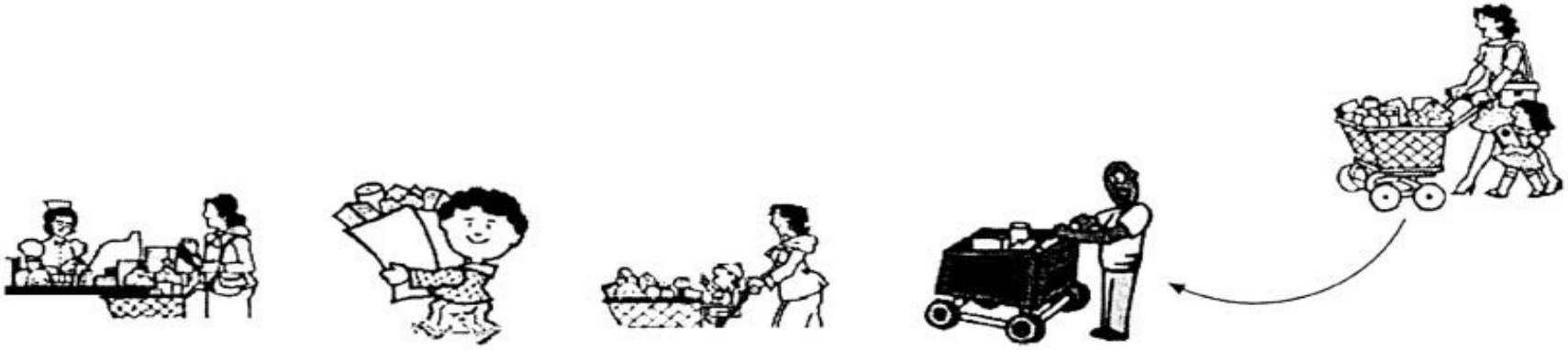
Queue & Priority Queue

WIA1002/ WIB1002 :

Data Structures

Queue

- A waiting line at a grocery store or a bank is a model of a queue.

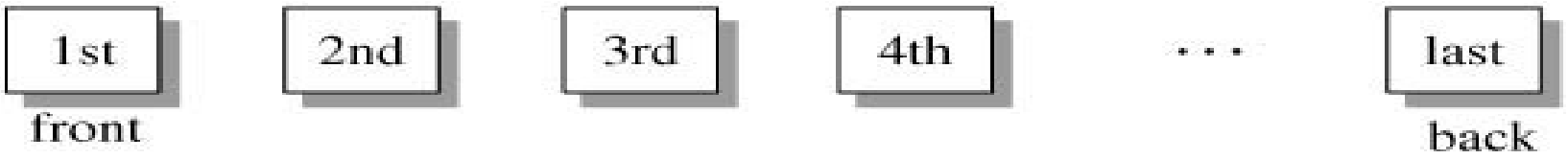


- Which is first in this queue?
- Which is last?

Queues

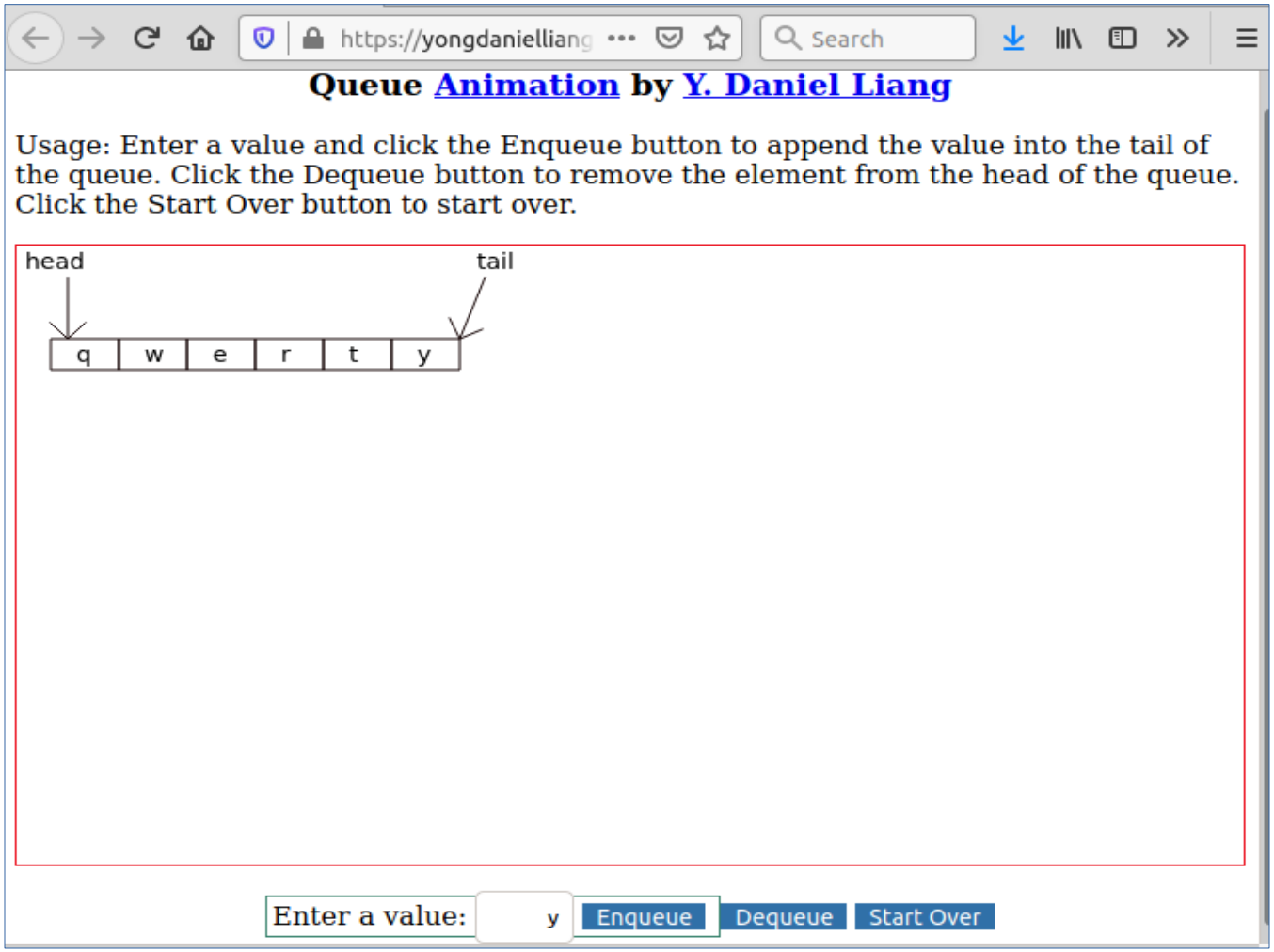
- A queue represents a waiting list.

- A queue can be viewed as a special type of **list**, where the elements are inserted into the end (back/tail) of the queue, and are accessed and deleted from the beginning (front/head) of the queue.



- An item removed from the queue is the first element that was added into the queue. A queue has FIFO (first-in-first-out) ordering.

Queue Animation



https://yongdanielliang.github.io/animation/web/Queue.html

Check Point

1. Where is element inserted and deleted from a queue?

2. What is this ordering called?

Check Point

1. Where is element inserted and deleted from a queue?

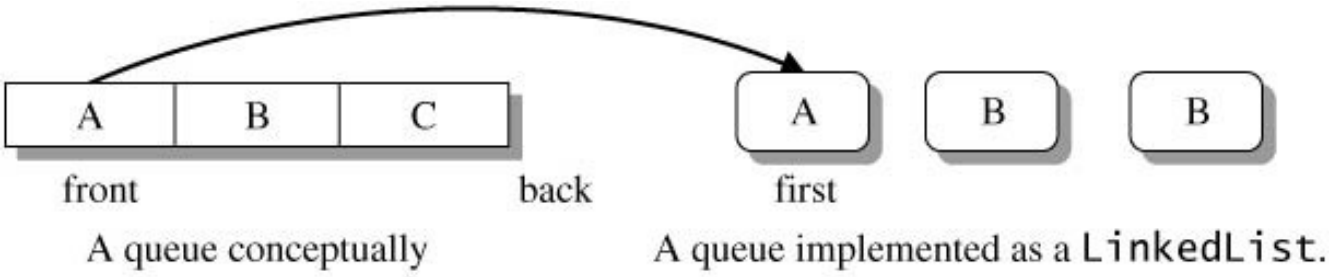
   The elements are inserted into the end (back/tail) of the queue, and are accessed and deleted from the beginning (front/head) of the queue

2. What is this ordering called?

   FIFO (first-in-first-out) ordering.

Implementing Queue

- Since deletions are made at the beginning of the list, it is more efficient to implement a queue using a linked list than an array list.
- Using **LinkedList** to implement **Queue**



Methods/Operations in Queue

- enqueue() ✉ en(in) / add
- dequeue() ✉ delete

Implementing Queue

- two ways to design the queue class using LinkedList:
  - Using inheritance: You can define a queue (GenericQueue) class by extending LinkedList class
  - Using composition: You can define an linked list as a data field in the queue(GenericQueue) class

(a) Using inheritance

(b) Using composition

GenericQueue<E> Class

GenericStack class using a LinkedList & composition approach.

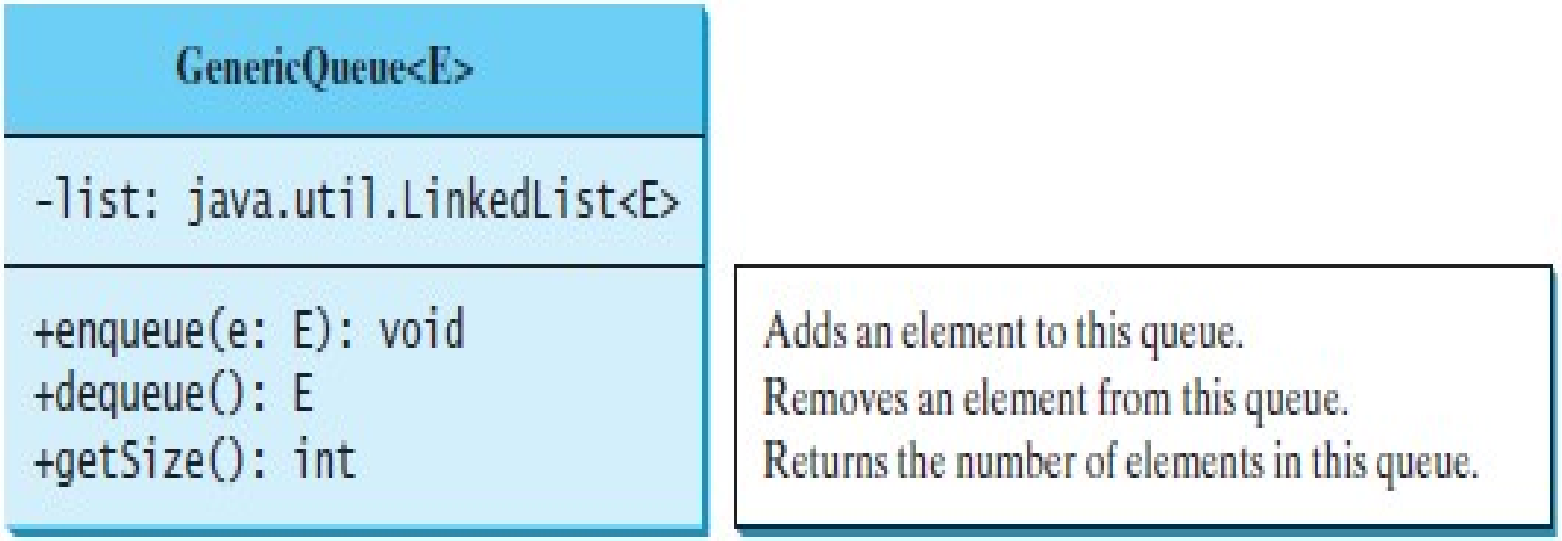


FIGURE 24.22 GenericQueue uses a linked list to provide a first-in, first-out data structure.

GenericQueue<E>

LISTING 24.7 GenericQueue.java

```
1  public class GenericQueue<E> {
2    private java.util.LinkedList<E> list
3      = new java.util.LinkedList<>();
4
5    public void enqueue(E e) {
6      list.addLast(e);
7    }
8
9    public E dequeue() {
10     return list.removeFirst();
11   }
12
13   public int getSize() {
14     return list.size();
15   }
16
17   @Override
18   public String toString() {
19     return "Queue: " + list.toString();
20   }
21 }
```

https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html

Test GenericQueue class

```
23        // Create a queue
24        GenericQueue<String> queue = new GenericQueue<>();
25
26        // Add elements to the queue
27        queue.enqueue("Tom"); // Add it to the queue
28        System.out.println("(7) " + queue);
29
30        queue.enqueue("Susan"); // Add it to the queue
31        System.out.println("(8) " + queue);
32
33        queue.enqueue("Kim"); // Add it to the queue
34        queue.enqueue("Michael"); // Add it to the queue
35        System.out.println("(9) " + queue);
36
37        // Remove elements from the queue
38        System.out.println("(10) " + queue.dequeue());
39        System.out.println("(11) " + queue.dequeue());
40        System.out.println("(12) " + queue);
41    }
42  }
```

```
(7) Queue: [Tom]
(8) Queue: [Tom, Susan]
(9) Queue: [Tom, Susan, Kim, Michael]
(10) Tom
(11) Susan
(12) Queue: [Kim, Michael]
```

Check Point

1. Which type of data structure is best to use to implement Queue?
2. What are the 2 important operations for Queue?

Check Point

1. Which type of data structure is best used to implement Queue?

   Linked list

2. What are the 2 important operations for Queue?

   enqueue & dequeue

Priority Queue

A **regular queue** is a **first-in and first-out (FIFO)** data structure. Elements are appended to the end of the queue and are removed from the beginning of the queue.

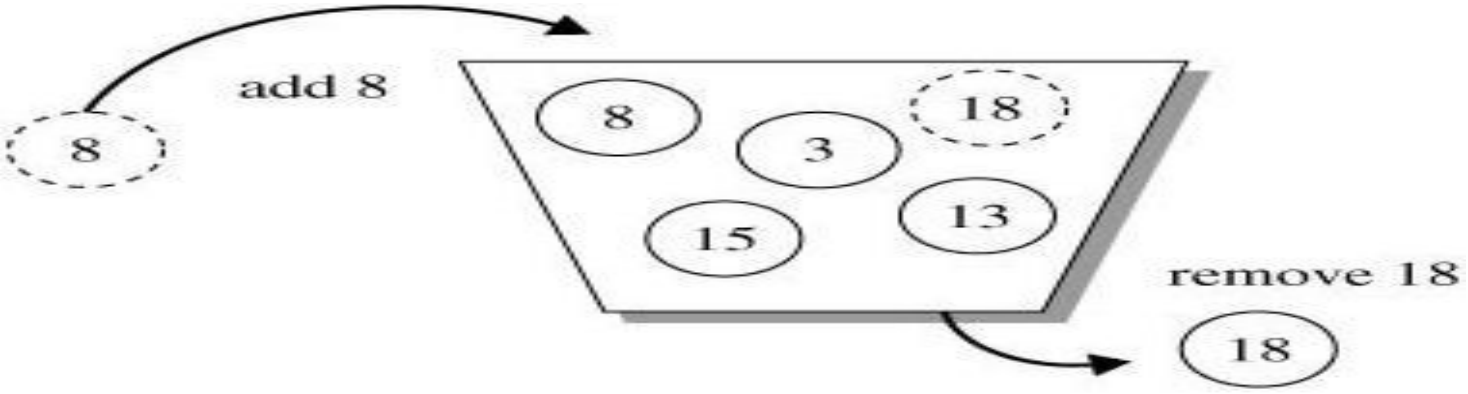
In a **priority queue**, **elements are assigned with priorities**. When accessing elements, the element with the **highest priority is removed first**.

A **priority queue** has a **largest-in, first-out** behavior (however you need to double check whether the larger the higher the priority, or the smaller the higher the priority).

For example, the emergency room in a hospital assigns priority numbers to patients; the patient with the highest priority is treated first. The assumption here the higher the number the higher the priority.

Priority Queue

- A priority queue is a collection in which all elements have a comparison (priority) ordering.
- It provides only simple access and update operations where a deletion always removes the element of highest priority



- In the above diagram, the assumption is the higher the number, the higher the priority.

The PriorityQueue Class from Java API

- import java.util.PriorityQueue
- Implemented interface java.util.Queue<E>

Constructors

Constructor and Description
`PriorityQueue()` Creates a `PriorityQueue` with the default initial capacity (11) that orders its elements according to their **natural ordering**.
`PriorityQueue(Collection<? extends E> c)` Creates a `PriorityQueue` containing the elements in the specified collection.
`PriorityQueue(int initialCapacity)` Creates a `PriorityQueue` with the specified initial capacity that orders its elements according to their **natural ordering**.
`PriorityQueue(int initialCapacity, Comparator<? super E> comparator)` Creates a `PriorityQueue` with the specified initial capacity that orders its elements according to the specified comparator.
`PriorityQueue(PriorityQueue<? extends E> c)` Creates a `PriorityQueue` containing the elements in the specified priority queue.
`PriorityQueue(SortedSet<? extends E> c)` Creates a `PriorityQueue` containing the elements in the specified sorted set.

https://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html

PriorityQueue

- To add an element :
  - offer(E e) [returns boolean]
- To remove an object:
  - remove (Object o) [returns bolean]
- To retrieve, but not remove the head of the PriorityQueue:
  - peek () [returns E]
- To retrieve and remove the head of the PriorityQueue:
  - poll () [returns E]

PriorityQueue

- To clear the PriorityQueue:
  - clear() [void]
- To check the size of the Priority Queue :
  - size(E e) [returns integer]
- To check whether object o is in the PriorityQueue:
  - contains (Object o) [returns bolean]
- PriorityQueue:
  - peek () [returns E]

PriorityQueue Example

```
 1  import java.util.*;
 2
 3  public class PriorityQueueDemo {
 4    public static void main(String[] args) {
 5      PriorityQueue<String> queue1 = new PriorityQueue<>();
 6      queue1.offer("Oklahoma");
 7      queue1.offer("Indiana");
 8      queue1.offer("Georgia");
 9      queue1.offer("Texas");
10      System.out.println("Priority queue using Comparable:");
11
12      while (queue1.size() > 0) {
13        System.out.print(queue1.remove() + " ");
14      }
15
16      PriorityQueue<String> queue2
17        = new PriorityQueue(4, Collections.reverseOrder());
18      queue2.offer("Oklahoma");
19      queue2.offer("Indiana");
20      queue2.offer("Georgia");
21      queue2.offer("Texas");
22      System.out.println("\nPriority queue using Comparator:");
23      while (queue2.size() > 0) {
24          System.out.print(queue2.remove() + " ");
25      }
26      System.out.println();
27    }
28  }
```

```
Priority queue using Comparable:
Georgia Indiana Oklahoma Texas
Priority queue using Comparator:
Texas Oklahoma Indiana Georgia
```

PriorityQueue Example2

```
 1  public class Customer implements Comparable<Customer> {
 2      private Integer id;
 3      private String name;
 4
 5      public Customer(Integer id, String name) {
 6          this.id = id;
 7          this.name = name;
 8      }
 9
10      public Integer getID() {
11          return id;
12      }
13      public void setID(Integer id)   {
14          this.id = id;
15      }
16      public String getName() {
17          return name;
18      }
19      public void setName(String name)    {
20          this.name = name;
21      }
22
23      @Override
24      public int compareTo(Customer c) {
25          return this.getID().compareTo(c.getID());
26      }
27
28      @Override
29      public String toString() {
30          return "Customer [ id=" + id + ", name=" + name + " ]" ;
31      }
32  }
```

Create a class Customer

PriorityQueue Example2

```
 1  import java.util.*;
 2
 3  public class PriorityQueue2 {
 4    public static void main(String[] args) {
 5      PriorityQueue<Customer> customerQueue
 6        = new PriorityQueue<>(Collections.reverseOrder());
 7
 8      customerQueue.add(new Customer(3, "Donald" ));
 9      customerQueue.add(new Customer(1, "Chong" ));
10      customerQueue.add(new Customer(2, "Ali" ));
11      customerQueue.add(new Customer(4, "Bala" ));
12
13      Customer c = customerQueue.peek();
14      if (c!=null) {
15          System.out.println(c.getName() + " is in queue");
16          while ((c = customerQueue.poll())!=null
17              System.out.println(c);
18      }
19      System.out.println();
20    }
21  }
22
```

```
Bala is in the queue
Customer [ id=4, name=Bala ]
Customer [ id=3, name=Donald ]
Customer [ id=2, name=Ali ]
Customer [ id=1, name=Chong ]
```

Exercise

- Implement a generic Queue using array
- Implement a generic Queue using ArrayList
- Hints: you can name the class ArrayQueue<E> and provide the implementation for the following methods:
  - public ArrayQueue()
  - Public ArrayQueue(int initial)
  - public void enqueue(E e)
  - public E dequeue()
  - public E getElement()
  - public boolean isEmpty()
  - public int size()
  - public void resize()

Reference

- Chapter 19 and 24, Liang, Introduction to Java Programming, 10th Edition, Global Edition, Pearson, 2015