Abstraction facilitates the correct construction of complex systems

Example: voltage levels $\rightarrow$ bits $\rightarrow$ machine language $\rightarrow$ assembly language $\rightarrow$ Java

But abstraction may trade off efficiency for correctness

Arrays: must declare size beforehand (waste memory)

[in Java, initialization wastes run-time as well...]

A data structure: a way of storing and organizing data in a computer

Abstract Data Types: ADT’s – an abstract model for a certain class of data structures that have similar behavior

Java: an interface specifies an ADT; a class that implements the interface is the data structure
Abstraction

- Software developers use ADT’s; we will learn
  1. How to use common ADT’s (wearing our software developer hat)
     - Java: Choosing the right interface
  2. How to implement these ADTs efficiently (our computer scientist hat)
     - Java: write concrete classes for implementing the interfaces
Introduction to the course

Some of the topics we will cover

• “Linear” data structures – stacks and queues
  • Implementation using arrays and linked lists

• A brief introduction to run-time analysis: Big-Oh notation
  • Illustration via sorting (insertion/ bubble/ ... /merge-sort) and searching

• The priority queue ADT
  • Implementation using heaps
  • Heapsort

• Some more sorting: quicksort; radix- and bucket- sort; external sorting

• Dynamic Dictionaries
  • binary search trees (BSTs),
  • balanced BSTs,
  • hash tables

• Graphs – representation; top-sort; shortest paths