SELF PRACTICE

SmartCliff
Career Mobility Solutions

SDE Readiness Training

Practice No. **: 9**

Topic **: Collections in Java**

Date **: 18-05-2024**

Solve the following problems

Q. No.	**Question Detail**	**Level**
1	You are tasked with creating a program to manage a library's book inventory using ArrayLists. Implement a Java class called **LibraryInventory** with the following functionalities: **i. Adding Books:** a. Adds a new book title to the library inventory. **ii. Removing Books:** a. Removes a specific book title from the inventory. Returns true if the book was successfully removed, false otherwise. b. Removes books from the inventory based on a specified condition. **iii. Searching and Checking:** a. Checks if a book with the given title exists in the inventory. Returns true if found, otherwise false. b. Checks if the library inventory is empty. Returns true if empty, otherwise false. **iv. Listing Books:** a. Lists all the books in the inventory, typically alphabetically. **v. Sorting and Ordering:** a. Sorts the books in the inventory alphabetically by title. b. Sorts the books in the inventory alphabetically by author. **vi. Size and Capacity:** a. Returns the number of books currently in the inventory.	Medium

Sometimes later becomes never. DO IT NOW!

	b. Increases the capacity of the inventory by the specified amount. **vii. Iteration and Conversion:** a. Iterates over the inventory and prints each book's title and author. b. Converts the inventory ArrayList to a regular array of book titles. c. Returns a special iterator capable of iterating over the inventory and performing remove operations on the books. **viii. Additional Functionality:** a. Keeping track of the number of copies available for each book. b. Methods for lending and returning books, which involve decrementing and incrementing the available copies respectively.	
2	You are tasked with implementing a Java class called LinkedListPractice to manage a list of students using a linked list. Include the following functionalities along with their respective methods: **i. Adding Students:** a. Implement a method to add a new student to the list. **ii. Removing Students:** a. Implement a method to remove a specific student from the list by their name. b. Implement a method to remove all students with a specified age. **iii. Searching and Checking:** a. Implement a method to check if a student with a given name exists in the list. b. Implement a method to check if the list is empty. **iv. Listing Students:** a. Implement a method to print the names of all students in the list. **v. Size and Capacity:**	

	a. Implement a method to get the total number of students in the list. b. Implement a method to increase the capacity of the list by a specified amount. **vi. Iteration and Conversion:** a. Implement a method to iterate over the list and print each student's name and age. b. Implement a method to convert the linked list to an array of student names. c. Implement a method to return a special iterator that iterates over the list and performs remove operations on the students. d. Implement a method to return a descending iterator that iterates over the list in reverse order. **vii. Sorting and Ordering:** a. Implement a method to sort the students in the list alphabetically by their names. b. Implement a method to sort the students in the list by their ages in ascending order. **viii. Additional Functionality:** a. Include functionality to keep track of each student's age and grade. b. Implement methods to update a student's age or grade. a. Implement a method to clear the entire list of students.	
3	You are tasked with implementing a Java class called **VectorPractice** to manage a list of products using a Vector. Include the following functionalities along with their respective methods: i. **Adding Products**: a. Implement a method to add a new product to the vector. ii. **Removing Products**: a. Implement a method to remove a specific product from the vector by its name.	

b. Implement a method to remove all products with a specified category.

iii. **Searching and Checking**:
   a. Implement a method to check if a product with a given name exists in the vector.
   b. Implement a method to check if the vector is empty.

iv. **Listing Products**:
   a. Implement a method to print the details of all products in the vector.

v. **Size and Capacity**:
   a. Implement a method to get the total number of products in the vector.
   b. Implement a method to increase the capacity of the vector by a specified amount.
   c. Implement a method to trim the capacity of the vector to its current size, removing any unused capacity beyond the actual number of elements stored.

vi. **Iteration and Conversion**:
   a. Implement a method to iterate over the vector and print each product's details.
   b. Implement a method to convert the vector to an array of product objects.

vii. **Sorting and Ordering**:
   a. Implement a method to sort the products in the vector alphabetically by their names.
   b. Implement a method to sort the products in the vector by their prices in ascending order.

viii. **Additional Functionality**:
   a. Include functionality to keep track of each product's category and price.
   b. Implement methods to update a product's category or price.
   c. Implement a method to clear the entire vector of products.

4	You are tasked with managing a stack of books using a Java class called **StackPractice** and a Stack. Given the following initial books: 1. Title: "The Great Gatsby", Author: "F. Scott Fitzgerald", Publication Year: 1925 2. Title: "To Kill a Mockingbird", Author: "Harper Lee", Publication Year: 1960 3. Title: "1984", Author: "George Orwell", Publication Year: 1949 **i. Pushing Books:** a. Implement a method to push a new book onto the stack. **ii. Popping Books:** a. Implement a method to pop the top book from the stack. b. Implement a method to remove and return the top book from the stack using the poll() method. **iii. Peeking:** a. Implement a method to peek at the top book of the stack without removing it. **iv. Searching and Checking:** a. Implement a method to check if a book To Kill a Mockingbird exists in the stack. b. Implement a method to check if the stack is empty. **v. Listing Books:** a. Implement a method to print the titles of all books in the stack. **vi. Size and Capacity:** a. Implement a method to get the total number of books in the stack. b. Implement a method to increase the capacity of the stack by a specified amount. **vii. Iteration and Conversion:** a. Implement a method to iterate over the stack and print each book's title. **viii. Additional Functionality:**	

	a. Include functionality to keep track of each book's author and publication year. b. Implement methods to update a book's author or publication year. c. Implement a method to clear the entire stack of books.	
5	You are managing a priority queue of characters representing tasks to be executed. Below is the initial set of tasks : 1. Task: 'A', 2. Task: 'B', 3. Task: 'C', 4. Task: 'D' i. **Adding Elements:** a. Add elements to the priority queue. ii. **Removing Elements:** a. Remove and retrieve the head of the priority queue. iii. **Accessing Elements:** a. Retrieve the head of the priority queue without removing it. iv. **Checking Queue Status:** a. Check whether the priority queue is empty. b. Return the number of elements in the priority queue. v. **Custom Comparator:** a. Implement a custom comparator to order characters based on their ASCII values, ensuring the element with the maximum ASCII value has the highest priority. vi. **Clearing the Queue:** Remove all elements from the priority queue	
6	Create a new Java class named ArrayDequePractice. Import the necessary Java Collection classes. Initialize an ArrayDeque object named "characterDeque" to store characters and perform the following operations: i. **Adding Elements:**	

a. Add the characters 'A', 'B', 'C', 'D', 'E', and 'F' to the characterDeque.

ii. **Adding Elements at Both Ends:**

a. Add the character 'X' to the beginning of the characterDeque.

b. Add the character 'Y' to the end of the characterDeque.

iii. **Removing Elements:**

a. Remove and retrieve the first element from the characterDeque.

b. Remove and retrieve the last element from the characterDeque.

iv. **Accessing Elements:**

a. Retrieve, without removing, the first element of the characterDeque.

b. Retrieve, without removing, the last element of the characterDeque.

c. Retrieve a character from the characterDeque at a random index and print it.

v. **Checking Deque Status:**

a. Check whether the characterDeque is empty.

b. Determine and print the size of the characterDeque.

vi. **Dynamic Resizing:**

a. Add the characters 'G', 'H', 'I', 'J', 'K', 'L', and 'M' to the characterDeque, observing how it dynamically resizes to accommodate the additional elements.

b. Remove several elements from the characterDeque, ensuring it dynamically shrinks when elements are removed.

vii. **Iteration and Conversion:**

a. Iterate through the elements of the characterDeque and print each character.

b. Use a descending iterator to iterate through the elements of the characterDeque and print each character in reverse order.

	c. Convert the characterDeque into an array and print the resulting array. **viii. Clearing the Deque:** a. Clear all elements from the characterDeque. b. Verify whether the characterDeque is empty after clearing.	
7	Create a new Java class named **HashSetPractice**. Import the necessary Java Collection classes. Initialize a HashSet object named "stringSet" to store strings and perform the following operations: **i. Adding Elements:** a. Add the following strings to the stringSet: "apple", "banana", "orange", "grape". b. Add all elements from a collection named "additionalSet" to the stringSet. **ii. Removing Elements:** a. Remove the string "banana" from the stringSet. b. Remove all elements from the stringSet. c. Remove all elements present in a collection named "removalSet" from the stringSet. **iii. Checking Set Status:** a. Check whether the stringSet contains the string "orange". b. Determine and print the size of the stringSet. c. Check if the stringSet is empty. **iv. Iteration and Conversion:** a. Iterate through the elements of the stringSet and print each string. b. Convert the stringSet into an array and print the resulting array. c. Print the string representation of the stringSet. **v. Retaining Elements:** a. Retain only the elements in the stringSet that are contained in a collection named "retainSet".	
8	Create a new Java class named LinkedHashSetPractice. Import the necessary Java Collection classes. Initialize a LinkedHashSet	

	named "wordSet" to store strings. Add the elements {"dog", "cat", "bird", "fish", "rabbit", "turtle"} to the wordSet and perform the following operations: i. **Adding Elements:** a. Add the string "horse" to the wordSet. ii. **Removing Elements:** a. Remove the string "bird" from the wordSet. iii. **Checking if Set Contains Elements:** a. Check if the wordSet contains the string "fish". iv. **Checking Set Status:** a. Check if the wordSet is empty. b. Determine the size of the wordSet. v. **Iterating Over Set:** a. Iterate through the elements of the wordSet using an iterator obtained and print each element. vi. **Converting Set to Array:** a. Convert the wordSet into an array and print the resulting array. vii. **Hash Code of Set:** a. Print the hash code of the wordSet. viii. **Clearing the Set:** a. Clear all elements from the wordSet.	
9	Create a new Java class named TreeSetStringPractice. Import the necessary Java Collection classes. Define a custom Comparator for strings to reverse the order. Initialize a TreeSet named "stringSet" to store strings, sorted according to the custom Comparator. Add the strings {"apple", "banana", "cherry", "date", "kiwi", "orange"} to the stringSet and perform the following operations: i. **Adding Elements:** a. Add the string "grape" to the stringSet. ii. **Removing Elements:** a. Remove the string "date" from the stringSet. iii. **Checking if Set Contains Elements:** a. Check if the stringSet contains the string "banana".	

iv. **Checking Set Status:**
   a. Check if the stringSet is empty.
   b. Determine the size of the stringSet.

v. **Iterating Over Set:**
   a. Iterate through the elements of the stringSet and print each element.

vi. **Retrieving First and Last Elements:**
   a. Retrieve and print the first (lowest) element and last (highest) element.

vii. **Polling First and Last Elements:**
   a. Retrieve and remove the first (lowest) element and last (highest) element.

10 Create a new Java class named TreeMapPractice. Import the necessary Java Collection classes. Initialize a TreeMap named "studentMap" to store student names (String) as keys and their corresponding ages (Integer) as values. Add the following entries to the studentMap and perform the following operations:

i. **Adding Entries:**
   a. Add the following entries to the studentMap:
      - "Alice" : 20
      - "Bob" : 22
      - "Charlie" : 18
      - "David" : 25
      - "Eva" : 21

ii. **Removing Entry:**
   a. Remove the entry for "Charlie" from the studentMap.

iii. **Checking if Map Contains Key:**
   a. Check if the studentMap contains the key "Bob".

iv. **Checking Map Status:**
   a. Check if the studentMap is empty.
   b. Determine the size of the studentMap.

v. **Iterating Over Map Entries:**
   a. Iterate through the entries of the studentMap and print each key-value pair.

vi. **Retrieving Entry with Maximum Key:**

	a. Retrieve and print the entry with the maximum key (lexicographically last). **vii. Retrieving Entry with Minimum Key:** a. Retrieve and print the entry with the minimum key (lexicographically first). **viii. Polling First and Last Entries:** a. Retrieve and remove the first entry (lexicographically first) from the studentMap. b. Retrieve and remove the last entry (lexicographically last) from the studentMap.	
11	Create a new Java class named **HashMapPractice**. Import the necessary Java Collection classes. Initialize a HashMap named "wordCountMap" to store words (String) as keys and their corresponding counts (Integer) as values. Add the following entries to the wordCountMap and perform the following operations: **i. Adding Key-Value Pairs:** a. Add the following key-value pairs to the wordCountMap: - "apple" : 5 - "banana" : 8 - "cherry" : 3 - "date" : 6 - "grape" : 4 ii. **Copying Mappings:** a. Create a new HashMap named "copyMap". ii. Copy all mappings from the wordCountMap to the copyMap. iii. **Retrieving Values:** a. Retrieve and print the count associated with the word "date". iv. **Removing a Mapping:** a. Remove the mapping for the word "cherry" from the wordCountMap. v. **Checking for Key Presence:** a. Check if the wordCountMap contains the word "banana". ii. Check if the wordCountMap contains the count 4. vi. **Checking HashMap Status:**	

	a. Check if the wordCountMap is empty. ii. Determine and print the number of key-value mappings in the wordCountMap. vii. **Iterating Over Entries:** a. Iterate through the entries of the wordCountMap and print each word-count pair. viii. **Retrieving Keys and Values:** a. Retrieve and print the set of words in the wordCountMap. ii. Retrieve and print the collection of counts in the wordCountMap.	
12	Create a new Java class named LinkedHashMapPractice. Import the necessary Java Collection classes. Initialize a LinkedHashMap named "vehicleTypeMap" to store vehicle types (String) as keys and their corresponding categories (String) as values. Use the following key-value pairs: {"car": "sedan", "truck": "pickup", "motorcycle": "sportbike", "van": "minivan", "suv": "crossover"} Perform the following operations: **i. Adding Key-Value Pairs:** a. Add the given key-value pairs to the vehicleTypeMap. **ii. Copying Mappings:** a. Create a new LinkedHashMap named "copyMap". b. Copy all mappings from the vehicleTypeMap to the copyMap. **iii. Retrieving Values:** a. Retrieve and print the category associated with the vehicle type "motorcycle". **iv. Removing a Mapping:** a. Remove the mapping for the vehicle type "van" from the vehicleTypeMap. **v. Checking for Key Presence:** a. Check if the vehicleTypeMap contains the vehicle type "suv". b. Check if the vehicleTypeMap contains the category "pickup".	

vi. **Checking LinkedHashMap Status:**

  a. Check if the vehicleTypeMap is empty.

  b. Determine and print the number of key-value mappings in the vehicleTypeMap.

vii. **Iterating Over Entries:**

  a. Iterate through the entries of the vehicleTypeMap and print each vehicle type-category pair.

viii. **Retrieving Keys and Values:**

  a. Retrieve and print the set of vehicle types in the vehicleTypeMap.

  b. Retrieve and print the collection of categories in the vehicleTypeMap