Linear data structures: "What to use when"

- Lists (single, double-linked)
- Stacks
- Queues
- Arrays

"Collections" (Java) - Abstract List
List Interface (generic types)
\[ \text{List}\langle T \rangle \]

\text{generic type}

Implementation:
- LinkedList (single)
- Array List
- Vector

Linked List
- add(e) or remove(e)
- adding in n-th position \( O(n) \)

Array List
- Adding at front:
  - shift content \( O(n) \)
  - put the new element \( O(1) \)
- \( O(m) \)
Advantage of ArrayList: random access
Any element (index i) accesses in \(O(1)\)

\[
\begin{array}{c}
\text{start address} + i \times \text{length of element}
\end{array}
\]

Linked List: follow references \(\Rightarrow O(m)\)
Setting ith element to some value: same
Reaching max capacity of ArrayList:
  \(\Rightarrow\) allocate more memory \(\Rightarrow O(m)\)
  \(\Rightarrow\) copy content

ArrayList vs. Vector
Both use array “under the hood”
Vector has some older methods
Single thread

2-4 “cores” common
Graphical processing units (GPUs)
Code \(\Rightarrow\) split it into “Threads”
Threads execute in parallel (1 thread/process)\(\Rightarrow\) fairly independent, but sometimes they need to interact with each other.
Synchronization may be required
- Eg. multiple players accessing same resource
- "Spawn off" computation → returns some result
- Lock some resource for some time

ArrayList → synchronization is not automatic
Vector → synchronized automatically
Synchronization slows things down (a bit)

Thread is a class ⇒ `run()`
Locking—specifically gives context to one thread on shared structures
(also slows things down)

Stacks vs Queues

- **LIFO** (Last In First Out)
  - Execution stack
    - "Undo" features
- First In First Out (FIFO)
  - Multiple jobs that need to be processed "fairly"
What if First in First Out is not enough?

- Priority of each job

FIFO -> highest priority goes first

Priority Queue

<table>
<thead>
<tr>
<th>Content</th>
<th>Priority</th>
</tr>
</thead>
</table>

(int)

You may want it to change with time

- Sorted away a list (sorted in order of priority)
- Inserting a new element is expensive (need to do it in right position)
- If priorities change, queue may need to be rearranged

- Heap: better data structure for priority queues (makes insert faster)
Trees: "non-linear" data structure

- Root (one!)
  - Parent
  - Children
  - Descendants

In general: each node has
- 1 parent
- 0 to ... children
- Descendants: all nodes below
- Ancestors: all nodes above (until the root)

In general trees: can have as many children as wanted
Binary trees: up to 2 children per node
E.g. Game tree
Tic tac toe

Node: position

Next possible moves

E.g. Phylogeny tree

Dinosaurus
Birds
Dodo
Robin
Cardinals

E.g.: Expression tree
\((2 + 3) \times 5\)