menu
- Articles

featured articles
- QuickSort Tutorial
- Linked Lists Tutorial

A tutorial on Queues in Java

The basic concept of queues is one people normally find easy to understand, because we encounter them on a daily basis. When you go to the bank, unless you're rich enough to avoid having to wait in line, you will wait in a queue for the teller. The person at the front of the queue gets served as soon as a teller becomes free. When a new person enters the bank, they go to the back of the line (queue). This means that queues work on a FIFO (first in first out) basis.

There is more than one way of implementing a queue. The most common way is to use **linked lists** (see Linked Lists Tutorial). My implementation uses linked lists.

The purpose of this article is to help you understand how queues are implemented. These days, it makes more sense to use the queues from the Java library, unless you are doing something very custom.

In my opinion, the best way to learn is by going through examples, so I hope you will find the following example helpful.

```
Code being executed is in blue.

--------------------
Queue q = new Queue();
An empty queue is created.

Person alice = new Person("Alice");
Person bob = new Person("Bob");
Two objects of type Person are created.

This does not affect the queue yet.

q.enqueue(alice);
Alice has been added to the queue.

System.out.println(((Person)q.peek()).getName());
"Alice" is outputted. The peek() method returns
the object at the front of the queue without affecting
the queue itself.

q.enqueue(bob);
Bob has been added to the queue.
Bob is not at the front of the queue, and so
q.peek() would still return Alice.

Using the bank example, let's say a teller becomes
free.
q.dequeue();
The dequeue() method will return the 'alice' object
and remove it from the queue. Bob is now the only
one left in the queue.

System.out.println(((Person)q.peek()).getName());
"Bob" is outputted because he is next in the queue.

Fortunately for Bob, another teller becomes free,
so he is dequeued and the queue becomes empty.
q.dequeue();
The dequeue() method will return the 'bob' object
and remove it from the queue.
```

As mentioned above, my implementation of the queue uses linked lists. For consistency, I've chosen to use the linked list code from my Linked Lists Tutorial. **Copy the code from there into a file and name it LinkedList.java.**

This is my Queue.java:

```
public class Queue {
        private LinkedList list;

        // Queue constructor
        public Queue()
        {
                // Create a new LinkedList.
                list = new LinkedList();
        }

        public boolean isEmpty()
        // Post: Returns true if the queue is empty. Otherwise, false.
        {
                return (list.size() == 0);
        }

        public void enqueue(Object item)
        // Post: An item is added to the back of the queue.
        {
                // Append the item to the end of our linked list.
                list.add(item);
        }

        public Object dequeue()
        // Pre: this.isEmpty() == false
        // Post: The item at the front of the queue is returned and
        //         deleted from the queue. Returns null if precondition
        //         not met.
        {
                // Store a reference to the item at the front of the queue
                //   so that it does not get garbage collected when we
                //   remove it from the list.
                // Note: list.get(...) returns null if item not found at
                //   specified index. See postcondition.
                Object item = list.get(1);
                // Remove the item from the list.
                // My implementation of the linked list is based on the
                //   J2SE API reference. In both, elements start at 1,
                //   unlike arrays which start at 0.
                list.remove(1);

                // Return the item
                return item;
        }

        public Object peek()
        // Pre: this.isEmpty() == false
        // Post: The item at the front of the queue is returned and
        //         deleted from the queue. Returns null if precondition
        //         not met.
        {
                // This method is very similar to dequeue().
                // See Queue.dequeue() for comments.
                return list.get(1);
        }
}
```

I will also provide you with my Person class in case you'd like to try following my Alice & Bob example above. Here is my Person.java:

```
public class Person {
        String name;

        // Person constructor
        public Person(String _name)
        // Post: An instance of Person is initialized.
        {
```

```
        {
                // Set the name of this Person object to the name passed into the con
                name = _name;
        }

        public String getName()
        // Post: The name of this person is returned.
        {
                return name;
        }
} // End of Person class
```

Written by Pavel.