**Question**

A class `Deque` is designed to implement a double-ended queue (deque) in Java. The deque supports operations to insert and delete elements from both the front and rear ends. Some of the data members/member functions are given below:

Class name: `Deque`

Data members/instance variables:

- `arr[]`: An integer array to store the elements of the deque.

- `front`: An integer to keep track of the front index of the deque.

- `rear`: An integer to keep track of the rear index of the deque.

- `size`: An integer representing the maximum size of the deque.

Member functions/methods:

- `Deque(int size)`: Constructor to initialize the deque with a given size.

- `boolean isFull()`: Checks if the deque is full.

- `boolean isEmpty()`: Checks if the deque is empty.

- `void insertfront(int key)`: Inserts an element at the front of the deque.

- `void insertrear(int key)`: Inserts an element at the rear of the deque.

- `void deletefront()`: Deletes an element from the front of the deque.

- `void deleterear()`: Deletes an element from the rear of the deque.

- `int getFront()`: Returns the element at the front of the deque without removing it.

- `int getRear()`: Returns the element at the rear of the deque without removing it.

Specify the class `Deque` giving details of the constructor, `boolean isFull()`, `boolean isEmpty()`, `void insertfront(int key)`, `void insertrear(int key)`, `void deletefront()`, `void deleterear()`, `int getFront()`, and `int getRear()`. Define the `main()` function to create an object and call the functions accordingly to demonstrate the functionality of the deque.

**Algorithm**

1. Start

2. Define a class Deque with the following instance variables:

- `MAX`: a constant integer representing the maximum size of the deque.

- `arr[]`: an integer array to store the elements of the deque.

- `front`: an integer to keep track of the front index of the deque.

- `rear`: an integer to keep track of the rear index of the deque.

- `size`: an integer representing the maximum size of the deque.

3. Define a parameterized constructor for the class Deque:

- Accept an integer `size` as a parameter.

- Initialize `arr` with a new integer array of size `MAX`.

- Initialize `front` with -1.

- Initialize `rear` with 0.

- Initialize `this.size` with the provided `size`.

4. Define a method `isFull()` to check if the deque is full:

- Return `true` if either `front == 0 && rear == size - 1` or `front == rear + 1`.

5. Define a method `isEmpty()` to check if the deque is empty:

- Return `true` if `front == -1`.

6. Define a method `insertfront(int key)` to insert an element at the front of the deque:

- If the deque is full, print "Overflow" and return.

- If the deque is empty, set `front` and `rear` to 0.

- If `front` is 0, set `front` to `size - 1`.

- Otherwise, decrement `front` by 1.

- Set `arr[front]` to `key`.

7. Define a method `insertrear(int key)` to insert an element at the rear of the deque:

- If the deque is full, print "Overflow" and return.

- If the deque is empty, set `front` and `rear` to 0.

- If `rear` is `size - 1`, set `rear` to 0.

- Otherwise, increment `rear` by 1.

- Set `arr[rear]` to `key`.

8. Define a method `deletefront()` to delete an element from the front of the deque:

- If the deque is empty, print "Queue Underflow" and return.

- If `front` is equal to `rear`, set both to -1 (deque becomes empty).

- If `front` is `size - 1`, set `front` to 0.

- Otherwise, increment `front` by 1.

9. Define a method `deleterear()` to delete an element from the rear of the deque:

- If the deque is empty, print "Underflow" and return.

- If `front` is equal to `rear`, set both to -1 (deque becomes empty).

- If `rear` is 0, set `rear` to `size - 1`.

- Otherwise, decrement `rear` by 1.

10. Define a method `getFront()` to return the element at the front of the deque without removing it:

- If the deque is empty, print "Underflow" and return -1.

- Return `arr[front]`.

11. Define a method `getRear()` to return the element at the rear of the deque without removing it:

- If the deque is empty or `rear` is less than 0, print "Underflow" and return -1.

- Return `arr[rear]`.

12. In the main method:

- Create an object of the class Deque with a size of 4.

- Demonstrate the functionality of the deque by inserting elements at the rear, deleting elements from the rear, inserting elements at the front, and deleting elements from the front.

- Print the front and rear elements after each operation.

13. End

**Variable Description**

|  |  |  |
| --- | --- | --- |
| **Variable** | **Data type** | **Purpose** |
| MAX | int | Maximum size of array |
| arr | int[] | To store elements of deque |
| front | int | Index to keep track of front element |
| rear | int | Index to keep track of rear element |
| size | int | Maximum size of deque |
| key | int | Integer value to be inserted into queue |