Hands-on No. : 9

Topic : Java Collections – List, Queue

Date : 18-05-2024

Solve the following problems

Question No.	Question Detail	Level
1	**Arraylist** Create a new Java class called ListPractice. Import the necessary Java Collection classes and create an ArrayList of Strings named "stringList" and do the following operations: **a. Adding Elements:** i. Add the following strings to the list: "apple", "banana", "orange", "grape". **b. Removing Elements:** i. Remove the element at index 2 from the list. ii. Remove the first occurrence of "banana" from the list. **c. Accessing Elements:** i. Print the element at index 1. ii. Replace the element at index 0 with "pear". **d. Searching and Checking:** i. Check if the list contains "orange" and print the result. ii. Find and print the index of the last occurrence of "grape". iii. Check if the list is empty and print the result. **e. List Operations:** i. Create a new ArrayList of Strings named "newList".	Easy

	ii. Add the strings "kiwi", "pineapple", "melon" to the newList. iii. Add all elements from newList to stringList starting from index 2. **f. Size and Capacity:** i. Print the size of the stringList. ii. Clear the stringList and print its size again. **g. Iteration and Conversion:** i. Use an iterator to iterate over the elements in the stringList and print each element. ii. Create a sublist of stringList from index 1 to 3 and print it. iii. Convert the stringList to an array and print the array. **h. Sorting and Ordering:** i. Sort the elements in stringList in natural order and Print the sorted list. ii. Check if stringList equals newList and print the result. iii. Print the hash code of stringList. iv. Implement a custom comparator to sort stringList in reverse alphabetical order and Print	
2	**Linked List:** Creating a new Java class named LinkedListPractice. Ensure you import the necessary Java Collection classes such as LinkedList and ListIterator. Initialize a LinkedList called "integerList" to store integers and do the following operations: **a. Adding Elements:**	Easy

	i. Populate the integerList with the integers 10, 20, 30, and 40. **b. Removing Elements:** i. Remove the integer at the second position from the integerList. ii. Eliminate the first occurrence of the integer 20 from the integerList. **c. Accessing Elements:** i. Print out the integer stored at the second position in the integerList. ii. Replace the integer at the first position in the integerList with the value 50. **d. Searching and Checking:** i. Determine if the integer 30 exists in the integerList and print the result. ii. Identify and print the index of the last occurrence of the integer 40 in the integerList. iii. Check whether the integerList is empty and print the result. **e. List Iteration and Conversion:** i. Iterate through the elements of the integerList using a ListIterator and print each element. ii. Create a sublist of the integerList containing elements from the second to the fourth position and print it. iii. Convert the integerList into an array and print the resulting array. **f. Size and Capacity:** i. Determine and print the size of the integerList. ii. Clear all elements from the integerList and print its size again.	
3	**Vector:**	Easy

Create a new Java class named VectorPractice. Import the necessary Java Collection classes. Initialize a Vector named "flowerVector" to store flower objects and do the following operations:

a. Adding Elements:

i. Add the following flowers to the flowerVector: Rose, Lily, Tulip, Daisy.

b. Removing Elements:

i. Remove the flower at the second position from the flowerVector.

ii. Remove the first occurrence of the flower "Lily" from the flowerVector.

c. Accessing Elements:

i. Print out the flower stored at the second position in the flowerVector.

ii. Replace the flower at the first position in the flowerVector with "Sunflower".

d. Searching and Checking:

i. Check if the flower "Tulip" exists in the flowerVector and print the result.

ii. Identify and print the index of the last occurrence of the flower "Daisy" in the flowerVector.

iii. Check whether the flowerVector is empty and print the result.

e. Iteration and Conversion:

i. Iterate through the elements of the flowerVector using a for-each loop and print each flower.

ii. Create a sublist of the flowerVector containing elements from the second to the fourth position and print it.

iii. Convert the flowerVector into an array and print the resulting array.

f. Size and Capacity:

	i. Determine and print the size of the flowerVector. ii. Print the current capacity of the flowerVector. **g. Dynamic Array Operations:** i. Add two more flowers ("Orchid" and "Carnation") to the flowerVector. ii. Check and print the new capacity of the flowerVector after adding the flowers. iii. Remove the flower at index 3 from the flowerVector. iv. Print the size of the flowerVector after removal.	
4	**Stack:** Create a new Java class named StackPractice. Import the necessary Java Collection classes .Initialize a Stack named "integerStack" to store integer values and do the following : **a. Adding Elements:** i. Push the following integers onto the integerStack: 10, 20, 30, 40. **b. Removing Elements:** i. Pop the top element from the integerStack. **c. Accessing Elements:** i. Peek at the top element of the integerStack without removing it. **d. Searching and Checking:** i. Search for the integer 30 in the integerStack and print its position relative to the top of the stack. ii. Check whether the integerStack is empty and print the result. **e. Size and Capacity:** i. Print the current size of the integerStack.	Easy

	ii. Determine and print the capacity of the integerStack. **f. Iteration and Conversion:** i. Iterate through the elements of the integerStack using a for-each loop and print each element. ii. Convert the integerStack into an array and print the resulting array. **g. Clearing the Stack:** i. Clear all elements from the integerStack. ii. Verify whether the integerStack is empty after clearing.	
5	**Priority Queue:** Priority QueueCreate a new Java class named QueuePractice. Import the necessary Java Collection classes. Initialize a Queue object named "integerQueue" where the elements are ordered based on their natural ordering and do the following operations: **a. Adding Elements:** i. Add the following integers to the integerQueue: 10, 20, 30, 40. **b. Removing Elements:** i. Remove the head element from the integerQueue. **c. Accessing Elements:** i. Peek at the head element of the integerQueue without removing it. **d. Checking Queue Status:** i. Check whether the integerQueue is empty. ii. Determine and print the size of the integerQueue. **e. Iteration and Conversion:** i. Iterate through the elements of the integerQueue and print each element.	Easy

	ii. Convert the integerQueue into an array and print the resulting array. **f. Clearing the Queue:** i. Clear all elements from the integerQueue. ii. Verify whether the integerQueue is empty after clearing.	
6	Create a new Java class named QueuePractice. Import the necessary Java Collection classes.Initialize a Priority Queue object named "integerQueue" where elements are ordered specified by a custom Comparator and perform the following operations: **a. Adding Elements:** i. Add the following integers to the integerQueue: 10, 20, 30, 40. **b. Removing Elements:** i. Remove the head element from the integerQueue. **c. Accessing Elements:** i. Peek at the head element of the integerQueue without removing it. **d. Checking Queue Status:** i. Check whether the integerQueue is empty. ii. Determine and print the size of the integerQueue. **e. Custom Comparator:** i. Implement a custom Comparator class that orders integers in descending order. **f. Clearing the Queue:** i. Clear all elements from the integerQueue. ii. Verify whether the integerQueue is empty after clearing.	Easy

7	Create a new Java class named PriorityQueueCharPractice. Import the necessary Java Collection classes. Initialize a Priority Queue object named "charQueue" where elements are ordered based on their ASCII values, with the element with the maximum ASCII value having the highest priority and perform the following operations: **a. Adding Elements:** i. Add the following characters to the charQueue: 'a', 'b', 'c', 'd'. **b. Removing Elements:** i. Remove the head element from the charQueue. **c. Accessing Elements:** i. Peek at the head element of the charQueue without removing it. **d. Checking Queue Status:** i. Check whether the charQueue is empty. ii. Determine and print the size of the charQueue. **e. Custom Comparator:** i. Implement a custom Comparator class that orders characters based on their ASCII values, ensuring the element with the maximum ASCII value has the highest priority. **f. Iteration and Conversion:** i. Iterate through the elements of the charQueue and print each element. ii. Convert the charQueue into an array and print the resulting array. **g. Clearing the Queue:** i. Clear all elements from the charQueue. Verify whether the charQueue is empty after clearing.	Medium
8	**ArrayDeque**	Easy

	Create a new Java class named ArrayDequePractice. Import the necessary Java Collection classes.Initialize an ArrayDeque object named "integerDeque" to store integers and perform the following operations: **a. Adding Elements:** i. Add the following integers to the integerDeque: 12, 24, 45, 67, 87, 43. **b. Adding Elements at Both Ends:** i. Add the integer 100 to the beginning of the integerDeque. ii. Add the integer 200 to the end of the integerDeque. **c. Removing Elements:** i. Remove and retrieve the first element from the integerDeque. ii. Remove and retrieve the last element from the integerDeque. **d. Accessing Elements:** i. Retrieve, but do not remove, the first element of the integerDeque. ii. Retrieve, but do not remove, the last element of the integerDeque. iii. Retrieve an element from the integerDeque at a random index and print it. **e. Checking Deque Status:** i. Check whether the integerDeque is empty. ii. Determine and print the size of the integerDeque. **f. Dynamic Resizing:** i. Add the integers 300, 400, 500, 600, 700, 800, 900 to the integerDeque, observing how it dynamically resizes to accommodate the additional elements.	

	ii. Remove several elements from the integerDeque, ensuring it dynamically shrinks when elements are removed. **g. Iteration and Conversion:** i. Iterate through the elements of the integerDeque and print each element. ii. Convert the integerDeque into an array and print the resulting array. **h. Clearing the Deque:** i. Clear all elements from the integerDeque. ii. Verify whether the integerDeque is empty after clearing.	