Self Practice - Generics and Collections

1. ArrayLists

Problem Statement

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 functi

1. Adding Books:
    1. Adds a new book title to the library inventory.
2. Removing Books:
    1. Removes a specific book title from the inventory. Returns true if the book was successfully removed, false otherwise.
    2. Removes books from the inventory based on a specified condition.
3. Searching and Checking:
    1. Checks if a book with the given title exists in the inventory. Returns true if found, otherwise false.
    2. Checks if the library inventory is empty. Returns true if empty, otherwise false.
4. Listing Books:
    1. Lists all the books in the inventory, typically alphabetically.
5. Sorting and Ordering:
    1. Sorts the books in the inventory alphabetically by title.
    2. Sorts the books in the inventory alphabetically by author.
6. Size and Capacity:
    1. Returns the number of books currently in the inventory.
    2. Increases the capacity of the inventory by the specified amount.
7. Iteration and Conversion: 1a. Iterates over the inventory and prints each book's title and author.
    2. Converts the inventory ArrayList to a regular array of book titles.
    3. Returns a special iterator capable of iterating over the inventory and performing remove operations on the books.
8. Additional Functionality:
    1. Keeping track of the number of copies available for each book.
    2. Methods for lending and returning books, which involve decrementing and incrementing the available copies respectively.

Analysis

1. We need to create a class LibraryInventory
2. The class should have an ArrayList of Book objects.
3. we need to add bookstitle to the arraylist
4. we need to remove bookstitle from the arraylist
5. we need to search for bookstitle in the arraylist
6. we need to list all the bookstitle in the arraylist
7. we need to sort the bookstitle in the arraylist
8. we need to increase the capacity of the arraylist
9. we need to iterate over the arraylist
10. we need to convert the arraylist to a regular array
11. we need to return a special iterator
12. we need to keep track of the number of copies available for each book
13. we need to lend and return books

Code

```java
package com.self_practice;

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

public class LibraryInventory {
    private ArrayList<String> books = new ArrayList<String>();

    public void addBook(String title) {
        books.add(title);
    }

    public boolean removeBook(String title) {
        return books.remove(title);
    }

    public void removeBooksByCondition(String condition) {
        books.removeIf(book -> book.contains(condition));
    }

    public boolean searchBook(String title) {
        return books.contains(title);
    }

    public boolean isEmpty() {
        return books.isEmpty();
    }

    public void listBooks() {
        books.forEach(System.out::println);
    }

    public void sortBooksByTitle() {
        books.sort(String::compareTo);
    }

    public void sortBooksByAuthor() {
        books.sort((book1, book2) -> book1.split(":")[1].compareTo(book2.split(":")[1]));
    }

    public int size() {
        return books.size();
    }

    public void increaseCapacity(int amount) {
        books.ensureCapacity(books.size() + amount);
    }

    public void iterateBooks() {
        Iterator<String> iterator = books.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }

    public String[] convertToArray() {
        return books.toArray(new String[0]);
    }

    public Iterator<String> getIterator() {
        return books.iterator();
    }

    public static void main(String[] args) {
        LibraryInventory library = new LibraryInventory();
        library.addBook("Book1:Author1");
        library.addBook("Book2:Author2");
        library.addBook("Book3:Author3");

        library.listBooks();
        library.removeBook("Book2:Author2");
        library.listBooks();

        System.out.println(library.searchBook("Book1:Author1"));
        System.out.println(library.searchBook("Book2:Author2"));

        library.sortBooksByTitle();
        library.listBooks();

        library.sortBooksByAuthor();
        library.listBooks();

        System.out.println(library.size());
        library.increaseCapacity(5);

        library.iterateBooks();

        String[] bookTitles = library.convertToArray();
        for (String title : bookTitles) {
            System.out.println(title);
        }

        Iterator<String> iterator = library.getIterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }

        library.removeBooksByCondition("Book");
        library.listBooks();

        System.out.println(library.isEmpty());

        library.addBook("Book4:Author4");
        library.addBook("Book5:Author5");
        library.addBook("Book6:Author6");

        library.listBooks();

        System.out.println(library.size());

        library.removeBooksByCondition("Author");

        library.listBooks();

        System.out.println(library.isEmpty());
    }
}
```

Output

```
Book1:Author1
Book2:Author2
Book3:Author3
Book1:Author1
Book3:Author3
true
false
Book1:Author1
Book3:Author3
```

```
Book1:Author1
Book3:Author3
2
Book1:Author1
Book3:Author3
Book1:Author1
Book3:Author3
Book1:Author1
Book3:Author3
true
Book4:Author4
Book5:Author5
Book6:Author6
3
true
```

2. LinkedListPractice

Problem Statement

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

1. Adding Students:
    1. Implement a method to add a new student to the list.
2. Removing Students:
    1. Implement a method to remove a specific student from the list by their name.
    2. Implement a method to remove all students with a specified age.
3. Searching and Checking:
    1. Implement a method to check if a student with a given name exists in the list.
    2. Implement a method to check if the list is empty.
4. Listing Students:
    1. Implement a method to print the names of all students in the list.
5. Size and Capacity:
    1. Implement a method to get the total number of students in the list.
    2. Implement a method to increase the capacity of the list by a specified amount.
6. Iteration and Conversion:
    1. Implement a method to iterate over the list and print each student's name and age.
    2. Implement a method to convert the linked list to an array of student names.
    3. Implement a method to return a special iterator that iterates over the list and performs remove operations on the students.
    4. Implement a method to return a descending iterator that iterates over the list in reverse order.
7. Sorting and Ordering:
    1. Implement a method to sort the students in the list alphabetically by their names.
    2. Implement a method to sort the students in the list by their ages in ascending order.
8. Additional Functionality:
    1. Include functionality to keep track of each student's age and grade.
    2. Implement methods to update a student's age or grade.
    3. Implement a method to clear the entire list of students.

Analysis

1. We need to create a class LinkedListPractice
2. The class should have a LinkedList of Student objects.
3. we need to add students to the linkedlist
4. we need to remove students from the linkedlist

5. we need to search for students in the linkedlist
6. we need to list all the students in the linkedlist
7. we need to get the total number of students in the linkedlist
8. we need to increase the capacity of the linkedlist
9. we need to iterate over the linkedlist
10. we need to convert the linkedlist to an array
11. we need to return a special iterator
12. we need to return a descending iterator
13. we need to sort the students in the linkedlist
14. we need to keep track of each student's age and grade
15. we need to update a student's age or grade
16. we need to clear the entire list of students

Code

```java
package com.self_practice;

import java.util.Iterator;
import java.util.LinkedList;

public class LinkedListPractice {
    private LinkedList<String> students = new LinkedList<String>();

    public void addStudent(String name) {
        students.add(name);
    }

    public boolean removeStudent(String name) {
        return students.remove(name);
    }

    public void removeStudentsByAge(int age) {
        students.removeIf(student -> student.contains(String.valueOf(age)));
    }

    public boolean searchStudent(String name) {
        return students.contains(name);
    }

    public boolean isEmpty() {
        return students.isEmpty();
    }

    public void listStudents() {
        students.forEach(System.out::println);
    }

    public int size() {
        return students.size();
    }

    public void iterateStudents() {
        Iterator<String> iterator = students.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }

    public String[] convertToArray() {
        return students.toArray(new String[0]);
    }
    public Iterator<String> getIterator() {
        return students.iterator();
    }

    public Iterator<String> getDescendingIterator() {
        return students.descendingIterator();
    }

    public void sortStudentsByName() {
        students.sort(String::compareTo);
    }

    public void sortStudentsByAge() {
        students.sort((student1, student2) -> Integer.parseInt(student1.split(":")[1]) - Integer.parseInt(student2.split(":")[1])
    }

    public void clearStudents(){
        students.clear();
    }

    public static void main(String[] args) {
        LinkedListPractice list = new LinkedListPractice();
        list.addStudent("Student1:20");
        list.addStudent("Student2:25");
        list.addStudent("Student3:30");

        list.listStudents();
        list.removeStudent("Student2:25");
        list.listStudents();

        System.out.println(list.searchStudent("Student1:20"));
        System.out.println(list.searchStudent("Student2:25"));

        System.out.println(list.size());

        list.iterateStudents();

        String[] studentNames = list.convertToArray();
        for (String name : studentNames) {
            System.out.println(name);
        }

        Iterator<String> iterator = list.getIterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }

        list.removeStudentsByAge(20);
        list.listStudents();

        System.out.println(list.isEmpty());

        list.addStudent("Student4:35");
        list.addStudent("Student5:40");
        list.addStudent("Student6:45");

        list.listStudents();

        System.out.println(list.size());

        list.removeStudentsByAge(30);

        list.listStudents();

        System.out.println(list.isEmpty());
        list.sortStudentsByName();

        list.listStudents();

        list.sortStudentsByAge();

        list.listStudents();

        list.addStudent("Student7:50");

        list.listStudents();

        list.clearStudents();

        list.listStudents();

        System.out.println(list.isEmpty());
    }
}
```

Output

```
Student1:20
Student2:25
Student3:30
Student1:20
Student3:30
true
false
2
Student1:20
Student3:30
Student1:20
Student3:30
Student1:20
Student3:30
Student3:30
false
Student3:30
Student4:35
Student5:40
Student6:45
4
Student4:35
Student5:40
Student6:45
false
Student4:35
Student5:40
Student6:45
Student4:35
Student5:40
Student6:45
Student4:35
Student5:40
Student6:45
Student7:50
true
```

---

3. VectorPractice

Problem Statement

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 r

1. Adding Products:
   1. Implement a method to add a new product to the vector.
2. Removing Products:
   1. Implement a method to remove a specific product from the vector by its name.
   2. Implement a method to remove all products with a specified category.
3. Searching and Checking: 1.. Implement a method to check if a product with a given name exists in the vector.
   2. Implement a method to check if the vector is empty.
4. Listing Products:
   1. Implement a method to print the details of all products in the vector.
5. Size and Capacity:
   1. Implement a method to get the total number of products in the vector.
   2. Implement a method to increase the capacity of the vector by a specified amount.
   3. 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.
6. Iteration and Conversion:
   1. Implement a method to iterate over the vector and print each product's details.
   2. Implement a method to convert the vector to an array of product objects.
7. Sorting and Ordering:
   1. Implement a method to sort the products in the vector alphabetically by their names.
   2. Implement a method to sort the products in the vector by their prices in ascending order.
8. Additional Functionality:
   1. Include functionality to keep track of each product's category and price.
   2. Implement methods to update a product's category or price.
   3. Implement a method to clear the entire vector of products.

Analysis

1. We need to create a class VectorPractice
2. The class should have a Vector of Product objects.
3. we need to add products to the vector
4. we need to remove products from the vector
5. we need to search for products in the vector
6. we need to list all the products in the vector
7. we need to get the total number of products in the vector
8. we need to increase the capacity of the vector
9. we need to trim the capacity of the vector
10. we need to iterate over the vector
11. we need to convert the vector to an array
12. we need to sort the products in the vector
13. we need to keep track of each product's category and price
14. we need to update a product's category or price
15. we need to clear the entire vector of products

Code

```java
package com.self_practice;

import java.util.Iterator;
import java.util.Vector;

public class VectorPractice {
    private Vector<String> products = new Vector<String>();

    public void addProduct(String name) {
        products.add(name);
    }

    public boolean removeProduct(String name) {
        return products.remove(name);
    }

    public void removeProductsByCategory(String category) {
        products.removeIf(product -> product.contains(category));
    }

    public boolean searchProduct(String name) {
        return products.contains(name);
    }

    public boolean isEmpty() {
        return products.isEmpty();
    }

    public void listProducts() {
        products.forEach(System.out::println);
    }

    public int size() {
        return products.size();
    }

    public void increaseCapacity(int amount) {
        products.ensureCapacity(products.size() + amount);
    }

    public void trimCapacity() {
        products.trimToSize();
    }

    public void iterateProducts() {
        Iterator<String> iterator = products.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }

    public String[] convertToArray() {
        return products.toArray(new String[0]);
    }

    public void sortProductsByName() {
        products.sort(String::compareTo);
    }

    public void sortProductsByPrice() {
        products.sort((product1, product2) -> Integer.parseInt(product1.split(":")[1]) - Integer.parseInt(product2.split(":")[1])
    }

    public void clearProducts(){
        products.clear();
    }

    public static void main(String[] args) {
        VectorPractice list = new VectorPractice();
        list.addProduct("Product1:100");
        list.addProduct("Product2:200");
        list.addProduct("Product3:300");
        list.listProducts();
        list.removeProduct("Product2:200");
        list.listProducts();

        System.out.println(list.searchProduct("Product1:100"));
        System.out.println(list.searchProduct("Product2:200"));

        System.out.println(list.size());

        list.iterateProducts();

        String[] productNames = list.convertToArray();
        for (String name : productNames) {
            System.out.println(name);
        }

        list.removeProductsByCategory("Product");
        list.listProducts();

        System.out.println(list.isEmpty());

        list.addProduct("Product4:400");
        list.addProduct("Product5:500");
        list.addProduct("Product6:600");

        list.listProducts();

        System.out.println(list.size());

        list.removeProductsByCategory("Product3");
        list.listProducts();

        System.out.println(list.isEmpty());

        list.sortProductsByName();

        list.listProducts();

        list.sortProductsByPrice();

        list.listProducts();

        list.addProduct("Product7:700");

        list.listProducts();

        list.clearProducts();

        list.listProducts();

        System.out.println(list.isEmpty());
    }
}
```

Output

```
Product1:100
Product2:200
Product3:300
Product1:100
Product3:300
true
false
```

```
2
Product1:100
Product3:300
Product1:100
Product3:300
true
Product4:400
Product5:500
Product6:600
3
Product4:400
Product5:500
Product6:600
false
Product4:400
Product5:500
Product6:600
Product4:400
Product5:500
Product6:600
Product4:400
Product5:500
Product6:600
Product7:700
true
```

---

4. StackPractice

Problem Statement

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
4. Pushing Books:
    1. Implement a method to push a new book onto the stack.
5. Popping Books:
    1. Implement a method to pop the top book from the stack.
    2. Implement a method to remove and return the top book from the stack using the poll() method.
6. Peeking:
    1. Implement a method to peek at the top book of the stack without removing it.
7. Searching and Checking:
    1. Implement a method to check if a book To Kill a Mockingbird exists in the stack.
    2. Implement a method to check if the stack is empty.
8. Listing Books:
    1. Implement a method to print the titles of all books in the stack.
9. Size and Capacity:
    1. Implement a method to get the total number of books in the stack.
    2. Implement a method to increase the capacity of the stack by a specified amount.
10. Iteration and Conversion:
    1. Implement a method to iterate over the stack and print each book's title.

11. Additional Functionality:
    1. Include functionality to keep track of each book's author and publication year.
    2. Implement methods to update a book's author or publication year.
    3. Implement a method to clear the entire stack of books.

Analysis

1. We need to create a class StackPractice
2. The class should have a Stack of Book objects.
3. we need to push books to the stack
4. we need to pop books from the stack
5. we need to peek at the top book of the stack
6. we need to search for books in the stack
7. we need to list all the books in the stack
8. we need to get the total number of books in the stack
9. we need to increase the capacity of the stack
10. we need to iterate over the stack
11. we need to keep track of each book's author and publication year
12. we need to update a book's author or publication year
13. we need to clear the entire stack of books

Code

```java
package com.self_practice;

import java.util.Iterator;
import java.util.Stack;

public class StackPractice {
    private Stack<String> books = new Stack<String>();

    public void pushBook(String title) {
        books.push(title);
    }

    public String popBook() {
        return books.pop();
    }

    public String peekBook() {
        return books.peek();
    }

    public boolean searchBook(String title) {
        return books.contains(title);
    }

    public boolean isEmpty() {
        return books.isEmpty();
    }

    public void listBooks() {
        books.forEach(System.out::println);
    }

    public int size() {
        return books.size();
    }
    public void increaseCapacity(int amount) {
        books.ensureCapacity(books.size() + amount);
    }

    public void iterateBooks() {
        Iterator<String> iterator = books.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }

    public static void main(String[] args) {
        StackPractice stack = new StackPractice();
        stack.pushBook("The Great Gatsby");
        stack.pushBook("To Kill a Mockingbird");
        stack.pushBook("1984");

        stack.listBooks();
        stack.popBook();
        stack.listBooks();

        System.out.println(stack.searchBook("To Kill a Mockingbird"));
        System.out.println(stack.searchBook("1984"));

        System.out.println(stack.size());

        stack.iterateBooks();

        stack.pushBook("Animal Farm");
        stack.pushBook("Brave New World");
        stack.pushBook("The Catcher in the Rye");

        stack.listBooks();

        System.out.println(stack.size());

        stack.pushBook("The Grapes of Wrath");

        stack.listBooks();

        stack.increaseCapacity(5);

        stack.listBooks();

        stack.popBook();
        stack.popBook();
        stack.popBook();

        stack.listBooks();

        System.out.println(stack.isEmpty());

        stack.pushBook("The adventures of Tom Sawyer");

        stack.listBooks();

        stack.popBook();
        stack.popBook();

        stack.listBooks();

        System.out.println(stack.isEmpty());
    }
}
```

Output

```
The Great Gatsby
To Kill a Mockingbird
1984
The Great Gatsby
To Kill a Mockingbird
true
false
2
The Great Gatsby
To Kill a Mockingbird
The Great Gatsby
To Kill a Mockingbird
Animal Farm
Brave New World
The Catcher in the Rye
5
The Great Gatsby
To Kill a Mockingbird
Animal Farm
Brave New World
The Catcher in the Rye
The Grapes of Wrath
The Great Gatsby
To Kill a Mockingbird
Animal Farm
Brave New World
The Catcher in the Rye
The Grapes of Wrath
The Great Gatsby
To Kill a Mockingbird
Animal Farm
false
The Great Gatsby
To Kill a Mockingbird
Animal Farm
The adventures of Tom Sawyer
The Great Gatsby
To Kill a Mockingbird
false
```

5. Priority Queue Practice

Problem Statement

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'

1. Adding Elements:
    1. Add elements to the priority queue.
2. Removing Elements:
    1. Remove and retrieve the head of the priority queue.
3. Accessing Elements:

    1. Retrieve the head of the priority queue without removing it.
4. Checking Queue Status:
    1. Check whether the priority queue is empty.
    2. Return the number of elements in the priority queue.
5. Custom Comparator:
    1. Implement a custom comparator to order characters based on their ASCII values, ensuring the element with the maximum ASCII value has the highest priority
6. Clearing the Queue:
    1. Remove all elements from the priority queue

Analysis

1. We need to create a class PriorityQueuePractice
2. The class should have a PriorityQueue of Character objects.
3. we need to add elements to the priority queue
4. we need to remove elements from the priority queue
5. we need to access elements from the priority queue
6. we need to check whether the priority queue is empty
7. we need to return the number of elements in the priority queue
8. we need to implement a custom comparator
9. we need to clear the priority queue

Code

```java
package com.self_practice;

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

public class PriorityQueuePractice {
    private PriorityQueue<Character> tasks = new PriorityQueue<Character>(new Comparator<Character>() {
        @Override
        public int compare(Character task1, Character task2) {
            return task2 - task1;
        }
    });

    public void addTask(Character task) {
        tasks.add(task);
    }

    public Character removeTask() {
        return tasks.poll();
    }

    public Character peekTask() {
        return tasks.peek();
    }

    public boolean isEmpty() {
        return tasks.isEmpty();
    }

    public int size() {
        return tasks.size();
    }

    public void clearTasks() {
        tasks.clear();
    }
    public static void main(String[] args) {
        PriorityQueuePractice queue = new PriorityQueuePractice();
        queue.addTask('A');
        queue.addTask('B');
        queue.addTask('C');
        queue.addTask('D');

        System.out.println(queue.size());
        System.out.println(queue.peekTask());
        System.out.println(queue.removeTask());
        System.out.println(queue.size());
        System.out.println(queue.peekTask());

        queue.clearTasks();
        System.out.println(queue.size());
        System.out.println(queue.isEmpty());
    }
}
```

Output

```
4
D
D
3
C
0
true
```

---

6. Array Deque Practice

Problem Statement

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

1. Adding Elements:
    1. Add the characters 'A', 'B', 'C', 'D', 'E', and 'F' to the characterDeque.
2. Adding Elements at Both Ends:
    1. Add the character 'X' to the beginning of the characterDeque.
    2. Add the character 'Y' to the end of the characterDeque.
3. Removing Elements:
    1. Remove and retrieve the first element from the characterDeque.
    2. Remove and retrieve the last element from the characterDeque.
4. Accessing Elements:
    1. Retrieve, without removing, the first element of the characterDeque.
    2. Retrieve, without removing, the last element of the characterDeque.
    3. Retrieve a character from the characterDeque at a random index and print it.
5. Checking Deque Status:
    1. Check whether the characterDeque is empty.
    2. Determine and print the size of the characterDeque.
6. Dynamic Resizing:
    1. Add the characters 'G', 'H', 'I', 'J', 'K', 'L', and 'M' to the characterDeque, observing how it dynamically resizes to accommodate the additional elements.
    2. Remove several elements from the characterDeque, ensuring it dynamically shrinks when elements are removed.
7. Iteration and Conversion:
    1. Iterate through the elements of the characterDeque and print each character.

    2. Use a descending iterator to iterate through the elements of the characterDeque and print each character in reverse order.
    3. Convert the characterDeque into an array and print the resulting array.
8. Clearing the Deque:
    1. Clear all elements from the characterDeque.
    2. Verify whether the characterDeque is empty after clearing

Analysis

1. We need to create a class ArrayDequePractice
2. The class should have a ArrayDeque of Character objects.
3. we need to add elements to the ArrayDeque
4. we need to add elements at both ends of the ArrayDeque
5. we need to remove elements from the ArrayDeque
6. we need to access elements from the ArrayDeque
7. we need to check whether the ArrayDeque is empty
8. we need to determine the size of the ArrayDeque
9. we need to dynamically resize the ArrayDeque
10. we need to iterate over the ArrayDeque
11. we need to use a descending iterator to iterate over the ArrayDeque
12. we need to convert the ArrayDeque to an array
13. we need to clear the ArrayDeque
14. we need to verify whether the ArrayDeque is empty after clearing

Code

```
package com.self_practice;

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

public class ArrayDequePractice {
    private ArrayDeque<Character> characterDeque = new ArrayDeque<Character>();

    public void addCharacters() {
        characterDeque.add('A');
        characterDeque.add('B');
        characterDeque.add('C');
        characterDeque.add('D');
        characterDeque.add('E');
        characterDeque.add('F');
    }

    public void addCharactersAtBothEnds() {
        characterDeque.addFirst('X');
        characterDeque.addLast('Y');
    }

    public Character removeFirstCharacter() {
        return characterDeque.pollFirst();
    }

    public Character removeLastCharacter() {
        return characterDeque.pollLast();
    }

    public Character peekFirstCharacter() {
        return characterDeque.peekFirst();
    }
```

```java
    public Character peekLastCharacter() {
        return characterDeque.peekLast();
    }

    public void accessCharacterAtIndex(int index) {
        Iterator<Character> iterator = characterDeque.iterator();
        int i = 0;
        while (iterator.hasNext()) {
            if (i == index) {
                System.out.println(iterator.next());
                break;
            }
            iterator.next();
            i++;
        }
    }

    public boolean isEmpty() {
        return characterDeque.isEmpty();
    }

    public int size() {
        return characterDeque.size();
    }

    public void addMoreCharacters() {
        characterDeque.add('G');
        characterDeque.add('H');
        characterDeque.add('I');
        characterDeque.add('J');
        characterDeque.add('K');
        characterDeque.add('L');
        characterDeque.add('M');
    }

    public void removeCharacters() {
        characterDeque.pollFirst();
        characterDeque.pollFirst();
        characterDeque.pollLast();
        characterDeque.pollLast();
    }

    public void iterateCharacters() {
        Iterator<Character> iterator = characterDeque.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }

    public void iterateCharactersDescending() {
        Iterator<Character> iterator = characterDeque.descendingIterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }

    public void convertToArray() {
        Character[] characters = characterDeque.toArray(new Character[0]);
        for (Character character : characters) {
            System.out.println(character);
        }
    }

    public void clearCharacters() {
        characterDeque.clear();
    }
    public static void main(String[] args) {
        ArrayDequePractice deque = new ArrayDequePractice();
        deque.addCharacters();
        deque.addCharactersAtBothEnds();

        System.out.println(deque.removeFirstCharacter());

        System.out.println(deque.removeLastCharacter());

        System.out.println(deque.peekFirstCharacter());
        System.out.println(deque.peekLastCharacter());

        deque.accessCharacterAtIndex(2);

        System.out.println(deque.isEmpty());
        System.out.println(deque.size());

        deque.addMoreCharacters();
        deque.removeCharacters();

        deque.iterateCharacters();
        deque.iterateCharactersDescending();
        deque.convertToArray();

        deque.clearCharacters();
        System.out.println(deque.isEmpty());
    }
}
```

Output

```
X
Y
A
F
C
false
6
C
D
E
F
G
H
I
J
K
K
J
I
H
G
F
E
D
C
C
D
E
F
G
H
I
```

```
J
K
true
```

7. Hash Set Practice

Problem Statement

Create a new Java class named HashSetPractice. Import the necessary Java Collection classes. Initialize a HashSet object named "stringSet" to store strings and perfo

1. Adding Elements:
   1. Add the following strings to the stringSet: "apple", "banana", "orange", "grape".
   2. Add all elements from a collection named "additionalSet" to the stringSet.
2. Removing Elements:
   1. Remove the string "banana" from the stringSet.
   2. Remove all elements from the stringSet.
   3. Remove all elements present in a collection named "removalSet" from the stringSet.
3. Checking Set Status:
   1. Check whether the stringSet contains the string "orange".
   2. Determine and print the size of the stringSet.
   3. Check if the stringSet is empty.
4. Iteration and Conversion:
   1. Iterate through the elements of the stringSet and print each string.
   2. Convert the stringSet into an array and print the resulting array.
   3. Print the string representation of the stringSet.
5. Retaining Elements:
   1. Retain only the elements in the stringSet that are contained in a collection named "retainSet".

Analysis

1. We need to create a class HashSetPractice
2. The class should have a HashSet of String objects.
3. we need to add elements to the HashSet
4. we need to remove elements from the HashSet
5. we need to check whether the HashSet contains a specific element
6. we need to determine the size of the HashSet
7. we need to check whether the HashSet is empty
8. we need to iterate over the HashSet
9. we need to convert the HashSet to an array
10. we need to retain only the elements in the HashSet that are contained in another collection

Code

```
package com.self_practice;

import java.util.HashSet;
import java.util.Iterator;

public class HashSetPractice {
    private HashSet<String> stringSet = new HashSet<String>();

    public void addStrings() {
        stringSet.add("apple");
        stringSet.add("banana");
        stringSet.add("orange");
        stringSet.add("grape");
    }

    public void addAllFromCollection(HashSet<String> additionalSet) {
        stringSet.addAll(additionalSet);
    }

    public void removeString(String string) {
        stringSet.remove(string);
    }

    public void removeAll() {
        stringSet.clear();
    }

    public void removeAllFromCollection(HashSet<String> removalSet) {
        stringSet.removeAll(removalSet);
    }

    public boolean containsString(String string) {
        return stringSet.contains(string);
    }

    public int size() {
        return stringSet.size();
    }

    public boolean isEmpty() {
        return stringSet.isEmpty();
    }

    public void iterateStrings() {
        Iterator<String> iterator = stringSet.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }

    public void convertToArray() {
        String[] strings = stringSet.toArray(new String[0]);
        for (String string : strings) {
            System.out.println(string);
        }
    }

    public void retainElements(HashSet<String> retainSet) {
        stringSet.retainAll(retainSet);
    }

    public static void main(String[] args) {
        HashSetPractice set = new HashSetPractice();
        set.addStrings();

        HashSet<String> additionalSet = new HashSet<String>();
        additionalSet.add("mango");
        additionalSet.add("kiwi");

        set.addAllFromCollection(additionalSet);

        set.removeString("banana");

        set.removeAll();

        HashSet<String> removalSet = new HashSet<String>();
        removalSet.add("orange");
        removalSet.add("grape");

        set.removeAllFromCollection(removalSet);

        System.out.println(set.containsString("orange"));
        System.out.println(set.size());
        System.out.println(set.isEmpty());

        set.addStrings();

        set.iterateStrings();
        set.convertToArray();

        HashSet<String> retainSet = new HashSet<String>();
        retainSet.add("apple");
        retainSet.add("banana");

        set.retainElements(retainSet);

        set.iterateStrings();
    }
}
```

Output

```
false
0
true
banana
orange
apple
grape
banana
orange
apple
grape
banana
apple
```

---

8. Linked Hash Set Practice

Problem Statement

Create a new Java class named LinkedHashSetPractice. Import the necessary Java Collection classes. Initialize a LinkedHashSet named "wordSet" to store strings. Add `"fish", "rabbit", "turtle"}` to the wordSet and perform the following operations:

1. Adding Elements:
    1. Add the string "horse" to the wordSet.
2. Removing Elements:
    1. Remove the string "bird" from the wordSet.
3. Checking if Set Contains Elements:
    1. Check if the wordSet contains the string "fish".
4. Checking Set Status:
    1. Check if the wordSet is empty.
    2. Determine the size of the wordSet.
5. Iterating Over Set:
    1. Iterate through the elements of the wordSet using an iterator obtained and print each element.
6. Converting Set to Array:

    1. Convert the wordSet into an array and print the resulting array.
7. Hash Code of Set:
    1. Print the hash code of the wordSet.
8. Clearing the Set:
    1. Clear all elements from the wordSet.

Analysis

1. We need to create a class LinkedHashSetPractice
2. The class should have a LinkedHashSet of String objects.
3. we need to add elements to the LinkedHashSet.
4. we need to remove elements from the LinkedHashSet.
5. we need to check whether the LinkedHashSet contains a specific element.
6. we need to determine the size of the LinkedHashSet.
7. we need to check whether the LinkedHashSet is empty.
8. we need to iterate over the LinkedHashSet.
9. we need to convert the LinkedHashSet to an array.
10. we need to print the hash code of the LinkedHashSet.
11. we need to clear all elements from the LinkedHashSet.

Code

```java
package com.self_practice;

import java.util.Iterator;
import java.util.LinkedHashSet;

public class LinkedHashSetPractice {
    private LinkedHashSet<String> wordSet = new LinkedHashSet<String>();

    public void addWords() {
        wordSet.add("dog");
        wordSet.add("cat");
        wordSet.add("bird");
        wordSet.add("fish");
        wordSet.add("rabbit");
        wordSet.add("turtle");
    }

    public void addWord(String word) {
        wordSet.add(word);
    }

    public void removeWord(String word) {
        wordSet.remove(word);
    }

    public boolean containsWord(String word) {
        return wordSet.contains(word);
    }

    public boolean isEmpty() {
        return wordSet.isEmpty();
    }

    public int size() {
        return wordSet.size();
    }

    public void iterateWords() {
        Iterator<String> iterator = wordSet.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }

    public void convertToArray() {
        String[] words = wordSet.toArray(new String[0]);
        for (String word : words) {
            System.out.println(word);
        }
    }

    public void printHashCode() {
        System.out.println(wordSet.hashCode());
    }

    public void clearWords() {
        wordSet.clear();
    }

    public static void main(String[] args) {
        LinkedHashSetPractice set = new LinkedHashSetPractice();
        set.addWords();

        set.addWord("horse");

        set.removeWord("bird");

        System.out.println(set.containsWord("fish"));
        System.out.println(set.isEmpty());
        System.out.println(set.size());

        set.iterateWords();
        set.convertToArray();
        set.printHashCode();

        set.clearWords();
    }
}
```

Output

```
true
false
6
dog
cat
fish
rabbit
turtle
horse
dog
cat
fish
rabbit
turtle
horse
-1698260835
```

9. Tree Set String Practice

Problem Statement

Create a new Java class named TreeSetStringPractice. Import the necessary Java Collection classes. Define a custom Comparator for strings to reverse the order. Initia
strings, sorted according to the custom Comparator. Add the strings {"apple", "banana", "cherry", "date", "kiwi", "orange"} to the stringSet and perform the following o

1. Adding Elements:
   - Add the string "grape" to the stringSet.
2. Removing Elements:
   - Remove the string "date" from the stringSet.
3. Checking if Set Contains Elements:
   - Check if the stringSet contains the string "banana".
4. Checking Set Status:
   - Check if the stringSet is empty.
   - Determine the size of the stringSet.
5. Iterating Over Set:
   - Iterate through the elements of the stringSet and print each element.
6. Retrieving First and Last Elements:
   - Retrieve and print the first (lowest) element and last (highest) element.
7. Polling First and Last Elements:
   - Retrieve and remove the first (lowest) element and last (highest) element.

Analysis

1. We need to create a class TreeSetStringPractice
2. The class should have a TreeSet of String objects.
3. we need to add elements to the TreeSet.
4. we need to remove elements from the TreeSet.
5. we need to check whether the TreeSet contains a specific element.
6. we need to determine the size of the TreeSet.
7. we need to check whether the TreeSet is empty.
8. we need to iterate over the TreeSet.
9. we need to retrieve the first and last elements of the TreeSet.
10. we need to poll the first and last elements of the TreeSet.

Code

```java
package com.self_practice;

import java.util.Iterator;
import java.util.TreeSet;

public class TreeSetStringPractice {
    private TreeSet<String> stringSet = new TreeSet<String>((str1, str2) -> str2.compareTo(str1));

    public void addStrings() {
        stringSet.add("apple");
        stringSet.add("banana");
        stringSet.add("cherry");
        stringSet.add("date");
        stringSet.add("kiwi");
        stringSet.add("orange");
    }

    public void addString(String string) {
        stringSet.add(string);
    }
    public void removeString(String string) {
        stringSet.remove(string);
    }

    public boolean containsString(String string) {
        return stringSet.contains(string);
    }

    public boolean isEmpty() {
        return stringSet.isEmpty();
    }

    public int size() {
        return stringSet.size();
    }

    public void iterateStrings() {
        Iterator<String> iterator = stringSet.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }

    public void retrieveFirstAndLast() {
        System.out.println(stringSet.first());
        System.out.println(stringSet.last());
    }

    public void pollFirstAndLast() {
        System.out.println(stringSet.pollFirst());
        System.out.println(stringSet.pollLast());
    }

    public static void main(String[] args) {
        TreeSetStringPractice set = new TreeSetStringPractice();
        set.addStrings();

        set.addString("grape");

        set.removeString("date");

        System.out.println(set.containsString("banana"));
        System.out.println(set.isEmpty());
        System.out.println(set.size());

        set.iterateStrings();

        set.retrieveFirstAndLast();

        set.pollFirstAndLast();
    }
}
```

Output

```
true
false
6
orange
kiwi
grape
cherry
banana
apple
```

```
orange
apple
orange
apple
```

10. Tree Map Practice

Problem Statement

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

1. Adding Entries:
    - Add the following entries to the studentMap:
        - "Alice" : 20
        - "Bob" : 22
        - "Charlie" : 18
        - "David" : 25
        - "Eva" : 21
2. Removing Entry:
    - Remove the entry for "Charlie" from the studentMap.
3. Checking if Map Contains Key:
    - Check if the studentMap contains the key "Bob".
4. Checking Map Status:
    - Check if the studentMap is empty.
    - Determine the size of the studentMap.
5. Iterating Over Map Entries:
    - Iterate through the entries of the studentMap and print each key-value pair.
6. Retrieving Entry with Maximum Key:
    - Retrieve and print the entry with the maximum key (lexicographically last).
7. Retrieving Entry with Minimum Key:
    - Retrieve and print the entry with the minimum key (lexicographically first).
8. Polling First and Last Entries:
    - Retrieve and remove the first entry (lexicographically first) from the studentMap.
    - Retrieve and remove the last entry (lexicographically last) from the studentMap.

Analysis

1. We need to create a class TreeMapPractice
2. The class should have a TreeMap of String keys and Integer values.
3. we need to add entries to the TreeMap.
4. we need to remove entries from the TreeMap.
5. we need to check whether the TreeMap contains a specific key.
6. we need to determine the size of the TreeMap.
7. we need to check whether the TreeMap is empty.
8. we need to iterate over the TreeMap.
9. we need to retrieve the entry with the maximum key.
10. we need to retrieve the entry with the minimum key.
11. we need to poll the first and last entries of the TreeMap.

Code

```
package com.self_practice;

import java.util.Iterator;
import java.util.Map;
import java.util.TreeMap;

public class TreeMapPractice {
    private TreeMap<String, Integer> studentMap = new TreeMap<String, Integer>();

    public void addEntries() {
        studentMap.put("Alice", 20);
        studentMap.put("Bob", 22);
        studentMap.put("Charlie", 18);
        studentMap.put("David", 25);
        studentMap.put("Eva", 21);
    }

    public void removeEntry(String key) {
        studentMap.remove(key);
    }

    public boolean containsKey(String key) {
        return studentMap.containsKey(key);
    }

    public boolean isEmpty() {
        return studentMap.isEmpty();
    }

    public int size() {
        return studentMap.size();
    }

    public void iterateEntries() {
        Iterator<Map.Entry<String, Integer>> iterator = studentMap.entrySet().iterator();
        while (iterator.hasNext()) {
            Map.Entry<String, Integer> entry = iterator.next();
            System.out.println(entry.getKey() + " : " + entry.getValue());
        }
    }

    public void retrieveMaxEntry() {
        Map.Entry<String, Integer> maxEntry = studentMap.lastEntry();
        System.out.println(maxEntry.getKey() + " : " + maxEntry.getValue());
    }

    public void retrieveMinEntry() {
        Map.Entry<String, Integer> minEntry = studentMap.firstEntry();
        System.out.println(minEntry.getKey() + " : " + minEntry.getValue());
    }

    public void pollFirstAndLast() {
        Map.Entry<String, Integer> firstEntry = studentMap.pollFirstEntry();
        Map.Entry<String, Integer> lastEntry = studentMap.pollLastEntry();
        System.out.println(firstEntry.getKey() + " : " + firstEntry.getValue());
        System.out.println(lastEntry.getKey() + " : " + lastEntry.getValue());
    }

    public static void main(String[] args) {
        TreeMapPractice map = new TreeMapPractice();
        map.addEntries();

        map.removeEntry("Charlie");

        System.out.println(map.containsKey("Bob"));
        System.out.println(map.isEmpty());
        System.out.println(map.size());

        map.iterateEntries();

        map.retrieveMaxEntry();
        map.retrieveMinEntry();

        map.pollFirstAndLast();
    }
}
```

Output

```
true
false
4
Alice : 20
Bob : 22
David : 25
Eva : 21
Eva : 21
Alice : 20
Alice : 20
Eva : 21
```

---

11. Hash Map Practice

Problem Statement

Create a new Java class named HashMapPractice. Import the necessary Java Collection classes. Initialize a HashMap named "wordCountMap" to store words (String) a (Integer) as values. Add the following entries to the wordCountMap and perform the following operations:

1. Adding Key-Value Pairs:
   - Add the following key-value pairs to the wordCountMap:
     - "apple" : 5
     - "banana" : 8
     - "cherry" : 3
     - "date" : 6
     - "grape" : 4
2. Copying Mappings:
   - Create a new HashMap named "copyMap". ii. Copy all mappings from the wordCountMap to the copyMap.
3. Retrieving Values:
   - Retrieve and print the count associated with the word "date".
4. Removing a Mapping:
   - Remove the mapping for the word "cherry" from the wordCountMap.
5. Checking for Key Presence:
   - Check if the wordCountMap contains the word "banana". ii. Check if the wordCountMap contains the count 4.
6. Checking HashMap Status:
   - Check if the wordCountMap is empty.
   - Determine and print the number of key-value mappings in the wordCountMap.
7. Iterating Over Entries:
   - Iterate through the entries of the wordCountMap and print each word-count pair.
8. Retrieving Keys and Values:
   - Retrieve and print the set of words in the wordCountMap.

- Retrieve and print the collection of counts in the wordCountMap.

Analysis

1. We need to create a class HashMapPractice
2. The class should have a HashMap of String keys and Integer values.
3. we need to add key-value pairs to the HashMap.
4. we need to copy mappings from one HashMap to another.
5. we need to retrieve values from the HashMap.
6. we need to remove mappings from the HashMap.
7. we need to check whether the HashMap contains a specific key or value.
8. we need to determine the size of the HashMap.
9. we need to iterate over the HashMap.
10. we need to retrieve the keys and values from the HashMap.

Code

```
package com.self_practice;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

public class HashMapPractice {
    private HashMap<String, Integer> wordCountMap = new HashMap<String, Integer>();

    public void addMappings() {
        wordCountMap.put("apple", 5);
        wordCountMap.put("banana", 8);
        wordCountMap.put("cherry", 3);
        wordCountMap.put("date", 6);
        wordCountMap.put("grape", 4);
    }

    public void copyMappings() {
        HashMap<String, Integer> copyMap = new HashMap<String, Integer>(wordCountMap);
        System.out.println(copyMap);
    }

    public void retrieveValue(String key) {
        System.out.println(wordCountMap.get(key));
    }

    public void removeMapping(String key) {
        wordCountMap.remove(key);
    }

    public void checkKeyPresence(String key) {
        System.out.println(wordCountMap.containsKey(key));
    }

    public void checkValuePresence(int value) {
        System.out.println(wordCountMap.containsValue(value));
    }

    public boolean isEmpty() {
        return wordCountMap.isEmpty();
    }

    public int size() {
        return wordCountMap.size();
    }
    public void iterateEntries() {
        Iterator<Map.Entry<String, Integer>> iterator = wordCountMap.entrySet().iterator();
        while (iterator.hasNext()) {
            Map.Entry<String, Integer> entry = iterator.next();
            System.out.println(entry.getKey() + " : " + entry.getValue());
        }
    }

    public void retrieveKeysAndValues() {
        System.out.println(wordCountMap.keySet());
        System.out.println(wordCountMap.values());
    }

    public static void main(String[] args) {
        HashMapPractice map = new HashMapPractice();
        map.addMappings();

        map.copyMappings();

        map.retrieveValue("date");

        map.removeMapping("cherry");

        map.checkKeyPresence("banana");
        map.checkValuePresence(4);

        System.out.println(map.isEmpty());
        System.out.println(map.size());

        map.iterateEntries();

        map.retrieveKeysAndValues();
    }
}
```

Output

```
{banana=8, date=6, apple=5, cherry=3, grape=4}
6
true
true
false
4
banana : 8
date : 6
apple : 5
grape : 4
[banana, date, apple, grape]
[8, 6, 5, 4]
```

---

12. Linked Hash Map Practice

Problem Statement

Create a new Java class named LinkedHashMapPractice. Import the necessary Java Collection classes. Initialize a LinkedHashMap named "vehicleTypeMap" to store v 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:

1. Adding Key-Value Pairs:
    - Add the given key-value pairs to the vehicleTypeMap.
2. Copying Mappings:
    - Create a new LinkedHashMap named "copyMap".
    - Copy all mappings from the vehicleTypeMap to the copyMap.
3. Retrieving Values:
    - Retrieve and print the category associated with the vehicle type "motorcycle".
4. Removing a Mapping:
    - Remove the mapping for the vehicle type "van" from the vehicleTypeMap.
5. Checking for Key Presence:
    - Check if the vehicleTypeMap contains the vehicle type "suv".
    - Check if the vehicleTypeMap contains the category "pickup".
6. Checking LinkedHashMap Status:
    - Check if the vehicleTypeMap is empty.
    - Determine and print the number of key-value mappings in the vehicleTypeMap.
7. Iterating Over Entries:
    - Iterate through the entries of the vehicleTypeMap and print each vehicle type-category pair.
8. Retrieving Keys and Values:
    - Retrieve and print the set of vehicle types in the vehicleTypeMap.
    - Retrieve and print the collection of categories in the vehicleTypeMap

Analysis

1. We need to create a class LinkedHashMapPractice.
2. The class should have a LinkedHashMap of String keys and String values.
3. we need to add key-value pairs to the LinkedHashMap.
4. we need to copy mappings from one LinkedHashMap to another.
5. we need to retrieve values from the LinkedHashMap.
6. we need to remove mappings from the LinkedHashMap.
7. we need to check whether the LinkedHashMap contains a specific key or value.
8. we need to determine the size of the LinkedHashMap.
9. we need to iterate over the LinkedHashMap.
10. we need to retrieve the keys and values from the LinkedHashMap.

Code

```
package com.self_practice;

import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.Map;

public class LinkedHashMapPractice {
    private LinkedHashMap<String, String> vehicleTypeMap = new LinkedHashMap<String, String>();

    public void addMappings() {
        vehicleTypeMap.put("car", "sedan");
        vehicleTypeMap.put("truck", "pickup");
        vehicleTypeMap.put("motorcycle", "sportbike");
        vehicleTypeMap.put("van", "minivan");
        vehicleTypeMap.put("suv", "crossover");
    }

    public void copyMappings() {
        LinkedHashMap<String, String> copyMap = new LinkedHashMap<String, String>(vehicleTypeMap);
        System.out.println(copyMap);
    }

    public void retrieveValue(String key) {
        System.out.println(vehicleTypeMap.get(key));
    }

    public void removeMapping(String key) {
        vehicleTypeMap.remove(key);
    }

    public void checkKeyPresence(String key) {
        System.out.println(vehicleTypeMap.containsKey(key));
    }

    public void checkValuePresence(String value) {
        System.out.println(vehicleTypeMap.containsValue(value));
    }

    public boolean isEmpty() {
        return vehicleTypeMap.isEmpty();
    }

    public int size() {
        return vehicleTypeMap.size();
    }

    public void iterateEntries() {
        Iterator<Map.Entry<String, String>> iterator = vehicleTypeMap.entrySet().iterator();
        while (iterator.hasNext()) {
            Map.Entry<String, String> entry = iterator.next();
            System.out.println(entry.getKey() + " : " + entry.getValue());
        }
    }

    public void retrieveKeysAndValues() {
        System.out.println(vehicleTypeMap.keySet());
        System.out.println(vehicleTypeMap.values());
    }

    public static void main(String[] args) {
        LinkedHashMapPractice map = new LinkedHashMapPractice();
        map.addMappings();

        map.copyMappings();

        map.retrieveValue("motorcycle");

        map.removeMapping("van");

        map.checkKeyPresence("suv");
        map.checkValuePresence("pickup");

        System.out.println(map.isEmpty());
        System.out.println(map.size());

        map.iterateEntries();

        map.retrieveKeysAndValues();
    }
}
```

Output

```
{car=sedan, truck=pickup, motorcycle=sportbike, van=minivan, suv=crossover}
sportbike
true
true
false
4
car : sedan
truck : pickup
motorcycle : sportbike
suv : crossover
[car, truck, motorcycle, suv]
[sedan, pickup, sportbike, crossover]
```