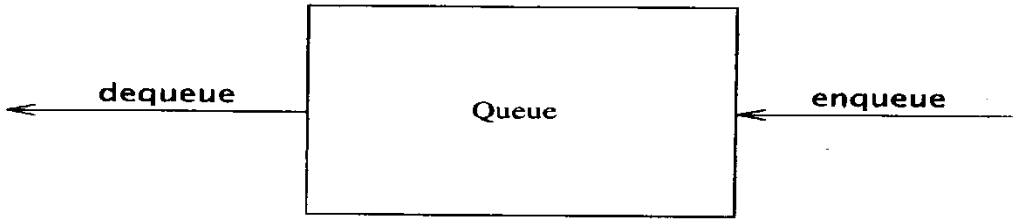
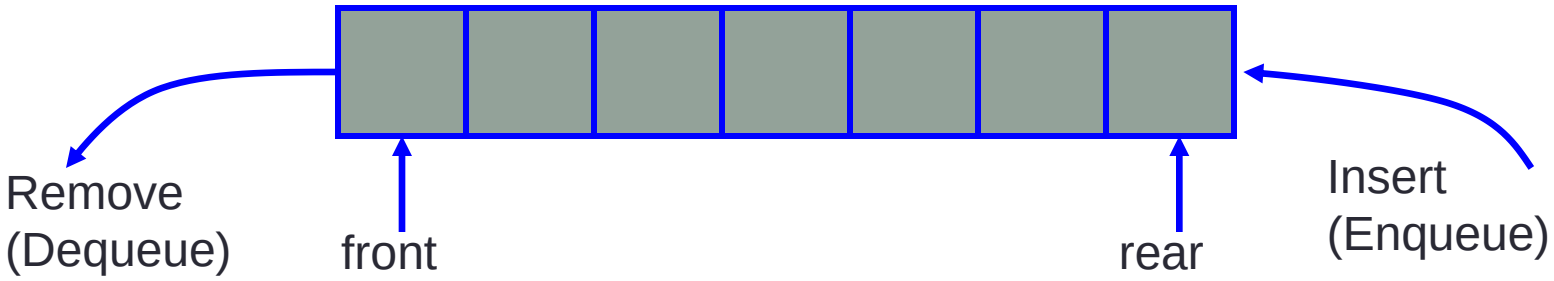
What is Queue?

- Queue is a data structure in which insertion is done at one end (Front), while deletion is performed at the other end (Rear)
  - Contrast with stack, where insertion and deletion at one and the same end
- It is First In, First Out (FIFO) structure
  - For example, customers standing in a check-out line in a store, the first customer in is the first customer served.

Queue operations

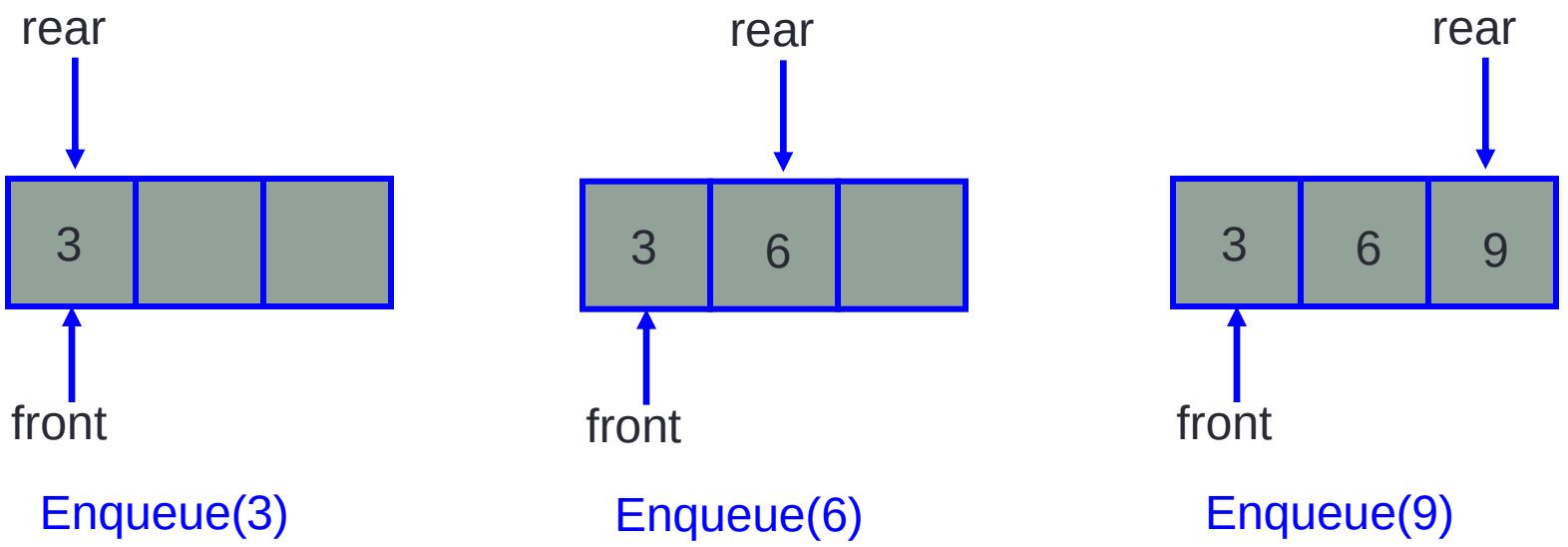
- Enqueue: insert an element at the rear of the list
- Dequeue: delete the element at the front of the list





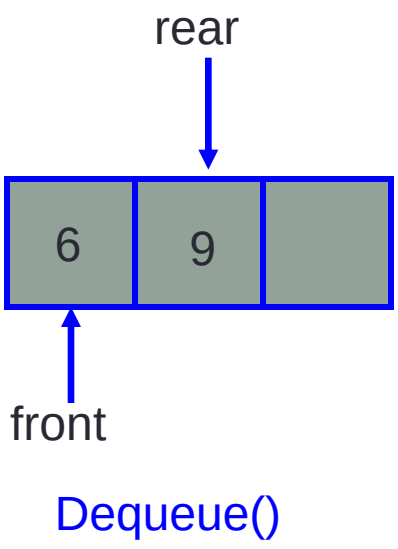
Building a Queue Step-by-Step

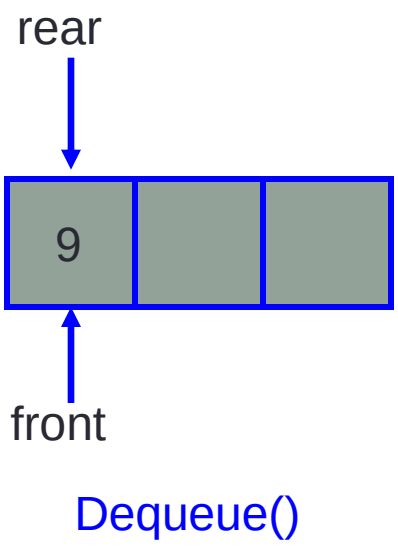
- There are several different algorithms to implement Enqueue and Dequeue
- Enqueuing
  - The front index is always fixed
  - The rear index moves forward in the array

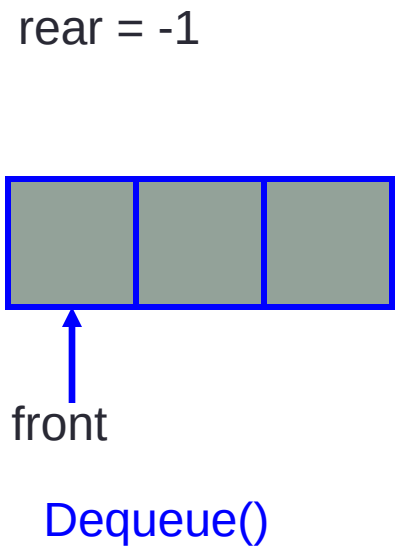


Building a Queue Step-by-Step

- Dequeuing
  - The element at the front of the queue is removed
  - Move all the elements after it by one position







The Queue Abstract Data Type

- Formally, the queue abstract data type defines a collection that keeps objects in a sequence, where
  - element access and deletion are restricted to the first element in the sequence, the ***front*** of the queue,
  - and element insertion is restricted to the end of the sequence, the ***rear*** of the queue.
  - This restriction enforces the rule that items are inserted and deleted in a queue according to the first-in first-out (FIFO) principle.
- The ***queue*** abstract data type (ADT) supports the following two fundamental methods:
  - enqueue(*e*): Insert element *e* at the rear of the queue.
  - dequeue(): Remove and return from the queue the object at the front;
    - an error occurs if the queue is empty.
- Additionally, similar to the case with the Stack ADT, the queue ADT includes the following supporting methods:
  - size(): Return the number of objects in the queue.
  - isEmpty(): Return a Boolean value that indicates whether the queue is empty.
- front(): Return, but do not remove, the front object in the queue;
    - an error occurs if the queue is empty.

The Queue Abstract Data Type

- Example : The following table shows a series of queue operations and their effects on an initially empty queue Q of integer objects. For simplicity, we use integers instead of integer objects as arguments of the operations.

Operation	Output	front ← Q ← rear
enqueue(5)	-	(5)
enqueue(3)	-	(5, 3)
dequeue( )	5	(3)
enqueue(7)	-	(3, 7)
dequeue( )	3	(7)
front( )	7	(7)
dequeue( )	7	( )
dequeue( )	"error"	( )
isEmpty( )	true	( )
enqueue(9)	-	(9)
enqueue(7)	-	(9, 7)
size()	2	(9, 7)
enqueue(3)	-	(9, 7, 3)
enqueue(5)	-	(9, 7, 3, 5)
dequeue( )	9	(7, 3, 5)

The Queue Abstract Data Type

- Example Applications
- There are several possible applications for queues.
  - Stores,
  - theaters,
  - reservation centers,
  - and other similar services typically process customer requests according to the FIFO principle.
- A queue would therefore be a logical choice for a data structure to handle transaction processing for such applications.
- For example, it would be a natural choice for handling calls to the reservation center of an airline or to the box office of a theater.

Implementing a Queue with a Generic Linked List

- We can efficiently implement the queue ADT using a generic singly linked list.
- For efficiency reasons,
  - the front of the queue to be at the head of the list,
  - and the rear of the queue to be at the tail of the list.
  - In this way, we remove from the head and insert at the tail. Note that we need to maintain references to both the head and tail nodes of the list. Rather than go into every detail of this implementation, we simply give a Java implementation for the fundamental queue methods in Code Fragment 7.7

Code Fragment 7.7: Methods enqueue and dequeue in the implementation of the queue ADT by means of a singly linked list, using nodes from class Node of Code Fragment 7.3

```
public void enqueue(E elem) {
  Node<E> node = new Node<E>();
  node.setElement(elem);
  node.setNext(null); // node will be new tail node
  if (size == 0)
    head = node; // special case of a previously empty queue
  else
    tail.setNext(node); // add node at the tail of the list
  tail = node; // update the reference to the tail node
  size++;
}
...
public E dequeue() throws EmptyQueueException {
  if (size == 0)
    throw new EmptyQueueException("Queue is empty.");
  E tmp = head.getElement();
  head = head.getNext();
  size--;
  if (size == 0)
    tail = null; // the queue is now empty
  return tmp;
}
```

End