Hands On - Generics

1. Arraylist

Problem Statement

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 ope

1. Adding Elements:
    1. Add the following strings to the list: "apple", "banana", "orange", "grape".
2. Removing Elements:
    1. Remove the element at index 2 from the list.
    2. Remove the first occurrence of "banana" from the list.
3. Accessing Elements:
    1. Print the element at index 1.
    2. Replace the element at index 0 with "pear".
4. Searching and Checking:
    1. Check if the list contains "orange" and print the result.
    2. Find and print the index of the last occurrence of "grape".
    3. Check if the list is empty and print the result.
5. List Operations:
    1. Create a new ArrayList of Strings named "newList".
    2. Add the strings "kiwi", "pineapple", "melon" to the newList.
    3. Add all elements from newList to stringList starting from index 2.
6. Size and Capacity:
    1. Print the size of the stringList.
    2. Clear the stringList and print its size again.
7. Iteration and Conversion:
    1. Use an iterator to iterate over the elements in the stringList and print each element.
    2. Create a sublist of stringList from index 1 to 3 and print it.
    3. Convert the stringList to an array and print the array.
8. Sorting and Ordering:
    1. Sort the elements in stringList in natural order and Print the sorted list.
    2. Check if stringList equals newList and print the result.
    3. Print the hash code of stringList.
    4. Implement a custom comparator to sort stringList in reverse alphabetical order and Print

Analysis

1. We need to create a class called ListPractice.
2. We need to import the necessary Java Collection classes.
3. We need to create an ArrayList of Strings named "stringList".
4. We need to add the following strings to the list: "apple", "banana", "orange", "grape".
5. We need to remove the element at index 2 from the list.
6. We need to remove the first occurrence of "banana" from the list.
7. We need to print the element at index 1.
8. We need to replace the element at index 0 with "pear".
9. We need to check if the list contains "orange" and print the result.
10. We need to find and print the index of the last occurrence of "grape".
11. We need to check if the list is empty and print the result.

12. We need to create a new ArrayList of Strings named "newList".
13. We need to add the strings "kiwi", "pineapple", "melon" to the newList.
14. We need to add all elements from newList to stringList starting from index 2.
15. We need to print the size of the stringList.
16. We need to clear the stringList and print its size again.
17. We need to use an iterator to iterate over the elements in the stringList and print each element.
18. We need to create a sublist of stringList from index 1 to 3 and print it.
19. We need to convert the stringList to an array and print the array.
20. We need to sort the elements in stringList in natural order and Print the sorted list.
21. We need to check if stringList equals newList and print the result.
22. We need to print the hash code of stringList.
23. We need to implement a custom comparator to sort stringList in reverse alphabetical order and Print

Code

```
package com.hands_on;
package com.hands_on;

import java.util.ArrayList;
import java.util.Iterator;

public class ListPractice {
    public static void main(String[] args) {
        System.out.println("Created a ArrayList named stringList.");
        ArrayList<String> stringList = new ArrayList<String>();

        // 1. Adding Elements

        stringList.add("apple");
        stringList.add("banana");
        stringList.add("orange");
        stringList.add("grape");

        // 2. Removing Elements
        //      i. removing element at the index 2 from the stringList
        System.out.println("Removed the value at index 2");
        stringList.remove(2);
        //      ii. removing the first occurence of string "banana" from the stringList
        System.out.println("Removed the value `banana`");
        stringList.remove("banana");

        // 3. Accessing Elements
        //      i. Print the element at index 1.
        System.out.println("The element at the index 1 = "+stringList.get(1));
        //      ii. Replace the element at index 0 with "pear".
        System.out.println("The value at the index 0 is replaced as `pear`");
        stringList.set(0, "pear");

        // 4. Search and Checking
        //      i. check if the list contains "orange" and print result
        System.out.println("Is the string contains `orange`: "+stringList.contains("orange"));
        //      ii. Find and print the index of last occurrence of grape
        System.out.println("The index of `grape`: "+stringList.lastIndexOf("grape"));
        //      iii. Check if the list is empty and print the result
        System.out.println("Is the ArrayList is empty: "+stringList.isEmpty());

        // 5. List Operations
        ArrayList<String> newList = new ArrayList<String>();
        newList.add("kiwi");
        newList.add("pineapple");
        newList.add("melon");
        stringList.addAll(2, newList);
```

```
        // 6. Size and Capacity
        //      i. Print the size of the stringList
        System.out.println("Size of stringList: "+stringList.size());
        //      ii. Clear the stringList and print its size again
        System.out.println("Cleared the stringList");
        stringList.clear();
        System.out.println("Size of stringList: "+stringList.size());

        // 7. Iteration and Conversion
        //      i. Use an iterator to iterate over the elements in the newList and print each element
        System.out.print("Iteration over newList: ");
        Iterator<String> iterator = newList.iterator();
        while (iterator.hasNext()) {
            String name = iterator.next();
            System.out.print(name +", ");
        }
        //      ii. Create a sublist of newList from index 1 to 3 and print it
        System.out.print("\nsub list of the newList from 1 to 3 :");
        System.out.println(newList.subList(1, 3));
        //      iii. Convert the newList to an array and print the array
        System.out.print("Converted into String Array: ");
        String[] strArray = newList.toArray(new String[0]);
        for (String str : strArray) {
            System.out.print(str + ", ");
        }

        // 8. Sorting and Ordering
        //      i. Sort the elements in newList in natural order and Print the sorted list
        System.out.print("\nnewList sorted in natural Order: ");
        stringList.sort(null);
        System.out.println(newList);
        //      ii. Check if stringList equals newList and print the result
        System.out.println("Is stringList and newList is equal: "+stringList.equals(newList));
        //      iii. Print the hash code of newList
        System.out.println("HashCode of newList: "+newList.hashCode());
        //      iv. Implement a custom comparator to sort newList in reverse alphabetical order and Print
        newList.sort((s1, s2) -> s2.compareTo(s1));
        System.out.println("Sorted the newList in reverse alphabetic order: "+newList);
    }
}
```

Output

```
Created a ArrayList named stringList.
Removed the value at index 2
Removed the value `banana`
The element at the index 1 = grape
The value at the index 0 is replaced as `pear`
Is the string contains `orange`: false
The index of `grape`: 1
Is the ArrayList is empty: false
Size of stringList: 5
Cleared the stringList
Size of stringList: 0
Iteration over newList: kiwi, pineapple, melon,
sub list of the newList from 1 to 3 :[pineapple, melon]
Converted into String Array: kiwi, pineapple, melon,
newList sorted in natural Order: [kiwi, pineapple, melon]
Is stringList and newList is equal: false
HashCode of newList: -1602983656
Sorted the newList in reverse alphabetic order: [pineapple, melon, kiwi]
```

2. LinkedList

Problem Statement

Create a new Java class called LinkedListPractice. Import the necessary Java Collection classes and create a LinkedList of Integers named "integerList" and do the follc

1. Adding Elements:
    1. Add the following integers to the list: 10, 20, 30, 40.
2. Removing Elements:
    1. Remove the integer at the second position from the integerList.
    2. Eliminate the first occurrence of the integer 20 from the integerList.
3. Accessing Elements:
    1. Print out the integer stored at the second position in the integerList.
    2. Replace the integer at the first position in the integerList with the value 50.
4. Searching and Checking:
    1. Determine if the integer 30 exists in the integerList and print the result.
    2. Identify and print the index of the last occurrence of the integer 40 in the integerList.
    3. Check whether the integerList is empty and print the result.
5. List Iteration and Conversion:
    1. Iterate through the elements of the integerList using a ListIterator and print each element.
    2. Create a sublist of the integerList containing elements from the second to the fourth position and print it.
    3. Convert the integerList into an array and print the resulting array.
6. Size and Capacity:
    1. Determine and print the size of the integerList.
    2. Clear all elements from the integerList and print its size again.

Analysis

1. We need to create a class called LinkedListPractice.
2. We need to import the necessary Java Collection classes.
3. We need to create a LinkedList of Integers named "integerList".
4. We need to add the following integers to the list: 10, 20, 30, 40.
5. We need to remove the integer at the second position from the integerList.
6. We need to eliminate the first occurrence of the integer 20 from the integerList.
7. We need to print out the integer stored at the second position in the integerList.
8. We need to replace the integer at the first position in the integerList with the value 50.
9. We need to determine if the integer 30 exists in the integerList and print the result.
10. We need to identify and print the index of the last occurrence of the integer 40 in the integerList.
11. We need to check whether the integerList is empty and print the result.
12. We need to iterate through the elements of the integerList using a ListIterator and print each element.
13. We need to create a sublist of the integerList containing elements from the second to the fourth position and print it.
14. We need to convert the integerList into an array and print the resulting array.
15. We need to determine and print the size of the integerList.
16. We need to clear all elements from the integerList and print its size again.

Code

```java
package com.hands_on;

import java.util.LinkedList;
import java.util.ListIterator;

public class LinkedListPractice {
    public static void main(String[] args) {
```

```
        // 1. Adding Elements:
        System.out.println("Created an LinkedList named integerList");
        LinkedList<Integer> integerList = new LinkedList<Integer>();
        //      1.Add the following integers to the list: 10, 20, 30, 40.
        integerList.add(10);
        integerList.add(20);
        integerList.add(30);
        integerList.add(40);
        // 2. Removing Elements:
        //      1.Remove the integer at the second position from the integerList.
        System.out.println("Removed the index at the 2.");
        integerList.remove(2);
        //      2.Eliminate the first occurrence of the integer 20 from the integerList.
        System.out.println("Removed the value first occurrence of 20.");
        integerList.removeFirstOccurrence(20);
        // 3. Accessing Elements:
        //      1.Print out the integer stored at the second position in the integerList.
        try{
            System.out.println("The integer stored in second position is "+ integerList.get(2));
        } catch (Exception e){
            System.out.println(e);
        }
        //      2.Replace the integer at the first position in the integerList with the value 50.
        System.out.println("The integer in the first position get replaced with 50");
        integerList.set(0, 50);
        // 4. Searching and Checking:
        //      1.Determine if the integer 30 exists in the integerList and print the result.
        System.out.println("Is the integer 30 exists or not: "+ integerList.contains(30));
        //      2.Identify and print the index of the last occurrence of the integer 40 in the integerList.
        System.out.println("The Last occurrence of the integer '40' is "+ integerList.lastIndexOf(40));
        //      3.Check whether the integerList is empty and print the result.
        System.out.println("Is Empty: " + integerList.isEmpty());
        // 5. List Iteration and Conversion:
        //      1.Iterate through the elements of the integerList using a ListIterator and print each element.
        ListIterator<Integer> iterator = integerList.listIterator();
        System.out.print("Iteration using ListIterator: ");
        while (iterator.hasNext()) {
            System.out.print(iterator.next() + " ");
        }
        //      2.Create a sublist of the integerList containing elements from the second to the fourth position and print it.
        try{
            System.out.println(integerList.subList(2,4));
        } catch (Exception e){
            System.out.println("
"+ e);
        }
        //      3.Convert the integerList into an array and print the resulting array.
        int[] intArray = integerList.stream().mapToInt(i -> i).toArray();
        System.out.print("integerList converted into array: ");
        for (int i : intArray) {
            System.out.print(i + " ");
        }
        // 6. Size and Capacity:
        //      1.Determine and print the size of the integerList.
        System.out.println(integerList.size());
        //      2.Clear all elements from the integerList and print its size again.
        integerList.clear();
        System.out.println(integerList.size());

    }
}
```

Output

```
Created an LinkedList named integerList
Removed the index at the 2.
Removed the value first occurrence of 20.
java.lang.IndexOutOfBoundsException: Index: 2, Size: 2
The integer in the first position get replaced with 50
Is the integer 30 exists or not: false
The Last occurrence of the integer '40' is 1
Is Empty: false
Iteration using ListIterator: 50 40
java.lang.IndexOutOfBoundsException: toIndex = 4
integerList converted into array: 50 40 2
0
```

---

3. Vector:

Problem Statement

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

1. Adding Elements:
    1. Add the following flowers to the flowerVector: Rose, Lily, Tulip, Daisy.
2. Removing Elements:
    1. Remove the flower at the second position from the flowerVector.
    2. Remove the first occurrence of the flower "Lily" from the flowerVector.
3. Accessing Elements:
    1. Print out the flower stored at the second position in the flowerVector.
    2. Replace the flower at the first position in the flowerVector with "Sunflower".
4. Searching and Checking:
    1. Check if the flower "Tulip" exists in the flowerVector and print the result.
    2. Identify and print the index of the last occurrence of the flower "Daisy" in the flowerVector.
    3. Check whether the flowerVector is empty and print the result.
5. Iteration and Conversion:
    1. Iterate through the elements of the flowerVector using a for-each loop and print each flower.
    2. Create a sublist of the flowerVector containing elements from the second to the fourth position and print it.
    3. Convert the flowerVector into an array and print the resulting array.
6. Size and Capacity:
    1. Determine and print the size of the flowerVector.
    2. Print the current capacity of the flowerVector.
7. Dynamic Array Operations:
    1. Add two more flowers ("Orchid" and "Carnation") to the flowerVector.
    2. Check and print the new capacity of the flowerVector after adding the flowers.
    3. Remove the flower at index 3 from the flowerVector.
    4. Print the size of the flowerVector after removal.

Analysis

1. We need to create a class called VectorPractice.
2. We need to import the necessary Java Collection classes.
3. We need to initialize a Vector named "flowerVector" to store flower objects.
4. We need to add the following flowers to the flowerVector: Rose, Lily, Tulip, Daisy.
5. We need to remove the flower at the second position from the flowerVector.
6. We need to remove the first occurrence of the flower "Lily" from the flowerVector.
7. We need to print out the flower stored at the second position in the flowerVector.

8. We need to replace the flower at the first position in the flowerVector with "Sunflower".
9. We need to check if the flower "Tulip" exists in the flowerVector and print the result.
10. We need to identify and print the index of the last occurrence of the flower "Daisy" in the flowerVector.
11. We need to check whether the flowerVector is empty and print the result.
12. We need to iterate through the elements of the flowerVector using a for-each loop and print each flower.
13. We need to create a sublist of the flowerVector containing elements from the second to the fourth position and print it.
14. We need to convert the flowerVector into an array and print the resulting array.
15. We need to determine and print the size of the flowerVector.
16. We need to print the current capacity of the flowerVector.
17. We need to add two more flowers ("Orchid" and "Carnation") to the flowerVector.
18. We need to check and print the new capacity of the flowerVector after adding the flowers.
19. We need to remove the flower at index 3 from the flowerVector.
20. We need to print the size of the flowerVector after removal.

Code

```java
package com.hands_on;

import java.util.Vector;

public class VectorPractice {
    public static void main(String[] args) {
        // 1. Adding Elements:
        System.out.println("Created a Vector named flowerVector");
        Vector<String> flowerVector = new Vector<String>();
        //      1.Add the following flowers to the flowerVector: Rose, Lily, Tulip, Daisy.
        flowerVector.add("Rose");
        flowerVector.add("Lily");
        flowerVector.add("Tulip");
        flowerVector.add("Daisy");
        // 2. Removing Elements:
        //      1.Remove the flower at the second position from the flowerVector.
        System.out.println("Removed the index at the 2.");
        flowerVector.remove(2);
        //      2.Remove the first occurrence of the flower "Lily" from the flowerVector.
        System.out.println("Removed the value first occurrence of Lily.");
        flowerVector.remove("Lily");
        // 3. Accessing Elements:
        //      1.Print out the flower stored at the second position in the flowerVector.
        try{
            System.out.println("The flower stored in second position is "+ flowerVector.get(2));
        } catch (Exception e){
            System.out.println(e);
        }
        //      2.Replace the flower at the first position in the flowerVector with "Sunflower".
        System.out.println("The flower in the first position get replaced with Sunflower");
        flowerVector.set(0, "Sunflower");
        // 4. Searching and Checking:
        //      1.Check if the flower "Tulip" exists in the flowerVector and print the result.
        System.out.println("Is the flower Tulip exists or not: "+ flowerVector.contains("Tulip"));
        //      2.Identify and print the index of the last occurrence of the flower "Daisy" in the flowerVector.
        System.out.println("The Last occurrence of the flower 'Daisy' is "+ flowerVector.lastIndexOf("Daisy"));
        //      3.Check whether the flowerVector is empty and print the result.
        System.out.println("Is Empty: " + flowerVector.isEmpty());
        // 5. Iteration and Conversion:
        //      1.Iterate through the elements of the flowerVector using a for-each loop and print each flower.
        System.out.print("Iteration using for-each loop: ");
        for (String flower : flowerVector) {
            System.out.print(flower + " ");
        }
        //      2.Create a sublist of the flowerVector containing elements from the second to the fourth position and print it.
```

```
        try{
            System.out.println("
Sublist of the flowerVector"+flowerVector.subList(2,4));
        } catch (Exception e){
            System.out.println("
"+ e);
        }
        //      3.Convert the flowerVector into an array and print the resulting array.
        String[] flowerArray = flowerVector.toArray(new String[0]);
        System.out.print("flowerVector converted into array: ");
        for (String flower : flowerArray) {
            System.out.print(flower + " ");
        }
        // 6. Size and Capacity:
        //      1.Determine and print the size of the flowerVector.
        System.out.println("The Size of flowerVector: "+flowerVector.size());
        //      2.Print the current capacity of the flowerVector.
        System.out.println("The Capacity of flowerVector: "+flowerVector.capacity());
        // 7. Dynamic Array Operations:
        //      1.Add two more flowers ("Orchid" and "Carnation") to the flowerVector.
        flowerVector.add("Orchid");
        flowerVector.add("Carnation");
        //      2.Check and print the new capacity of the flowerVector after adding the flowers.
        System.out.println("New Capacity: "+ flowerVector.capacity());
        //      3.Remove the flower at index 3 from the flowerVector.
        flowerVector.remove(3);
        //      4.Print the size of the flowerVector after removal.
        System.out.println("Size after removal: "+ flowerVector.size());
    }
}
```

Output

```
Created a Vector named flowerVector
Removed the index at the 2.
Removed the value first occurrence of Lily.
java.lang.ArrayIndexOutOfBoundsException: Array index out of range: 2
The flower in the first position get replaced with Sunflower
Is the flower Tulip exists or not: false
The Last occurrence of the flower 'Daisy' is 1
Is Empty: false
Iteration using for-each loop: Sunflower Daisy
java.lang.IndexOutOfBoundsException: toIndex = 4
flowerVector converted into array: Sunflower Daisy The Size of flowerVector: 2
The Capacity of flowerVector: 10
New Capacity: 10
Size after removal: 3
```

4. Stack:

Problem Statement

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 f

1. Adding Elements:
    1. Push the following integers onto the integerStack: 10, 20, 30, 40.
2. Removing Elements:
    1. Pop the top element from the integerStack.
3. Accessing Elements:
    1. Peek at the top element of the integerStack without removing it.
4. Searching and Checking:

    1. Search for the integer 30 in the integerStack and print its position relative to the top of the stack.
    2. Check whether the integerStack is empty and print the result.
5. Size and Capacity:
    1. Print the current size of the integerStack.
    2. Determine and print the capacity of the integerStack.
6. Iteration and Conversion:
    1. Iterate through the elements of the integerStack using a for-each loop and print each element.
    2. Convert the integerStack into an array and print the resulting array.
7. Clearing the Stack:
    1. Clear all elements from the integerStack.
    2. Verify whether the integerStack is empty after clearing.

Analysis

1. We need to create a class called StackPractice.
2. We need to import the necessary Java Collection classes.
3. We need to initialize a Stack named "integerStack" to store integer values.
4. We need to push the following integers onto the integerStack: 10, 20, 30, 40.
5. We need to pop the top element from the integerStack.
6. We need to peek at the top element of the integerStack without removing it.
7. We need to search for the integer 30 in the integerStack and print its position relative to the top of the stack.
8. We need to check whether the integerStack is empty and print the result.
9. We need to print the current size of the integerStack.
10. We need to determine and print the capacity of the integerStack.
11. We need to iterate through the elements of the integerStack using a for-each loop and print each element.
12. We need to convert the integerStack into an array and print the resulting array.
13. We need to clear all elements from the integerStack.
14. We need to verify whether the integerStack is empty after clearing.

Code:

```java
package com.hands_on;

import java.util.Stack;

public class StackPractice {
    public static void main(String[] args) {
        // Initialize a Stack named "integerStack" to store integer values
        Stack<Integer> integerStack = new Stack<>();

        // Push integers onto the integerStack: 10, 20, 30, 40
        integerStack.push(10);
        integerStack.push(20);
        integerStack.push(30);
        integerStack.push(40);

        // Pop the top element from the integerStack
        Integer poppedElement = integerStack.pop();
        System.out.println("Popped element: " + poppedElement);

        // Peek at the top element of the integerStack without removing it
        Integer topElement = integerStack.peek();
        System.out.println("Top element (without removing): " + topElement);

        // Search for the integer 30 in the integerStack and print its position relative to the top of the stack
        int position = integerStack.search(30);
        System.out.println("Position of 30 relative to top: " + position);
```

```
        // Check whether the integerStack is empty and print the result
        boolean isEmpty = integerStack.isEmpty();
        System.out.println("Is integerStack empty? " + isEmpty);

        // Print the current size of the integerStack
        int size = integerStack.size();
        System.out.println("Size of integerStack: " + size);

        // Determine and print the capacity of the integerStack
        int capacity = integerStack.capacity(); // Not directly accessible, as Stack extends Vector
        System.out.println("Capacity of integerStack: " + capacity);

        // Iterate through the elements of the integerStack using a for-each loop and print each element
        System.out.println("Elements of integerStack:");
        for (Integer element : integerStack) {
            System.out.println(element);
        }

        // Convert the integerStack into an array and print the resulting array
        Integer[] integerArray = integerStack.toArray(new Integer[0]);
        System.out.println("Converted array:");
        for (Integer element : integerArray) {
            System.out.println(element);
        }

        // Clear all elements from the integerStack
        integerStack.clear();

        // Verify whether the integerStack is empty after clearing
        System.out.println("Is integerStack empty after clearing? " + integerStack.isEmpty());
    }
}
```

Output

```
Popped element: 40
Top element (without removing): 30
Position of 30 relative to top: 1
Is integerStack empty? false
Size of integerStack: 3
Capacity of integerStack: 10
Elements of integerStack:
10
20
30
Converted array:
10
20
30
Is integerStack empty after clearing? true
```

---

5. Priority Queue:

Problem Statement:

Priority QueueCreate a new Java class named QueuePractice. Import the necessary Java Collection classes. Initialize a Queue object named "integerQueue" where the natural ordering and do the following operations:

1. Adding Elements:
    1. Add the following integers to the integerQueue: 10, 20, 30, 40.

2. Removing Elements:
    1. Remove the head element from the integerQueue.
3. Accessing Elements:
    1. Peek at the head element of the integerQueue without removing it.
4. Checking Queue Status:
    1. Check whether the integerQueue is empty.
    2. Determine and print the size of the integerQueue.
5. Iteration and Conversion:
    1. Iterate through the elements of the integerQueue and print each element.
    2. Convert the integerQueue into an array and print the resulting array.
6. Clearing the Queue:
    1. Clear all elements from the integerQueue.
    2. Verify whether the integerQueue is empty after clearing.

Analysis

1. We need to create a class called QueuePractice.
2. We need to import the necessary Java Collection classes.
3. We need to initialize a Queue object named "integerQueue" where the elements are ordered based on their natural ordering.
4. We need to add the following integers to the integerQueue: 10, 20, 30, 40.
5. We need to remove the head element from the integerQueue.
6. We need to peek at the head element of the integerQueue without removing it.
7. We need to check whether the integerQueue is empty.
8. We need to determine and print the size of the integerQueue.
9. We need to iterate through the elements of the integerQueue and print each element.
10. We need to convert the integerQueue into an array and print the resulting array.
11. We need to clear all elements from the integerQueue.
12. We need to verify whether the integerQueue is empty after clearing.

Code

```java
package com.hands_on;

import java.util.PriorityQueue;

public class PriorityQueuePractice {
    public static void main(String[] args) {
        // Initialize a Queue object named "integerQueue" where the elements are ordered based on their natural ordering
        PriorityQueue<Integer> integerQueue = new PriorityQueue<>();

        // Add integers to the integerQueue: 10, 20, 30, 40
        integerQueue.add(10);
        integerQueue.add(20);
        integerQueue.add(30);
        integerQueue.add(40);

        // Remove the head element from the integerQueue
        Integer headElement = integerQueue.poll();
        System.out.println("Head element: " + headElement);

        // Peek at the head element of the integerQueue without removing it
        Integer peekElement = integerQueue.peek();
        System.out.println("Peek element: " + peekElement);

        // Check whether the integerQueue is empty
        boolean isEmpty = integerQueue.isEmpty();
        System.out.println("Is integerQueue empty? " + isEmpty);
```

```
        // Determine and print the size of the integerQueue
        int size = integerQueue.size();
        System.out.println("Size of integerQueue: " + size);

        // Iterate through the elements of the integerQueue and print each element
        System.out.print("Elements of integerQueue: ");
        for (Integer element : integerQueue) {
            System.out.print(element+" ");
        }

        // Convert the integerQueue into an array and print the resulting array
        Integer[] integerArray = integerQueue.toArray(new Integer[0]);
        System.out.print("\nConverted array: ");
        for (Integer element : integerArray) {
            System.out.print(element+" ");
        }

        // Clear all elements from the integerQueue
        integerQueue.clear();

        // Verify whether the integerQueue is empty after clearing
        System.out.println("\nIs integerQueue empty after clearing? " + integerQueue.isEmpty());
    }
}
```

Output

```
Head element: 10
Peek element: 20
Is integerQueue empty? false
Size of integerQueue: 3
Elements of integerQueue: 20 40 30
Converted array: 20 40 30
Is integerQueue empty after clearing? true
```

---

6. Custom Priority Queue Practice

Problem Statement

Create a new Java class named QueuePractice. Import the necessary Java Collection classes.Initialize a Priority Queue object named "integerQueue" where elements
Comparator and perform the following operations:

1. Adding Elements:
    1. Add the following integers to the integerQueue: 10, 20, 30, 40.
2. Removing Elements:
    1. Remove the head element from the integerQueue.
3. Accessing Elements:
    1. Peek at the head element of the integerQueue without removing it.
4. Checking Queue Status:
    1. Check whether the integerQueue is empty.
    2. Determine and print the size of the integerQueue.
5. Custom Comparator:
    1. Implement a custom Comparator class that orders integers in descending order.
6. Clearing the Queue:
    1. Clear all elements from the integerQueue.
    2. Verify whether the integerQueue is empty after clearing.

Analysis

1. We need to create a class called QueuePractice.
2. We need to import the necessary Java Collection classes.
3. We need to initialize a Priority Queue object named "integerQueue" where elements are ordered specified by a custom Comparator.
4. We need to add the following integers to the integerQueue: 10, 20, 30, 40.
5. We need to remove the head element from the integerQueue.
6. We need to peek at the head element of the integerQueue without removing it.
7. We need to check whether the integerQueue is empty.
8. We need to determine and print the size of the integerQueue.
9. We need to implement a custom Comparator class that orders integers in descending order.
10. We need to clear all elements from the integerQueue.
11. We need to verify whether the integerQueue is empty after clearing.

Code

```java
package com.hands_on;

import java.util.Comparator;
import java.util.PriorityQueue;


class CustomComparator implements Comparator<Integer> {
    @Override
    public int compare(Integer o1, Integer o2) {
        return o2 - o1;
    }
}

public class CustomPriorityQueuePractice {
    public static void main(String[] args) {
        // Initialize a Priority Queue object named "integerQueue" where elements are ordered specified by a custom Comparator
        PriorityQueue<Integer> integerQueue = new PriorityQueue<>(new CustomComparator());

        // Add integers to the integerQueue: 10, 20, 30, 40
        integerQueue.add(10);
        integerQueue.add(20);
        integerQueue.add(30);
        integerQueue.add(40);

        // Remove the head element from the integerQueue
        Integer headElement = integerQueue.poll();
        System.out.println("Head element: " + headElement);

        // Peek at the head element of the integerQueue without removing it
        Integer peekElement = integerQueue.peek();
        System.out.println("Peek element: " + peekElement);

        // Check whether the integerQueue is empty
        boolean isEmpty = integerQueue.isEmpty();
        System.out.println("Is integerQueue empty? " + isEmpty);

        // Determine and print the size of the integerQueue
        int size = integerQueue.size();
        System.out.println("Size of integerQueue: " + size);

        // Clear all elements from the integerQueue
        integerQueue.clear();

        // Verify whether the integerQueue is empty after clearing
        System.out.println("Is integerQueue empty after clearing? " + integerQueue.isEmpty());
```

```
    }
}
```

Output

```
Head element: 40
Peek element: 30
Is integerQueue empty? false
Size of integerQueue: 3
Is integerQueue empty after clearing? true
```

---

7. Priority Queue Char Practice

Problem Statement

Create a new Java class named PriorityQueueCharPractice. Import the necessary Java Collection classes. Initialize a Priority Queue object named "charQueue" where ASCII values, with the element with the maximum ASCII value having the highest priority and perform the following operations:

1. Adding Elements:
    1. Add the following characters to the charQueue: 'a', 'b', 'c', 'd'.
2. Removing Elements:
    1. Remove the head element from the charQueue.
3. Accessing Elements:
    1. Peek at the head element of the charQueue without removing it.
4. Checking Queue Status:
    1. Check whether the charQueue is empty.
    2. Determine and print the size of the charQueue.
5. Custom Comparator:
    1. Implement a custom Comparator class that orders characters based on their ASCII values, ensuring the element with the maximum ASCII value has the highes
6. Iteration and Conversion:
    1. Iterate through the elements of the charQueue and print each element.
    2. Convert the charQueue into an array and print the resulting array.
7. Clearing the Queue:
    1. Clear all elements from the charQueue. Verify whether the charQueue is empty after clearing.

Analysis

1. We need to create a class called PriorityQueueCharPractice.
2. We need to import the necessary Java Collection classes.
3. We need to initialize a Priority Queue object named "charQueue" where elements are ordered based on their ASCII values.
4. We need to add the following characters to the charQueue: 'a', 'b', 'c', 'd'.
5. We need to remove the head element from the charQueue.
6. We need to peek at the head element of the charQueue without removing it.
7. We need to check whether the charQueue is empty.
8. We need to determine and print the size of the charQueue.
9. We need to implement a custom Comparator class that orders characters based on their ASCII values.
10. We need to iterate through the elements of the charQueue and print each element.
11. We need to convert the charQueue into an array and print the resulting array.
12. We need to clear all elements from the charQueue.

Code

```java
package com.hands_on;

import java.util.Comparator;
import java.util.PriorityQueue;

class CustomCharComparator implements Comparator<Character> {
    @Override
    public int compare(Character o1, Character o2) {
        return o2 - o1;
    }
}

public class PriorityQueueCharPractice {
    public static void main(String[] args) {
        // Initialize a Priority Queue object named "charQueue" where elements are ordered based on their ASCII values
        PriorityQueue<Character> charQueue = new PriorityQueue<>(new CustomCharComparator());

        // Add characters to the charQueue: 'a', 'b', 'c', 'd'
        charQueue.add('a');
        charQueue.add('b');
        charQueue.add('c');
        charQueue.add('d');

        // Remove the head element from the charQueue
        Character headElement = charQueue.poll();
        System.out.println("Head element: " + headElement);

        // Peek at the head element of the charQueue without removing it
        Character peekElement = charQueue.peek();
        System.out.println("Peek element: " + peekElement);

        // Check whether the charQueue is empty
        boolean isEmpty = charQueue.isEmpty();
        System.out.println("Is charQueue empty? " + isEmpty);

        // Determine and print the size of the charQueue
        int size = charQueue.size();
        System.out.println("Size of charQueue: " + size);

        // Iterate through the elements of the charQueue and print each element
        System.out.print("Elements of charQueue:");
        for (Character element : charQueue) {
            System.out.print(element+" ");
        }

        // Convert the charQueue into an array and print the resulting array
        Character[] charArray = charQueue.toArray(new Character[0]);
        System.out.print("\nConverted array:");
        for (Character element : charArray) {
            System.out.print(element+" ");
        }

        // Clear all elements from the charQueue
        charQueue.clear();

        // Verify whether the charQueue is empty after clearing
        System.out.println("\nIs charQueue empty after clearing? " + charQueue.isEmpty());
    }
}
```

Output

```
Head element: d
Peek element: c
Is charQueue empty? false
Size of charQueue: 3
Elements of charQueue:c a b
Converted array:c a b
Is charQueue empty after clearing? true
```

---

8. ArrayDeque

Problem Statement

Create a new Java class named ArrayDequePractice. Import the necessary Java Collection classes.Initialize an ArrayDeque object named "integerDeque" to store integ operations:

1. Adding Elements:
    1. Add the following integers to the integerDeque: 12, 24, 45, 67, 87, 43.
2. Adding Elements at Both Ends:
    1. Add the integer 100 to the beginning of the integerDeque.
    2. Add the integer 200 to the end of the integerDeque.
3. Removing Elements:
    1. Remove and retrieve the first element from the integerDeque.
    2. Remove and retrieve the last element from the integerDeque.
4. Accessing Elements:
    1. Retrieve, but do not remove, the first element of the integerDeque.
    2. Retrieve, but do not remove, the last element of the integerDeque.
    3. Retrieve an element from the integerDeque at a random index and print it.
5. Checking Deque Status:
    1. Check whether the integerDeque is empty.
    2. Determine and print the size of the integerDeque.
6. Dynamic Resizing:
    1. Add the integers 300, 400, 500, 600, 700, 800, 900 to the integerDeque, observing how it dynamically resizes to accommodate the additional elements.
    2. Remove several elements from the integerDeque, ensuring it dynamically shrinks when elements are removed.
7. Iteration and Conversion:
    1. Iterate through the elements of the integerDeque and print each element.
    2. Convert the integerDeque into an array and print the resulting array.
8. Clearing the Deque:
    1. Clear all elements from the integerDeque.
    2. Verify whether the integerDeque is empty after clearing.

Analysis

1. We need to create a class called ArrayDequePractice.
2. We need to import the necessary Java Collection classes.
3. We need to initialize an ArrayDeque object named "integerDeque" to store integers.
4. We need to add the following integers to the integerDeque: 12, 24, 45, 67, 87, 43.
5. We need to add the integer 100 to the beginning of the integerDeque.
6. We need to add the integer 200 to the end of the integerDeque.
7. We need to remove and retrieve the first element from the integerDeque.
8. We need to remove and retrieve the last element from the integerDeque.
9. We need to retrieve, but not remove, the first element of the integerDeque.
10. We need to retrieve, but not remove, the last element of the integerDeque.

11. We need to retrieve an element from the integerDeque at a random index and print it.
12. We need to check whether the integerDeque is empty.
13. We need to determine and print the size of the integerDeque.
14. We need to add the integers 300, 400, 500, 600, 700, 800, 900 to the integerDeque, observing how it dynamically resizes to accommodate the additional elements.
15. We need to remove several elements from the integerDeque, ensuring it dynamically shrinks when elements are removed.
16. We need to iterate through the elements of the integerDeque and print each element.
17. We need to convert the integerDeque into an array and print the resulting array.
18. We need to clear all elements from the integerDeque.
19. We need to verify whether the integerDeque is empty after clearing.

Code

```java
package com.hands_on;

import java.util.ArrayDeque;
import java.util.Iterator;

public class ArrayDequeuePractice {
    public static void main(String[] args) {
        // Initialize an ArrayDeque object named "integerDeque" to store integers
        ArrayDeque<Integer> integerDeque = new ArrayDeque<>();

        // Add integers to the integerDeque: 12, 24, 45, 67, 87, 43
        integerDeque.add(12);
        integerDeque.add(24);
        integerDeque.add(45);
        integerDeque.add(67);
        integerDeque.add(87);
        integerDeque.add(43);

        // Add the integer 100 to the beginning of the integerDeque
        integerDeque.addFirst(100);

        // Add the integer 200 to the end of the integerDeque
        integerDeque.addLast(200);

        // Remove and retrieve the first element from the integerDeque
        Integer firstElement = integerDeque.pollFirst();
        System.out.println("First element: " + firstElement);

        // Remove and retrieve the last element from the integerDeque
        Integer lastElement = integerDeque.pollLast();
        System.out.println("Last element: " + lastElement);

        // Retrieve, but not remove, the first element of the integerDeque
        Integer first = integerDeque.peekFirst();
        System.out.println("First element (without removing): " + first);

        // Retrieve, but not remove, the last element of the integerDeque
        Integer last = integerDeque.peekLast();
        System.out.println("Last element (without removing): " + last);

        // Retrieve an element from the integerDeque at a random index and print it
        Iterator<Integer> iterator = integerDeque.iterator();
        Integer randomElement = null;
        if (iterator.hasNext()) {
            randomElement = iterator.next();
        }
        System.out.println("Random element: " + randomElement);

        // Check whether the integerDeque is empty
        boolean isEmpty = integerDeque.isEmpty();
```

```
        System.out.println("Is integerDeque empty? " + isEmpty);

        // Determine and print the size of the integerDeque
        int size = integerDeque.size();
        System.out.println("Size of integerDeque: " + size);

        // Add the integers 300, 400, 500, 600, 700, 800, 900 to the integerDeque
        for (int i = 300; i <= 900; i += 100) {
            integerDeque.add(i);
        }

        // Remove several elements from the integerDeque
        integerDeque.remove(300);
        integerDeque.remove(400);
        integerDeque.remove(500);

        // Iterate through the elements of the integerDeque and print each element
        System.out.print("Elements of integerDeque: ");

        for (Integer element : integerDeque) {
            System.out.print(element + " ");
        }

        // Convert the integerDeque into an array and print the resulting array
        Integer[] integerArray = integerDeque.toArray(new Integer[0]);
        System.out.print("\nConverted array: ");
        for (Integer element : integerArray) {
            System.out.print(element + " ");
        }

        // Clear all elements from the integerDeque
        integerDeque.clear();

        // Verify whether the integerDeque is empty after clearing
        System.out.println("\nIs integerDeque empty after clearing? " + integerDeque.isEmpty());

    }
}
```

Output

```
First element: 100
Last element: 200
First element (without removing): 12
Last element (without removing): 43
Random element: 12
Is integerDeque empty? false
Size of integerDeque: 6
Elements of integerDeque: 12 24 45 67 87 43 600 700 800 900
Converted array: 12 24 45 67 87 43 600 700 800 900
Is integerDeque empty after clearing? true
```