Stack - Assignment 1

Problem Statement

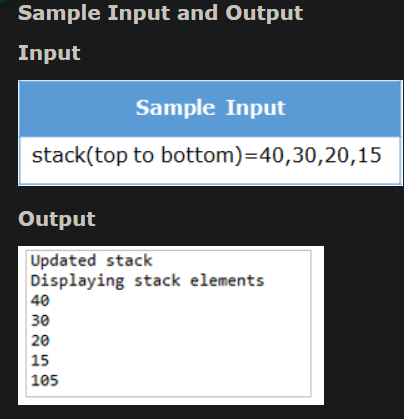
Given a stack of integers, calculate the sum of all the integers present in the stack. Modify the stack such that the sum is present in the bottom of the stack and all the other integers are present in the stack in the same order. Implement the logic inside calculateSum() method of the Tester class.

The Stack class has been provided.

Test the functionalities using the main() method of the Tester class.

Sample Input and Output

Input



public class Tester {  
 public static void main(String args[]) {  
  
 Stack stack = new Stack(10);  
 stack.push(15);  
 stack.push(20);  
 stack.push(30);  
 stack.push(40);  
  
 *calculateSum*(stack);  
  
 System.*out*.println("Updated stack");  
 stack.display();  
 }  
  
 public static void calculateSum(Stack stack) {  
 //Implement your code here  
 Stack tempStack = new Stack(12);  
 int a = 0;  
 while (!stack.isEmpty()) {  
 a += stack.peek();  
 tempStack.push(stack.pop());  
 }  
 stack.push(a);  
 while (!tempStack.isEmpty()) {  
  
 stack.push(tempStack.pop());  
 }  
  
 }  
}

—————————————————————————————————————————————————

Queue - Assignment 1

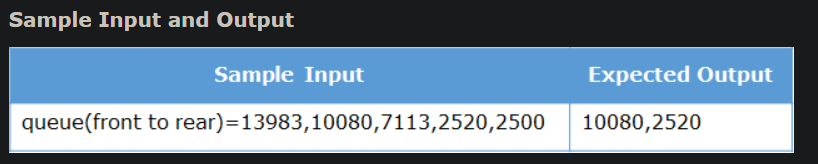
Problem Statement

Given a queue of whole numbers, find the numbers in the queue that are evenly divisible by all the numbers from 1 to 10 and return a new queue with all the evenly divisible numbers. Implement the logic inside findEvenlyDivisibleNumbers() method of the Tester class.

The Queue class has been provided.

Test the functionalities using the main() method of the Tester class.

Sample Input and Output



class Tester {  
  
 public static void main(String[] args) {  
  
 Queue queue = new Queue(7);  
 queue.enqueue(13983);  
 queue.enqueue(10080);  
 queue.enqueue(7113);  
 queue.enqueue(2520);  
 queue.enqueue(2500);  
  
 Queue outputQueue = *findEvenlyDivisibleNumbers*(queue);  
  
 System.*out*.println("Evenly divisible numbers");  
 outputQueue.display();  
  
 }  
  
 public static Queue findEvenlyDivisibleNumbers(Queue queue) {  
 // Create a result queue to hold numbers divisible by 1 to 10  
 Queue resultQueue = new Queue(queue.getMaxSize());  
  
 // Create a temporary queue to restore original contents if needed  
 Queue tempQueue = new Queue(queue.getMaxSize());  
  
 // Dequeue each element, check divisibility, and enqueue to result if valid  
 while(!queue.isEmpty()) {  
 int current = queue.dequeue();  
 if(*isDivisibleBy1To10*(current)) {  
 resultQueue.enqueue(current);  
 }  
  
 }  
  
 // Return the queue containing numbers divisible by 1..10  
 return resultQueue;  
 }  
  
 // Helper method to check if a number is divisible by all integers 1..10  
 public static boolean isDivisibleBy1To10(int number) {  
 for(int i = 1; i <= 10; i++) {  
 if(number % i != 0) {  
 return false;  
 }  
 }  
 return true;  
 }  
}

—————————————————————————————————————————————————

#### **Priority Queue for Employee Ranking**

Write a Java program that accepts a non-empty inStrQueue (String Queue) with each element in the format "employeeName:rank". The program should return an outStrQueue (Front → Rear) based on the following logic:

* Arrange employees in ascending order of rank.
* If two employees have the same rank, arrange them lexicographically by employeeName.
* Add all elements to outStrQueue in the sorted order.
* Return outStrQueue.

inStrQueue = ["John:3", "Alice:2", "Bob:2", "Eve:1"]

outStrQueue = ["Eve:1", "Alice:2", "Bob:2", "John:3"]

—————————————————————————————————————————————————

#### **Stack Number Manipulation Based on Digit Sum**

Write a Java program that accepts a non-empty inIntStack (Bottom → Top) as input and returns an outIntStack (Top → Bottom) based on the following logic:

* For each element in inIntStack, calculate the sum of its digits.
* If the digit sum is odd, add the element to tempDataStructure1.
* If the digit sum is even, add the element to tempDataStructure2.
* After processing all elements:
* Push all elements from tempDataStructure1 to outIntStack in **reverse order**.
* Then, push all elements from tempDataStructure2 in the **same order** as they appeared in inIntStack.
* Return outIntStack.

##### **Example:**

inIntStack = [34, 56, 12, 29, 81]

outIntStack = [81, 29, 12, 56, 34]

—————————————————————————————————————————————————Write a Java program that accepts a non-empty stack inStrStack (Top → Bottom) and returns an output queue outStrQueue (Front → Rear) based on the following logic:

### **Input:**

The stack inStrStack contains strings in the format Item-<Category> where:

* <Category> is either A, B, or C.
* **Category A**: Represents high-priority items.
* **Category B**: Represents medium-priority items.
* **Category C**: Represents low-priority items.

### **Processing Logic:**

For each element in inStrStack (Top → Bottom):

1. If the category is **A**, move the element to tempDataStructure1 (High-Priority List).
2. If the category is **B**, move the element to tempDataStructure2 (Medium-Priority List).
3. If the category is **C**, move the element to tempDataStructure3 (Low-Priority List).

### **Final Queue Creation:**

1. First, add all elements from tempDataStructure1 (High-Priority) to outStrQueue in the order they appeared in inStrStack.
2. Then, add all elements from tempDataStructure2 (Medium-Priority) to outStrQueue.
3. Finally, add all elements from tempDataStructure3 (Low-Priority) to outStrQueue.

Return the outStrQueue, which contains the elements sorted by their priority (from high to low).