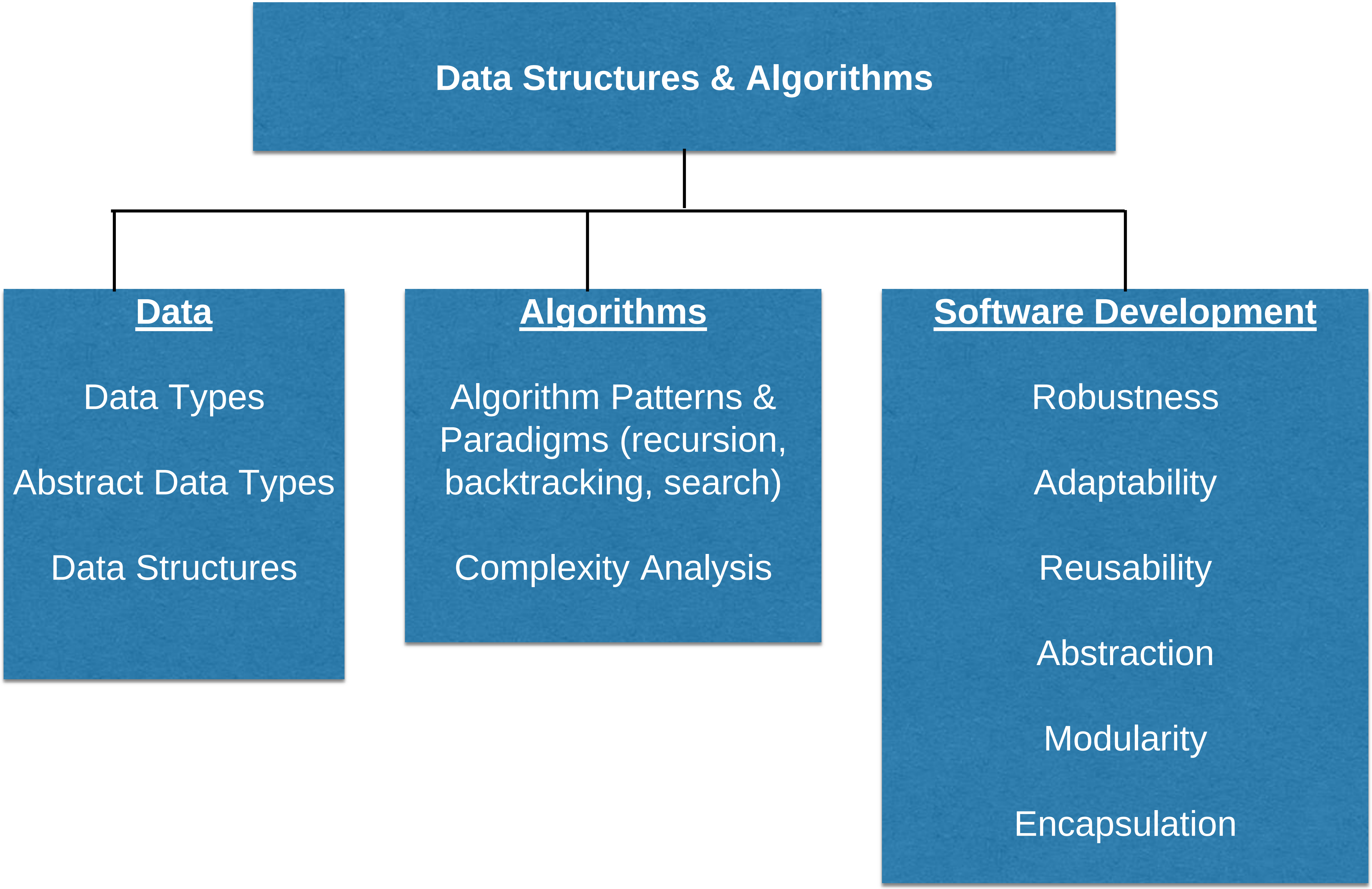
Data Structures
Algorithmic Strategies

Resources

Knowledge of Classical Problems
Fundamental Algorithms
Reusability, Customization and Extension

This course is designed to equip you with both!

Topics



Data Structures & Algorithms

Data

Data Types

Abstract Data Types

Data Structures

Algorithms

Algorithm Patterns & Paradigms (recursion, backtracking, search)

Complexity Analysis

Software Development

Robustness

Adaptability

Reusability

Abstraction

Modularity

Encapsulation

Algorithms

Algorithms

An Algorithm is a finite set of instructions that, if followed, accomplishes a general, well-specified task and must satisfy the following criteria:

① **Input** : There are zero or more quantities that are externally supplied.

② **Output** : At least one quantity is produced.

③ **Definiteness** : Each instruction is clear and unambiguous.

④ **Finiteness** : The algorithm terminates after a finite number of steps/instructions.

⑤ **Effectiveness** : Every instruction must be basic enough to be carried out. It must be definite and also be feasible.

Example: total method

- *Method* in sequential specification is a function:
  - (state Q, **input**[1] e) → (state Q* , **output**[2] r)

- Method has events:
  - invocation (start), **response**[4] (end), result

- *Total* method is a function **defined for any**[3,5] (Q, e)

Algorithms

Three desirable properties of a good algorithms are

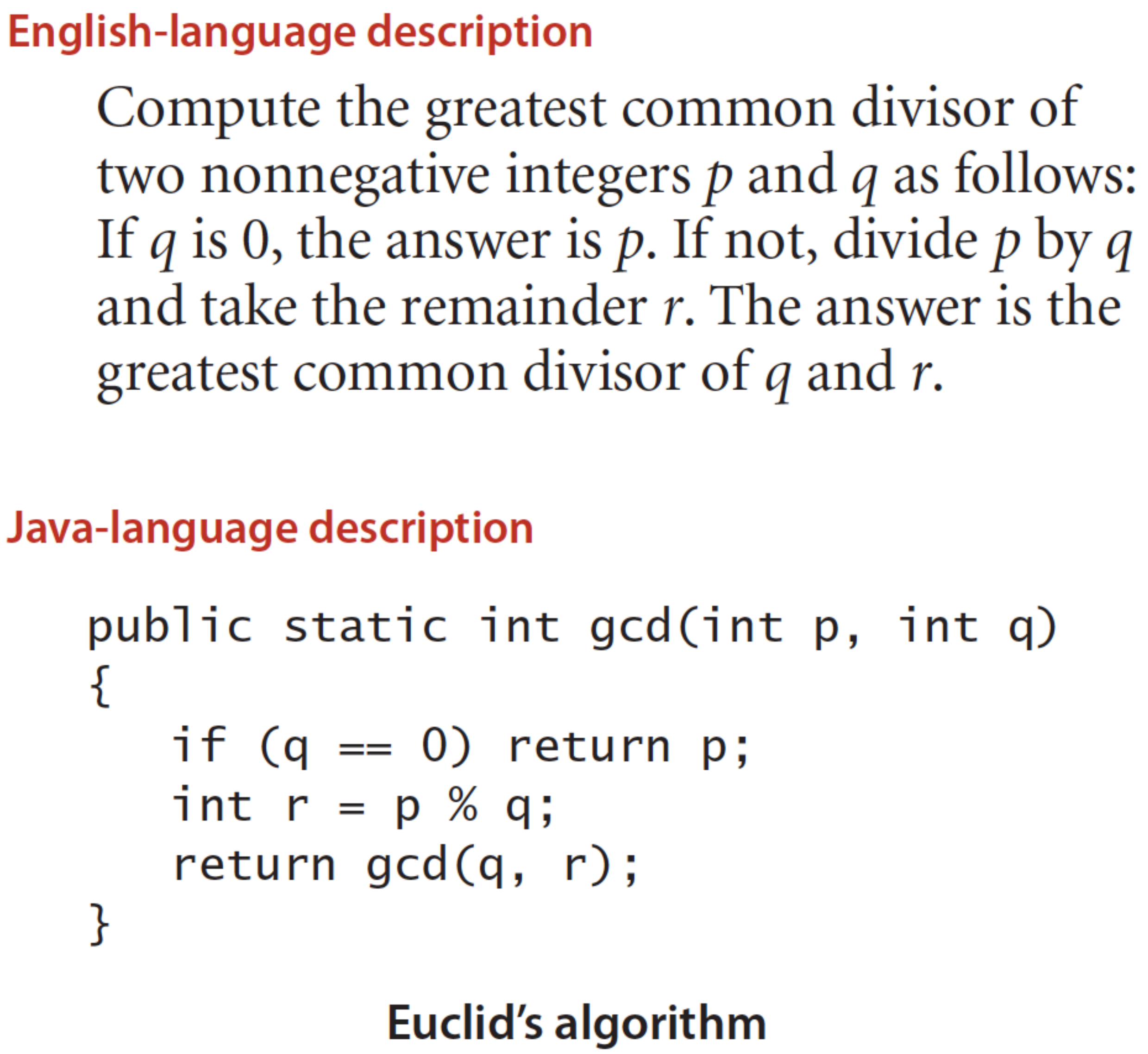
① Correctness

② Efficiency

③ Ease of implementation

Algorithms

Can be described in a natural language or by writing a computer program



English-language description

Compute the greatest common divisor of two nonnegative integers *p* and *q* as follows: If *q* is 0, the answer is *p*. If not, divide *p* by *q* and take the remainder *r*. The answer is the greatest common divisor of *q* and *r*.

Java-language description

```
public static int gcd(int p, int q)
{
   if (q == 0) return p;
   int r = p % q;
   return gcd(q, r);
}
```

Euclid's algorithm

Classifying Algorithms

- By ***Problem Domain:*** numeric, text processing, sorting, searching, networks, machine learning, …

- By ***Design Strategy:*** divide and conquer, greedy, dynamic programming, backtracking, …

- By ***Complexity:*** constant, linear, quadratic, cubic, exponential, …

- By ***Implementation Dimensions:*** sequential, parallel, recursive, iterative, …

Algorithms

(How to Choose The Right One)

- An art: requires cleverness, ingenuity, and sometimes dumb luck

- A Science: Principles of algorithm analysis, and widely applicable algorithm patterns have been developed over time

Data

Type

- Set of values

Z	{. . ., -2, -1, 0, 1, 2, . . .}
N	{0, 1, 2, 3, 4, . . .}
B	{false, true}

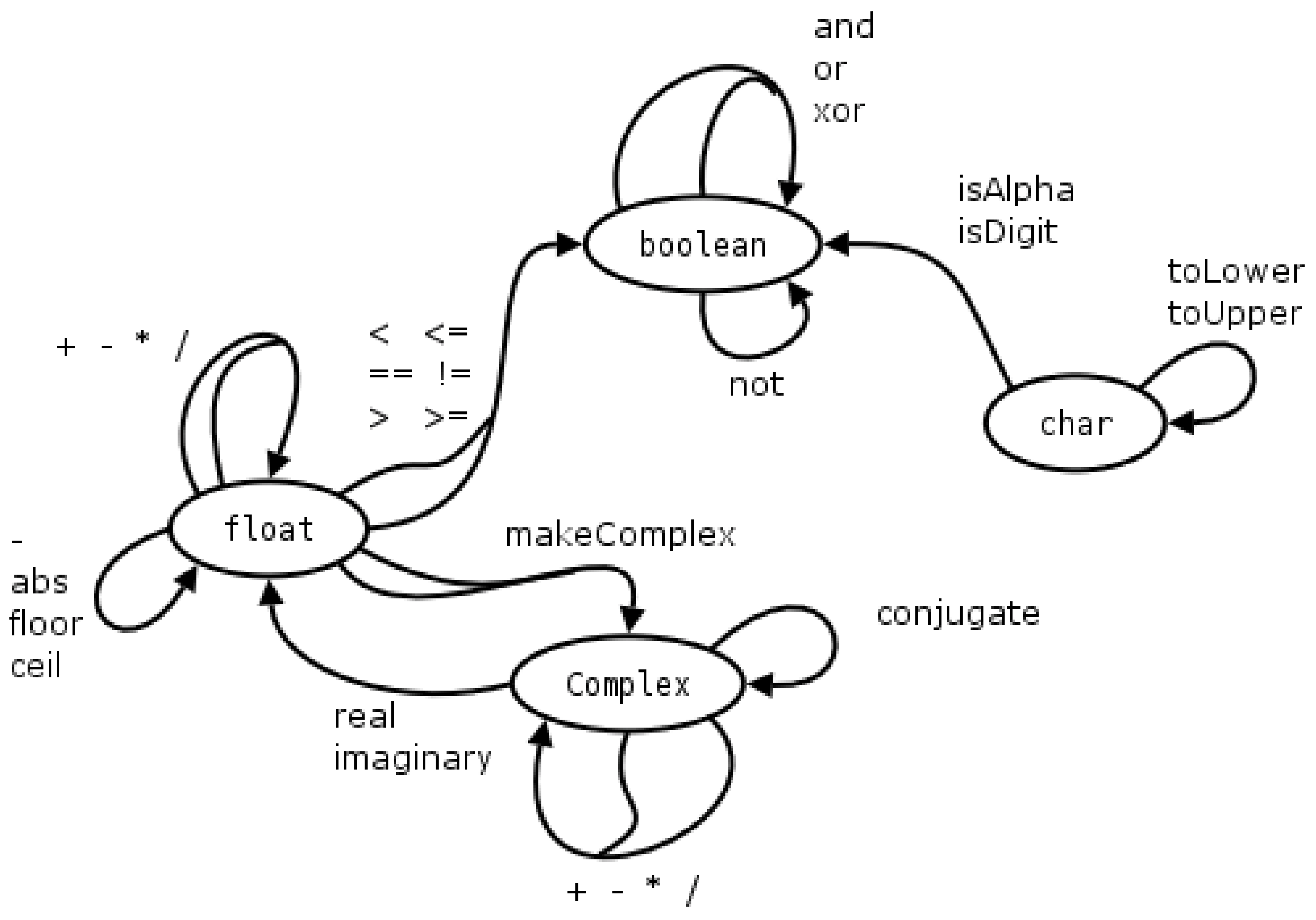
Value types limitation

- Value **range**
  - Overflows
- **Accuracy**
  - Floating point summation
  - Machine zero
  - Floating point comparison
    - Type in your browser:

```
javascript:document.write(Math.sqrt(7) * Math.sqrt(7) == 7)
```

Data Type

Data Type = Type + Operations



- Data Types
  - **Primitive** (value)
    - integer, float, boolean, character
    - string*, pointer*
  - **Complex** (reference)
    - employee, department, list, stack,… .

Abstract Data Type

- A type in which the **internal structure** and the internal working of the objects of the type are **unknown** to users of the type
- Users can only see the effects of the operations
- For example: **Stack ADT**, can be implemented as
  - ***Array Stack*** — a contiguous block of memory
  - ***Linked Stack*** — a non-contiguous memory blocks linked by pointer

Data Structures

- Arrangement of data for the purpose of being able to store and retrieve information
- «Physical» (exact) implementation of an ADT
- Example: **List** (ADT), can be implemented with
  - Array: **Array List** data structure
  - Linked nodes***:*** **Linked List** data structure

Data Structures

(Importance of Choosing The Right One)

- Changing the DS in a slow program can work the same way as an organ transplant does in a sick patient
- Has nothing to do with the correctness of the program
- Remember, it is better to be born with a good heart than have to wait for replacement
- For max benefit, choose the right data structure and design your program around it

Data Structures

(How to Choose The Right One)

- Some important questions to ask:
  - Can the DS be completely **filled at the beginning**, or will there be insertions along with deletions, lookups, updates and other operations?
  - Will the items be **processed in a well-defined** order, or will **random access** have to be supported?

Software Development

Software Development

Goals

Construct software that are:	
Correct	Do exactly what they are intended to do
Reliable	Do not crash
Robust	Can handle unexpected input
Predictable	Do not behave strangely, without any good reason
Reusable	Same code should be useable as a component of different systems in various applications

Think of a few more as an exercise!

Software Development

How to achieve these goals:	
Abstraction	Recognizing fundamental concepts, structures and behaviors, without concern for implementation details
Classification	Recognition that every object is an instance of some class
Hierarchy	Distillation of essential similarities and differences

Abstraction

- Primary way humans deal with complexity
  - View software components in an abstract way, that is, describe what they do without describing how they do it

Some Examples	
Driving a car	no need to know how internal combustion, and fuel cells work
Using a microwave	no need to know the physics to cook
Using a phone	no need to know how the voice is encoded

Abstraction

- Primary types of abstraction in programming
  - ***Procedural Abstraction:*** call a function with a known interface without knowing how it runs
  - ***Data Abstraction:*** declare objects with known types without knowing how those objects are laid out in memory or how the operations that manipulate them work

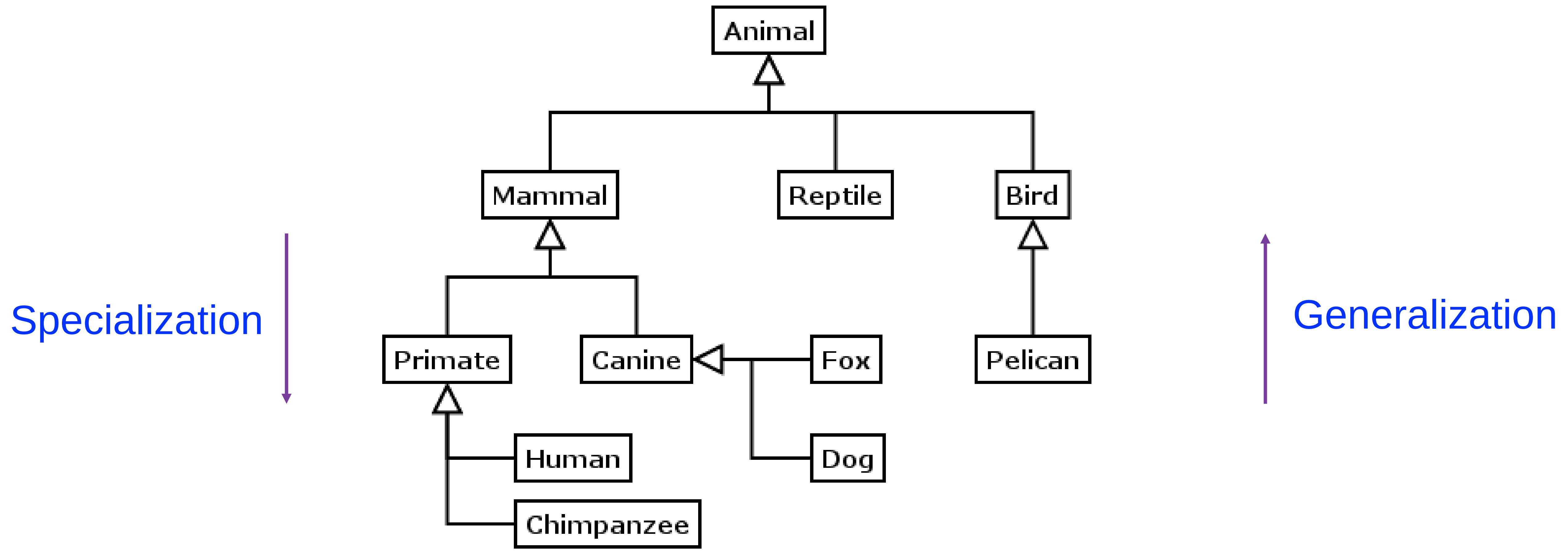
Classification

- Identifying similarities in structure and behavior in a number of objects
- Considering them as objects of the same class
- Giving a name to that class
  - ***Dog*** is a class
  - ***Your particular dog*** is an object of that class

Hierarchy

Is-a Hierarchy

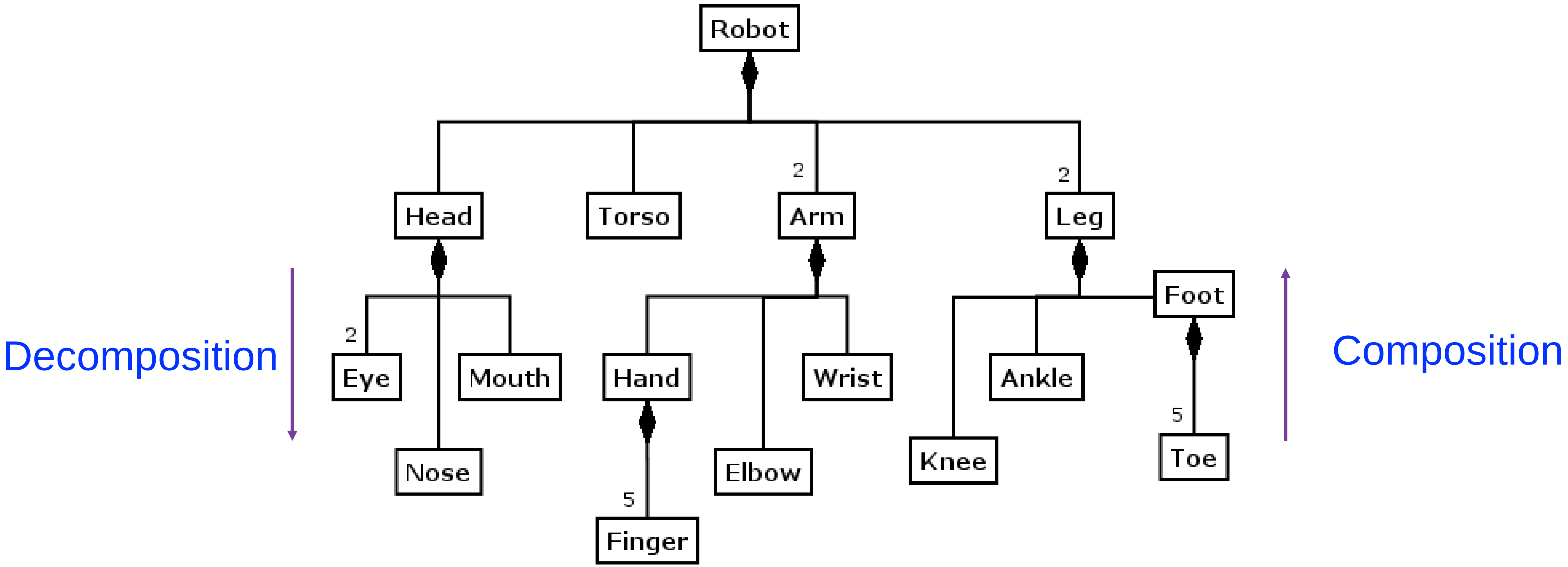
- Organization of classes as super- and sub-classes



Hierarchy

Has-a Hierarchy

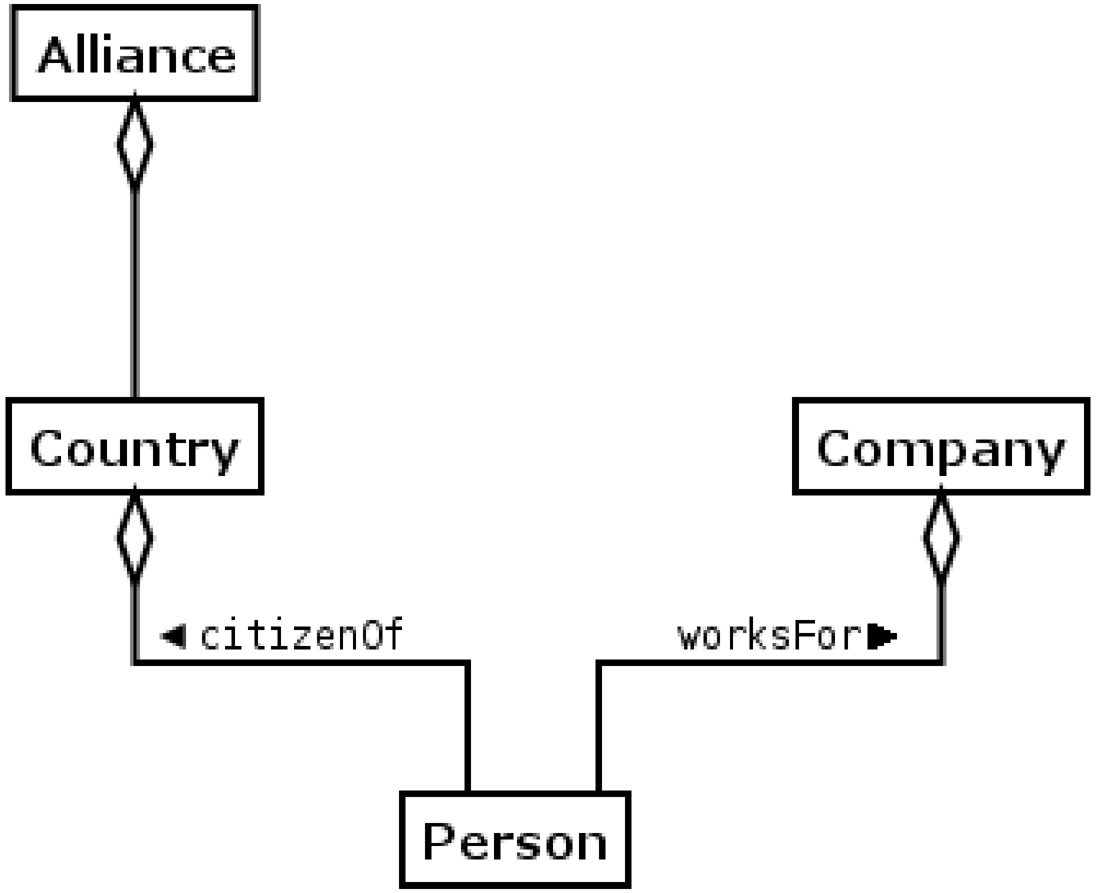
- Classes in containment hierarchy



Hierarchy

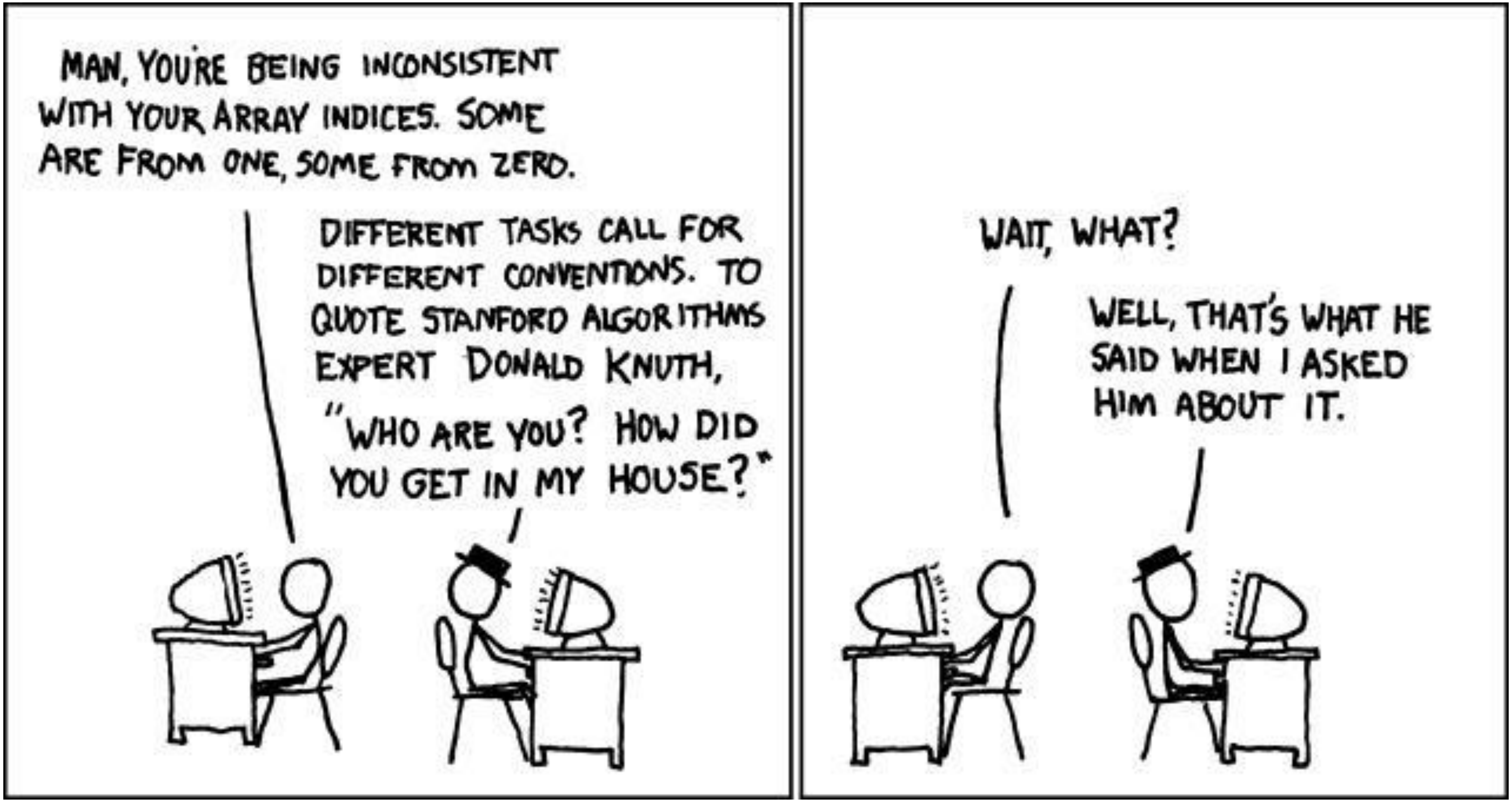
Member-of Hierarchy

- Classes related by groups and subgroups



So, Why Study All of These?

- Because we want to produce efficient software, one which minimizes these
  - Time
  - Space (memory)
  - Coding Time
  - Verification and Debugging Time
  - System Integration Time


MAN, YOU'RE BEING INCONSISTENT WITH YOUR ARRAY INDICES. SOME ARE FROM ONE, SOME FROM ZERO.
DIFFERENT TASKS CALL FOR DIFFERENT CONVENTIONS. TO QUOTE STANFORD ALGORITHMS EXPERT DONALD KNUTH, "WHO ARE YOU? HOW DID YOU GET IN MY HOUSE?"
WAIT, WHAT?
WELL, THAT'S WHAT HE SAID WHEN I ASKED HIM ABOUT IT.