DAY 3

SET INTERFACE

HashSet - Tryout 1

```
import java.util.HashSet;
import java.util.Set;
import java.util.Iterator;
class Tester {
	public static void main(String[] args) {
		// Creating HashSet
		Set<String> food = new HashSet<String>();

		// Checking if a HashSet is empty
		System.out.println("Is the set empty? : " + food.isEmpty());
		// Adding elements to the HashSet
		food.add("Pasta");
		food.add("Noodles");
		food.add("Sandwich");
		food.add("Pasta");
		food.add("Burger");
		food.add("Noodles");
		System.out.print("Set output without the duplicates: ");
		System.out.println(food);
		// Finding the size of the HashSet
		System.out.println("The number of food items in the set: " + food.size());
		// Checking if the HashSet contains the given element
		String foodItem = "Pasta";
		if (food.contains(foodItem))
			System.out.println(foodItem + " is already ordered");
		else
			System.out.println(foodItem + " is not ordered");
```

```
        // Removing an element from the HashSet
        if(food.remove("Burger"))
           System.out.println("Output after removing Burger from the set:" + food);
        // Traversing elements
        Iterator<String> item = food.iterator();
        while (item.hasNext())
                System.out.println(item.next());
        // Removing all the elements from the HashSet
        food.clear();
        System.out.println("After clear() => " + food);
    }
}
```

HashSet equals() and hashCode() methods – Tryout

```
import java.util.Set;
import java.util.HashSet;
import java.util.List;
import java.util.ArrayList;
class User {
    private int userId;
    private String userName;
    private String emailId;
    public User(int userId, String userName, String emailId) {
        this.userId = userId;
        this.userName = userName;
        this.emailId = emailId;
    }
    public int getUserId() {
        return userId;
    }
    public void setUserId(int userId) {
        this.userId = userId;
    }
```

```
	public String getUserName() {
		return userName;
	}
	public void setUserName(String userName) {
		this.userName = userName;
	}
	public String getEmailId() {
		return emailId;
	}
	public void setEmailId(String emailId) {
		this.emailId = emailId;
	}
	@Override
	public boolean equals(Object user) {
		User otherUser = (User) user;
		if (this.emailId.equals(otherUser.emailId))
			return true;
		return false;
	}
	@Override
	public int hashCode() {
		return emailId.hashCode();
	}
	@Override
	public String toString() {
		return "User Name: "+userName + ", Email Id: " + emailId;
	}
}
class Tester {
	public static void main(String[] args) {
		List<User> userList = new ArrayList<User>();
		userList.add(new User(1001, "Mike", "Mike@example.com"));
		userList.add(new User(1002, "Ben", "User@example.com"));
		userList.add(new User(1003, "Henry", "Henry@example.com"));
```

```
            userList.add(new User(1004, "Hannah", "User@example.com"));
            userList.add(new User(1005, "Ellie", "Henry@example.com"));
            userList.add(new User(1006, "Ryan", "Ryan@example.com"));

            Set<User> userSet = new HashSet<User>();
            userSet.addAll(userList);
            for (User user : userSet)
                  System.out.println(user);
      }
}
```

Set Interface - Exercise 1

```
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
class Student {
   private int studentId;
   private String studentName;
   private int courseId;
   public Student(int studentId, String studentName, int courseId) {
      this.studentId = studentId;
      this.studentName = studentName;
      this.courseId = courseId;
   }
   public int getStudentId() {
      return studentId;
   }

   public void setStudentId(int studentId) {
      this.studentId = studentId;
   }
   public String getStudentName() {
      return studentName;
```

```
        }
        public void setStudentName(String studentName) {
            this.studentName = studentName;
        }
        public int getCourseId() {
            return courseId;
        }
        public void setCourseId(int courseId) {
            this.courseId = courseId;
        }
        @Override
        public boolean equals(Object student) {
            if (this == student) return true;
            if (student == null || getClass() != student.getClass()) return false;
            Student otherStudent = (Student) student;
            return studentId == otherStudent.studentId;
        }
        @Override
        public int hashCode() {
            return studentId;
        }
        @Override
        public String toString() {
            return "Student Id: " + studentId + ", Student Name: " + studentName;
        }
    }
    class Tester {
        public static Set<Student> findDuplicateEntries(List<Student> students) {
            Set<Student> allStudents = new HashSet<>();
            Set<Student> duplicateStudents = new HashSet<>();
            for (Student student : students) {
                if (!allStudents.add(student)) {
                    duplicateStudents.add(student);
                }
            }
            return duplicateStudents;
        }
        public static void main(String[] args) {
            List<Student> students = new ArrayList<Student>();
            students.add(new Student(1001, "Dean", 111));
            students.add(new Student(1002, "Harley", 112));
            students.add(new Student(1003, "Franklin", 113));
            students.add(new Student(1005, "Arden", 113));
            students.add(new Student(1100, "Juliet", 112));
            students.add(new Student(1003, "Franklin", 111));
            students.add(new Student(1001, "Dean", 114));
            Set<Student> duplicateStudents = findDuplicateEntries(students);
            System.out.println("Students who have applied for re-evaluation in more than one subject");
            for (Student student : duplicateStudents)
                System.out.println(student);
        }
    }
```

HASHMAP

Methods in HashMap – Tryout

```
import java.util.Map;
import java.util.HashMap;
class Tester {
        public static void main(String args[]) {
                Map<String, Integer> books = new HashMap<String, Integer>();
                //Adding key-value pairs to the map
                books.put("Data Structures With Java", 50);
                books.put("Operating System", 80);
                books.put("Let Us C", 70);
                books.put("Java Fundamentals", 40);
                //Displaying all the key-value pairs present in the map
```

```
            System.out.println(books);
            //Traversing the map
            //entrySet() method is used to retrieve all the key value pairs
             for(Map.Entry<String, Integer> book:books.entrySet())
                  System.out.println(book.getKey()+", "+book.getValue());
             //keySet() method returns the keys in the Map
             for(String name:books.keySet())
                  System.out.println("key: "+name);
             //values() method returns the values in the Map
             for(int quantity:books.values())
                  System.out.println("value: "+quantity);
             //Removing element based on key
             books.remove("Let Us C");
             //Removing element based on value
             //Uncomment the code given below, execute and observe the output
    //books.remove(70);
             //Removing element based on key and value
             //Uncomment the code given below, execute and observe the output
             //books.remove("Let Us C", 70);
             System.out.println(books);
             //Replacing key-value pair in the map
             books.replace("Operating System", 80, 100);
             System.out.println(books);
             //Getting a value from the map based on key
             System.out.println(books.get("Java Fundamentals"));
             //Printing size of the map
             System.out.println(books.size());

             //Removing all the key-value pairs from the map
             books.clear();
             //Checking if the map is empty
             System.out.println(books.isEmpty());
    }
}
```

HashMap - Exercise

```
import java.util.Map;
import java.util.HashMap;
import java.util.Map.Entry;
class Student {
    public static Map<String, Double> findMaxMinScorers(Map<String, Double> studentMarks)
{
        double maxMarks = Double.MIN_VALUE;
        double minMarks = Double.MAX_VALUE;
        String maxScorer = "";
        String minScorer = "";
        for (Entry<String, Double> entry : studentMarks.entrySet()) {
            String student = entry.getKey();
            double marks = entry.getValue();
            if (marks > maxMarks) {
                maxMarks = marks;
                maxScorer = student;
            } else if (marks == maxMarks) {
                maxScorer += ", " + student;
            }
            if (marks < minMarks) {
                minMarks = marks;
                minScorer = student;
            } else if (marks == minMarks) {
                minScorer += ", " + student;
            }
        }
        Map<String, Double> result = new HashMap<>();
        result.put(maxScorer, maxMarks);
        result.put(minScorer, minMarks);
        return result;
    }
}
class Tester {
    public static void main(String args[]) {
        Map<String, Double> studentMarks = new HashMap<String, Double>();
        studentMarks.put("Lily", 90.0);
        studentMarks.put("Robin", 68.0);
        studentMarks.put("Marshall", 76.5);
        studentMarks.put("Neil", 67.0);
        studentMarks.put("Ted", 92.0);
        Map<String, Double> maxMinScorers = Student.findMaxMinScorers(studentMarks);
        System.out.println("Details of Top Scorers & Low
Scorers\n====================================");
        for (Entry<String, Double> entry : maxMinScorers.entrySet()) {
            System.out.println(entry.getKey() + " -- " + entry.getValue());
        }
    }
}
```

LINKEDLIST ASSIGNMENT-1

```
import java.util.LinkedList;
import java.util.List;
import java.util.HashSet;
import java.util.Set;
class Tester {
    public static List<Integer> removeDuplicates(List<Integer> list) {
        Set<Integer> seen = new HashSet<>();
       List<Integer> uniqueList = new LinkedList<>();
        for (Integer number : list) {
            if (seen.add(number)) { // add returns false if the element is already in the set
                uniqueList.add(number);
            }
        }
        return uniqueList;
    }
    public static void main(String args[]) {
        List<Integer> list = new LinkedList<Integer>();
        list.add(10);
        list.add(15);
        list.add(21);
        list.add(15);
        list.add(10);
        List<Integer> updatedList = removeDuplicates(list);
        System.out.println("Linked list without duplicates");
        for (Integer value : updatedList) {
            System.out.print(value + " ");
        }
    }
}
```

LINKEDLIST ASSIGNMENT -2

```
import java.util.LinkedList;
import java.util.List;
import java.util.HashSet;
import java.util.Set;
class Tester {
    public static List<Integer> findCommonElements(List<Integer> listOne, List<Integer>
listTwo) {
        Set<Integer> setOne = new HashSet<>();
        Set<Integer> commonElements = new HashSet<>();
        // Add elements from listOne to setOne
        for (Integer num : listOne) {
            setOne.add(num);
        }

        // Find common elements with listTwo
        for (Integer num : listTwo) {
            if (setOne.contains(num)) {
                commonElements.add(num);
            }
        }
        // Convert set to list and return
        List<Integer> result = new LinkedList<>(commonElements);
        return result;
    }
    public static void main(String[] args) {
        List<Integer> listOne = new LinkedList<Integer>();
        listOne.add(10);
        listOne.add(12);
        listOne.add(21);
        listOne.add(1);
        listOne.add(53);
        List<Integer> listTwo = new LinkedList<Integer>();
        listTwo.add(11);
        listTwo.add(21);
        listTwo.add(25);
        listTwo.add(53);
        listTwo.add(47);
        List<Integer> commonElements = findCommonElements(listOne, listTwo);
        System.out.println("Common Elements:");
        for (Integer num : commonElements) {
            System.out.print(num + " ");
        }
    }
}
```

LINKEDLIST ASSIGNMENT -3

```
import java.util.LinkedList;
import java.util.List;
class Tester {
    public static List<Integer> mergeLists(List<Integer> listOne, List<Integer> listTwo) {
        List<Integer> mergedList = new LinkedList<>();
        int i = 0, j = 0;
        int sizeOne = listOne.size(), sizeTwo = listTwo.size();
        // Merge elements from both lists
        while (i < sizeOne && j < sizeTwo) {
            if (listOne.get(i) <= listTwo.get(j)) {
                mergedList.add(listOne.get(i));
                i++;
            } else {
                mergedList.add(listTwo.get(j));
                j++;
            }
        }
        // Add remaining elements of listOne, if any
        while (i < sizeOne) {
            mergedList.add(listOne.get(i));
            i++;
        }
        // Add remaining elements of listTwo, if any
        while (j < sizeTwo) {
            mergedList.add(listTwo.get(j));
            j++;
        }
        return mergedList;
    }
    public static void main(String args[]) {
        List<Integer> listOne = new LinkedList<Integer>();
        listOne.add(10);
        listOne.add(13);
        listOne.add(21);
        listOne.add(42);
        listOne.add(56);
        List<Integer> listTwo = new LinkedList<Integer>();
        listTwo.add(15);
        listTwo.add(20);
        listTwo.add(21);
```

```
        listTwo.add(85);
        listTwo.add(92);

        List<Integer> mergedList = mergeLists(listOne, listTwo);
        System.out.println("Merged List: " + mergedList);
    }
}
```

LinkedList - Assignment 4

```
class Tester {
    public static void main(String args[]) {
        Queue queue = new Queue(5);
        queue.enqueue("Emily");
        queue.enqueue("Lily");
        queue.enqueue("Rachel");
        queue.enqueue("Rose");
        System.out.println("Queue after enqueue operations:");
        System.out.println(queue.getQueue());
        queue.dequeue();
        queue.dequeue();
        System.out.println("Queue after dequeue operations:");
        System.out.println(queue.getQueue());
    }
}
```

Set Interface - Assignment 1

class Student {

```
    private int studentId;
    private String studentName;
    private String emailId;
    private String event;
```

```
public Student(int studentId, String studentName, String emailId, String event) {
    this.studentId = studentId;
    this.studentName = studentName;
    this.emailId = emailId;
    this.event = event;
}
public int getStudentId() {
    return studentId;
}
public void setStudentId(int studentId) {
    this.studentId = studentId;
}
public String getStudentName() {
    return studentName;
}
public void setStudentName(String studentName) {
    this.studentName = studentName;
}
public String getEmailId() {
    return emailId;
}
public void setEmailId(String emailId) {
    this.emailId = emailId;
}
public String getEvent() {
    return event;
}
public void setEvent(String event) {
    this.event = event;
}
@Override
public boolean equals(Object student) {
    if (this == student) return true;
    if (student == null || getClass() != student.getClass()) return false;
```

```
        Student otherStudent = (Student) student;
        return emailId.equals(otherStudent.emailId);
    }
    @Override
    public int hashCode() {
        return emailId.hashCode();
    }
    @Override
    public String toString() {
        return "Student Id: " + studentId + ", Student Name: " + studentName + ", Email Id: " +
emailId;
    }
}
class Tester {

    public static Set<Student> findUnique(List<Student> students) {
        Set<Student> uniqueStudents = new HashSet<>();
        Set<String> seenEmailIds = new HashSet<>();
        for (Student student : students) {
            if (!seenEmailIds.contains(student.getEmailId())) {
                uniqueStudents.add(student);
                seenEmailIds.add(student.getEmailId());
            }
        }
        return uniqueStudents;
    }
    public static Set<Student> findDuplicates(List<Student> students) {
        Set<Student> duplicateStudents = new HashSet<>();
        Set<String> danceStudents = new HashSet<>();
        Set<String> musicStudents = new HashSet<>();
        for (Student student : students) {
            if (student.getEvent().equals("Dance")) {
                if (musicStudents.contains(student.getEmailId())) {
                    duplicateStudents.add(student);
                }
                danceStudents.add(student.getEmailId());
            } else if (student.getEvent().equals("Music")) {
                if (danceStudents.contains(student.getEmailId())) {
                    duplicateStudents.add(student);
                }
                musicStudents.add(student.getEmailId());
            }
        }
        return duplicateStudents;
    }
    public static void main(String[] args) {
        List<Student> students = new ArrayList<>();
        students.add(new Student(5004, "Wyatt", "Wyatt@example.com", "Dance"));
        students.add(new Student(5010, "Lucy", "Lucy@example.com", "Dance"));
        students.add(new Student(5550, "Aaron", "Aaron@example.com", "Dance"));
        students.add(new Student(5560, "Ruby", "Ruby@example.com", "Dance"));
        students.add(new Student(5015, "Sophie", "Sophie@example.com", "Music"));
        students.add(new Student(5013, "Clara", "Clara@example.com", "Music"));
        students.add(new Student(5010, "Lucy", "Lucy@example.com", "Music"));
        students.add(new Student(5011, "Ivan", "Ivan@example.com", "Music"));
        students.add(new Student(5550, "Aaron", "Aaron@example.com", "Music"));
        Set<Student> studentNominations = findUnique(students);
        System.out.println("Students who have submitted nominations");
        for (Student student : studentNominations)
            System.out.println(student);
        Set<Student> duplicateStudents = findDuplicates(students);
        System.out.println("\nStudents who have submitted nominations for both the events");
        for (Student student : duplicateStudents)
            System.out.println(student);
    }
}
```

HashMap – Assignment-1

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
class Tester {
    public static List<String> sortSales(Map<String, Integer> sales) {
        // Step 1: Convert Map entries to a List
        List<Entry<String, Integer>> entryList = new ArrayList<>(sales.entrySet());
        // Step 2: Sort the list based on sales (values) in descending order
        Collections.sort(entryList, (entry1, entry2) ->
entry2.getValue().compareTo(entry1.getValue()));
        // Step 3: Extract names from sorted entries
        List<String> sortedNames = new ArrayList<>();
        for (Entry<String, Integer> entry : entryList) {
            sortedNames.add(entry.getKey());
        }
        // Step 4: Return sorted list of names
        return sortedNames;
    }
    public static void main(String args[]) {
        Map<String, Integer> sales = new HashMap<>();
        sales.put("Mathew", 50);
        sales.put("Lisa", 76);
        sales.put("Courtney", 45);
        sales.put("David", 49);
        sales.put("Paul", 49);
        List<String> employees = sortSales(sales);
        System.out.println("Employees in the decreasing order of their
sales\n========================================");
        for (String employeeName : employees) {
            System.out.println(employeeName);
        }
    }
}
```

HashMap - Assignment 2

```
import java.util.HashMap;
import java.util.Map;
class Tester {
   public static Map<String, Integer> mergeMaps(Map<String, Integer> mapOne, Map<String,
Integer> mapTwo){
      Map<String, Integer> mergedMap = new HashMap<>();
      // Step 1: Merge entries from mapOne
      for (Map.Entry<String, Integer> entry : mapOne.entrySet()) {
         mergedMap.put(entry.getKey(), entry.getValue());
      }
      // Step 2: Merge entries from mapTwo
      for (Map.Entry<String, Integer> entry : mapTwo.entrySet()) {
         String key = entry.getKey();
         Integer value = entry.getValue();
         // Check if key already exists in mergedMap
         if (mergedMap.containsKey(key)) {
            // If key exists, create a new key with "new" appended
            mergedMap.put(key + "new", value);
         } else {
            // If key does not exist, add it directly
            mergedMap.put(key, value);
         }
      }
      return mergedMap;
   }
   public static void main(String args[]) {
      Map<String, Integer> mapOne = new HashMap<>();
      mapOne.put("Kelly", 10);
      mapOne.put("Micheal", 20);
```

```
        mapOne.put("Ryan", 30);
        Map<String, Integer> mapTwo = new HashMap<>();
        mapTwo.put("Jim", 15);
        mapTwo.put("Andy", 45);
        Map<String, Integer> mergedMap = mergeMaps(mapOne, mapTwo);

        System.out.println("Merged Map\n===========");
        for (Map.Entry<String, Integer> entry : mergedMap.entrySet()) {
            System.out.println(entry.getKey() + " -- " + entry.getValue());
        }
    }
}
```

HashMap – Assignment-3

```
import java.util.HashMap;
import java.util.Map;
class Tester {
    public static Map<Character, Integer> findOccurrences(String input) {
        Map<Character, Integer> occurrenceMap = new HashMap<>();
        // Traverse each character in the input string
        for (char c : input.toCharArray()) {
            // Check if character already exists in map
            if (occurrenceMap.containsKey(c)) {
                // If exists, increment count
                occurrenceMap.put(c, occurrenceMap.get(c) + 1);
            } else {
                // If not exists, add with count 1
                occurrenceMap.put(c, 1);
            }
        }
        return occurrenceMap;
    }
    public static void main(String args[]) {
        String input = "occurrence";
        Map<Character, Integer> occurrenceMap = findOccurrences(input);

        System.out.println("Occurrences of characters\n======================");
        for (Map.Entry<Character, Integer> entry : occurrenceMap.entrySet()) {
            System.out.println(entry.getKey() + " -- " + entry.getValue());
        }
    }
}
```

Queue using ArrayDeque – Tryout

```
import java.util.Deque;
import java.util.ArrayDeque;
class Tester{
            public static void main(String[] args) {

Deque<String> queue = new ArrayDeque<String>();            // no restrictions in capacity
                        queue.add("Joe");
                        queue.add("Jack");
                        queue.add("Eva");
                        queue.add("Mia");
                        queue.add("Luke");

                        System.out.println("People in queue - After addition of 5 people");
                        for (String str : queue) {
                                    System.out.println(str);
                        }
                        queue.remove();
                        queue.remove();
                        queue.remove();
                        System.out.println("\nPeople in queue - After removal of 3 people");
                        for (String str : queue) {
```

```
                        System.out.println(str);
                }
                System.out.println();
                System.out.println("Head of the queue using element() - "+queue.element());
                System.out.println("Head of the queue using peek() - "+queue.peek());
                queue.remove();
                queue.remove();
                // new person added to the empty queue using offer()
                queue.offer("Emma");
                // newly added person removed using poll()
                queue.poll();
                System.out.println();
System.out.println("Removing the head of the queue using poll when queue is empty - 
"+queue.poll());
// returns null since queue is empty

                System.out.println("Head of the queue using peek() when queue is empty - 
"+queue.peek());
// returns null since queue is empty
 /* Uncomment the lines of code given below one at a time and observe the output *
        //System.out.println("Head of the queue using element() when queue is empty - 
"+queue.element());    // throws NoSuchElementException since queue is empty

        //System.out.println("Removing the head of the queue using remove() when queue is 
empty");
        //queue.remove();                       // throws NoSuchElementException since queue is empty

            }
}
```

Stack using ArrayDeque - Tryout

```
import java.util.Deque;
import java.util.ArrayDeque;
class Tester {
        public static void main(String[] args) {
```

```
Deque<Integer> stack = new ArrayDeque<Integer>();         // no restrictions in capacity
                    stack.push(1);
                    stack.push(2);
                    stack.push(3);
                    stack.push(4);
                    stack.push(5);

                    System.out.println("Numbers in stack - After addition of 5 values");
                    for (Integer val : stack) {
                              System.out.println(val);

                    stack.pop();
                    stack.pop();
                    stack.pop();
                    System.out.println("\nNumbers in stack - After removal of 3 values");
                    for (Integer val : stack) {
                              System.out.println(val);
                    }
                    System.out.println();
                    System.out.println("Top of the stack using peek() - "+stack.peek());
                    stack.pop();
                    stack.pop();
//Uncomment the below code and observe the output
      //System.out.println("Trying to remove the element from the top of the stack using pop()
when stack is empty - "+stack.pop());           // throws NoSuchElementException since stack is
empty
          }
}
```

Queue Interface - Exercise 1

```
import java.util.Deque;
import java.util.ArrayDeque;
class Tester {
    public static Deque<Object> mergeQueue(Deque<Integer> intQueue, Deque<Character>
charQueue) {
        Deque<Object> mergedQueue = new ArrayDeque<>();
        // Continue until both queues are empty
        while (!intQueue.isEmpty() || !charQueue.isEmpty()) {
            // If integer queue is not empty, add its element
            if (!intQueue.isEmpty()) {
                mergedQueue.add(intQueue.poll());
            }
            // If character queue is not empty, add its element
            if (!charQueue.isEmpty()) {
                mergedQueue.add(charQueue.poll());
            }
        }
        return mergedQueue;
    }
    public static void main(String[] args) {
        Deque<Integer> integerQueue = new ArrayDeque<>();
        integerQueue.add(3);
        integerQueue.add(6);
        integerQueue.add(9);
        Deque<Character> characterQueue = new ArrayDeque<>();
        characterQueue.add('a');
        characterQueue.add('e');
        characterQueue.add('i');
        characterQueue.add('o');
        characterQueue.add('u');
        characterQueue.add('b');
        Deque<Object> mergedQueue = mergeQueue(integerQueue, characterQueue);
        System.out.println("The elements in the merged queue are:");
        for (Object element : mergedQueue) {
            System.out.println(element);
        }
    }
}
```

Queue Interface - Exercise

```
import java.util.Deque;
import java.util.ArrayDeque;
class Tester {
    public static Deque<Integer> changeSmallest(Deque<Integer> inputStack) {
        if (inputStack == null || inputStack.isEmpty()) {
            return inputStack;
        }
        // Find the minimum value
        int minValue = Integer.MAX_VALUE;
        Deque<Integer> tempStack = new ArrayDeque<>();
        while (!inputStack.isEmpty()) {
            int value = inputStack.pop();
            if (value < minValue) {
                minValue = value;
            }
            tempStack.push(value);
        }
        // Count occurrences of the minimum value
        int minCount = 0;
        Deque<Integer> intermediateStack = new ArrayDeque<>();
        while (!tempStack.isEmpty()) {
            int value = tempStack.pop();
            if (value == minValue) {
                minCount++;
            } else {
                intermediateStack.push(value);
            }
        }
        // Rebuild the input stack
        while (!intermediateStack.isEmpty()) {
            inputStack.push(intermediateStack.pop());
        }
        for (int i = 0; i < minCount; i++) {
            inputStack.push(minValue);
        }
        return inputStack;
    }
    public static void main(String[] args) {
        Deque<Integer> inputStack = new ArrayDeque<>();
        inputStack.push(10);
        inputStack.push(8);
        inputStack.push(5);
        inputStack.push(12);
        inputStack.push(5);
        Deque<Integer> updatedStack = changeSmallest(inputStack);
        System.out.println("Stack After Modification:");
        for (Integer value : updatedStack) {
            System.out.println(value);
        }
    }
}
```