Linear Data Structures

Lecturer: Nataliia Stulova  
Teaching assistant: Mohammadreza Hazirprasad
Linear data structures

- Arrays
- Lists
- Stacks
- Queues

- They are abstractions of all kinds of rows, sequences, and series from the real world...
- ...so their elements are arranged sequentially or linearly and linked one after another in a specified order
Arrays
Array data structure

- A native data structure to store a fixed number of elements of the same type
- Elements are accessed by their relative position (*random access*) - each element is independent of others

N-elements array indices range from 0 to N-1
Java arrays

On creation arrays of primitive types are filled with default values:

```java
boolean status[];
status = new boolean[2];
status[0] = true;
```

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>
Creating Java arrays

Arrays of primitive types

```java
int nums[] = new int[2];
nums[0] = 23;
nums[1] = 9;
int nums[] = {23, 9};
```

Arrays of objects

```java
Car parking[] = new Car[20];
parking[0] = new Car();
parking[0].setSpeed(0);
Car truck = new Car();
truck.fuel = 20;
parking[1] = truck;
```
Multi-dimensional arrays

Multidimensional arrays are arrays of arrays with each element of the array holding the reference of other array.

```java
MyType matrix[][] = new MyType[s1]..[sN];
```

Examples: spreadsheets, games (like sudoku), timetables, images

```java
int matrix[][] = new int[2][3];
matrix[0][0] = 4;
matrix[1][2] = 3;
```
java.util.Arrays

Reference Javadoc: Arrays (Java SE 11 & JDK 11)

This class contains various methods for manipulating arrays (such as sorting and searching):

- `fill()`
- `sort()` (last lecture)
- `binarySearch()` (last lecture)
- `copyOf()`
- `equals()`
- ...
Lists
Linked list data structure

- a data structure to store a *non*-fixed number of elements of the same type
- elements are accessed in their order (*sequential access*) - each element needs to be connected to the previous
implementing a linked list data structure from scratch in Java can involve Nested Classes - a way of logically grouping classes that are only used in one place.

```java
public class LinkedList<T> {
    // Node inner class
    public class Node {
        public T data; // Data to store
        public Node nextNode; // Link to next node
    }

    // Head node
    public Node headNode;

    ...
}
```
Java lists: Classes VS Interfaces

- **List<E>** is an *Interface* - a blueprint of a class, does not hold any implementation details
- **LinkedList<E>** is a *Class* - a blueprint of an object, has attributes and methods, does not hold any values
- **myList** is an *Object* - an instance of the LinkedList<E> class, holds concrete values in its attributes

```java
interface List<E> {
    add(...);
    remove(...);
    ...
}

class LinkedList<E> {
    attributes, if any
    add(...) {...}
    remove(...) {...}
    ...
}

List<String> myList = new LinkedList<String>();
myList.add("Potatoes");
```
Accessing list elements

List<String> groceries = Arrays.asList("Potatoes", "Ketchup", "Eggs");

**Loops**

```java
for (int i = 0; i < groceries.size(); i++) {
    System.out.println(groceries.get(i));
}

for (String product : groceries) {
    System.out.println(product);
}
```

**Iterators**

An interface to go through elements in a collection data structure:
- hasNext() method checks if there are any elements remaining in the list
- next() method returns the next element in the iteration

```java
Iterator<String> groceriesIterator = groceries.iterator();
while (groceriesIterator.hasNext()) {
    System.out.println(groceriesIterator.next());
}
```
java.util.List

Reference Javadoc: List (Java SE 11 & JDK 11)

Some classes implementing the List interface:

- LinkedList (Java SE 11 & JDK 11)
- ArrayList (Java SE 11 & JDK 11)
- Vector (Java SE 11 & JDK 11)

Differences: memory management, element access (some allow random access), allowing or not null elements, ...

A library interface that provides various useful operations on lists:

- get()
- add(), addAll()
- remove()
- contains(), containsAll()
- clone()
- equals()
- ...

Reference Javadoc: List (Java SE 11 & JDK 11)
Stacks
Stack data structure

- a data structure to store a non-fixed number of elements of the same type
- elements are stored sequentially, but accessed by the Last In First Out (LIFO) principle, one at a time, at the top of the stack

![Stack element added last](image)

- top of the stack
- bottom of the stack
Stack operations

Basic:

- **push**: add an element to the top of the stack
- **pop**: remove an element from the top of the stack and return it

Extra:

- **top/peek**: get the value of the top element of the stack without removing the element
- checks for emptiness and fullness
Stack implementation and use

Some examples of use

- an “undo” mechanism in text editors
- forward and backward navigation in web browsers
- expression parsing and evaluation (e.g.,)
- memory management (part II of this course)

Implementations

- array-based, esp. with fixed capacity
- as a resizable array (e.g., using a `Vector`)
- linked list-based
java.util.Stack<E>


The Stack class represents a last-in-first-out (LIFO) stack of objects.

- empty()
- peek()
- pop()
- push(E item)
- search(Object obj)
Queues
Queue data structure

- a data structure to store a non-fixed number of elements of the same type
- elements are stored sequentially, but accessed by the First In First Out (FIFO) principle, one at a time, at the top of the stack
Queue operations

Basic:
- **enqueue**: add an element to the back of the queue
- **dequeue**: remove an element from the front of the queue and return it

Extra:
- **front**: get the value of the first element of the queue without removing the element
- checks for emptiness and fullness
Queue implementation and use

Some examples of use

- handling of high-priority processes in an operating system is handled using queues
- ordering requests to a printer to print pages, the requests are handled by using a queue
- messages on social media, they are sent to a queue on the server

Implementations

- array-based, esp. with fixed capacity
- linked list-based
java.util.Queue<E>

Reference Javadoc: Queue (Java SE 11 & JDK 11)

A library interface that provides various queue operations:

### Summary of Queue methods

<table>
<thead>
<tr>
<th></th>
<th>Throws exception</th>
<th>Returns special value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>add(e)</td>
<td>offer(e)</td>
</tr>
<tr>
<td>Remove</td>
<td>remove()</td>
<td>poll()</td>
</tr>
<tr>
<td>Examine</td>
<td>element()</td>
<td>peek()</td>
</tr>
</tbody>
</table>
What you should remember

Use arrays when:

- you know the number of elements...
- ... or the number of elements will increase rarely
- you need fast access to individual elements

Use lists when:

- you do not know the number of elements
- you do not need fast access to individual elements
Summary and practice
Exercise 1: Arrays

Matrix multiplication

- write a class representing a 2D matrix
- attributes:
  - int matrix[][]
- methods:
  - Matrix(int rows, int cols) - constructor
  - Matrix add(Matrix other) - addition
  - Matrix product(Matrix other) - multiplication

I/O

Tests (JUnit, class MatrixTest)

- dimensions mismatch
- 3 correct cases: 1-column matrix, 1-row matrix, a 2x3 matrix

https://en.wikipedia.org/wiki/Matrix_(mathematics)#Basic_operations
Exercise 2: Lists

Computing various average values

- Write a class `Averages` to compute various means: arithmetic, geometric, and harmonic
  [https://en.wikipedia.org/wiki/Average](https://en.wikipedia.org/wiki/Average)
- Methods:
  - `static Double arithMean(ArrayList<E> nums)`
  - `static Double geomMean(ArrayList<E> nums)`
  - `static Double harmMean(ArrayList<E> nums)`

I/O

- Read a sequence of numbers from `System.in`
- Print average values to `System.out`

Tests

- One test for each method

```java
NEW static keyword: helper methods (and no objects!)
Double arMean = Averages.arithMean(ArrayList<E> nums)

boxed types: Integer, Float, Double....
```