TECH ELEVATOR

Module 1, Week 3

Interview Questions

- What is an ArrayList in Java, and how does it differ from a regular array?
- Can you store different types of data in an ArrayList, or does it only support a specific data type?
- How would you iterate through the elements of an ArrayList?
- How would you iterate through the elements of a HashMap?
- What happens if you try to add a duplicate key to a HashMap? How can you handle this situation?
- How do you remove key-value pairs from a HashMap?

Overview

- Data Structures / Collections
- Mid-Module Project

Week	Topics
Week 0	Welcome, Intro to Tools
Week 1	Variables and Data Types, Logical Branching
Week 2	Loops and Arrays, Command-Line Programs, Intro to Objects
Week 3	Collections
Week 4	Mid-Module Project (Code Review)
Week 5	Classes and Encapsulation
Week 6	Inheritance, Polymorphism
Week 7	Unit Testing, Exceptions and Error Handling
Week 8	File I/O Reading and Writing
Week 9	End-of-module project
Week 10	Assessment

Data Structures Overview

- Arrays
- Lists
- Maps
- **Sets**
- **Stacks**

Arrays vs. Lists

Array

- Fixed size
- Ordered
- Object
- Access values by index
- Can store primitives or objects
- Faster performance

List

- Can grow/shrink
- Ordered
- Object
- Access values by index
- Can’t store primitives, only objects
- Slower performance

List Operations

Operation	Description
add(T element)	Appends an element to end of list
add(int index, T element)	Inserts an element to list at specific index
get(int index)	Retrieve element at index
size()	Get number of elements
set(int index, element)	Replaces element at specified index
contains(T element)	Returns **true** if element exists in List
remove(T element)	Removes element from List
remove(int index)	Removes element from specified index;

Lists vs. Maps

List

- Ordered
- Values are all the same type
- Can have duplicates
- Access values by index

Map

- Unordered
- Key and value don’t need to be the same data type (but keys are all same type; values are all same type)
- Can’t have duplicate keys
- Access values by key

Map Operations

Operation	Description
put(K key, V value)	Adds or updates key-value entry for specified key.
get(K key)	Retrieves value assigned to key. Returns null if key doesn’t exist.
getOrDefault(K key, V defaultValue)	Retrieves value assigned to key. Returns defaultValue if key doesn’t exist.
size()	Get number of keys in map
containsKey(K key)	Returns **true** if key exists in Map
remove(K key)	Removes key-value entry for specified key.

Operations

	array	**List**	**Map**
Get an element	access by index	get	get
Insert an element	(need new array)	add	put
Remove an element	(need new array)	remove	remove
Check if an element exists	N/A (for-loop)	contains	containsKey

Sets

- Unordered
- Keys are all same type
- Can’t have duplicate keys

Set Operations

Operation	Description
add(T element)	Appends an element if it doesn't exist. If element exists, nothing happens.
size()	Get number of elements
contains(T element)	Returns **true** if element exists in List
remove(T element)	Removes element from List

Stacks vs. Queues

Stack

- Last In, First Out (LIFO)
- Insert/remove an element from one end (push, pop)
- No insertion/deletion in the middle
- (Stack of Dishes)

Queue

- First In, First Out (FIFO)
- Insert an element from one end; remove from the other end (push, pop)
- No insertion/deletion in the middle
- (People waiting in line)

Stack Operations

Operation	Description
push(T element)	Adds an element to “top” of list
pop()	Returns and removes element from “top” of list.
size()	Get number of elements
peek()	Returns element from “front” of list (no removal)

Queue Operations

Operation	Description
enqueue(T element)	Adds an element to “back” of list
dequeue()	Returns and removes element from “front” of list.
size()	Get number of elements
peek()	Returns element from “front” of list (no removal)

Data Structures

What data structure would you use for each of the use cases?

Think about:

- Does the **order** of the values matter?
- Do **duplicates** matter?
- Does how you **look up** values matter?
- Does how you **insert** values matter?

Data Structures

1. state names and populations
2. expense amounts to total up
3. podcasts in the “Play Next” section of a podcast app
4. capturing a user’s actions so an app can provide an “undo” button
5. text messages
6. the answers for a multiple-choice question that always has four options
7. customers in line to order tickets online that go on sale at a certain time
8. a person’s friends (on a social media site, for example)
9. student ids and grades

** For each collection type (array, List, Map, Set, Stack, Queue) think of another use case **

1. state names and populations	
2. expense amounts to total up	
3. podcasts in the “Play Next” section of a podcast app	
4. capturing a user’s actions so an app can provide an “undo” button	
5. text messages	
6. the answers for a multiple-choice question that always has four options	
7. customers in line to order tickets online that go on sale at a certain time	
8. a person’s friends (on a social media site, for example)	
9. student ids and grades	

Mid-Module Project

The design of the application is to allow users to search through a body of data containing books: title, author, year published, and price.

- Declaring and assigning values to variables, including constants
- Writing conditional code using boolean expressions
- Using arrays, collections, and loops
- Parsing strings into numeric data types
- Using methods of the String class for text processing and manipulation

Push as you go for visibility