Queue

Read: Chapter 7 of the text book.

- Stack: Last-in-first-out
- Queue: First-in-first-out; Enter the queue at the rear and remove from the front

1 Applications to use Queue data structure

- When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling.
- When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.

2 Queue operations

- enqueue(e): Insert element e at the rear of the queue
- dequeue(): Remove the element at the front of the queue;
- front(): Return a reference to the front element in the queue; Do not remove this element; an error occurs if the queue is empty
- size(): Return the number of elements in the queue
- isEmpty(): Return true if the queue is empty and false otherwise

3 Implementing a queue

- `MyQueue` interface

```
public interface MyQueue<E> {
    public void enqueue(E e);
    public E dequeue();
    public E front();
    public int size();
    public boolean isEmpty();
}
```

- Implementation: `LinkQueue<E>`
- Implementation: `ArrayQueue<E>`

4 Using a queue

4.1 Basic usage

```
queue.enqueue(10);

for(int i=0;i<5;i++){
    queue.enqueue(i+1);
    queue.printQueue();
}

while(!queue.isEmpty()){
    System.out.println("dequeue="+queue.dequeue());
    queue.printQueue();
}
```

5 Double-Ended Queue Operations

- insertFront (e): Insert a new element e at the beginning of the deque
- insertBack(e): Insert a new element e at the end of the deque
- eraseFront(): Remove the first element of the deque; an error occurs if the deque is empty
- eraseBack(): Remove the last element of the deque; an error occurs if the deque is empty
- front(): Return the first element of the deque; an error occurs if the deque is empty
- back(): Return the last element of the deque; an error occurs if the deque is empty
- size(): Return the number of elements of the deque
- isEmpty(): Return true if the dequeu is empty and false otherwise

6 Java classes

- `Java.util.Queue` // Generic interface
- `java.util.deque` // Generic interface
  - A linear collection that supports element insertion and removal at both ends. The name *deque* is short for "double ended queue" and is usually pronounced "deck".
- `LinkedList<E>`
  - Implements Deque, List, Queue, and others
  - Can be used as Stack, Queue, or Deque
  - All the operations perform as could be expected for a doubly-linked list
- `ArrayDeque<E>`
  - Implements Deque, Queue, and others
  - Resizable-array implementation

7 Summary

- Queues can be implemented using different data structures (e.g., Array, Linked list)
- Queues have many applications.
- Queue: Typical operations; Array implementation; Linked List implementation
- Deque: Typical operations; Doubly linked list implementation