BlockingQueue

```
public interface BlockingQueue<T> {
	public void put(T elem);
	public T take() throws InterruptedException;
	public void clear();
	public int remaingCapacity();
	public boolean isEmpty();
}
```

LinkedBlockinQueue

```
public class LinkedBlockinQueue<T> implements BlockingQueue<T> {
	final Object mutex = new Object();
	Node<T> head, tail;
	@Override
	public void put(T elem) {
		if(elem == null) {
			throw new NullPointerException("elem == null");
		}
		Node<T> node = new Node<T>();
		node.value = elem;
		node.next = null;
		synchronized (mutex) {
			if(tail == null) {
				head = tail = node;
			} else {
				tail.next = node;
				tail = node;
			}
			System.out.println("[ADDED]: " + elem.toString());
			if(head.next == null) {
				mutex.notify();
			}
		}
	}
	@Override
	public T take() throws InterruptedException {
		T result;
		synchronized (mutex) {
			while(isEmpty())
				mutex.wait();
			result = head.value;
			head = head.next;
			if(head == null)
				tail = null;
			mutex.notify();

		}
		System.out.println("[TAKED]: " + result.toString());
		return result;
	}
	@Override
	public void clear() {
		synchronized (mutex) {
			head = tail = null;
		}
	}
```

```
	@Override
	public int remaingCapacity() {
		return Integer.MAX_VALUE;
	}
	@Override
	public boolean isEmpty() {
		synchronized (mutex) {
			return head == null;
		}
	}
	static class Node<T> {
		private T value;
		private Node<T> next;
	}
}
```

ArrayBlockingQueue

```
public class ArrayBlockingQueue<T> implements BlockingQueue<T> {
    private int size, capacity;
    private int in, out;
    private Object[] queue;
    private Object mutex;
    public ArrayBlockingQueue() {
        this.size = 0;
        this.capacity = 7;
        this.in = 0;
        this.out = 0;
        this.queue = new Object[capacity];
        this.mutex = new Object();
    }
    public ArrayBlockingQueue(int capacity) {
        if(capacity <= 0)
            throw new IllegalArgumentException("capacity <= 0");
        this.size = 0;
        this.capacity = capacity;
        this.in = 0;
        this.out = 0;
        this.queue = new Object[capacity];
        this.mutex = new Object();
    }
    @Override
    public void put(T object) throws
InterruptedException {
        if(object == null)
          throw new NullPointerException("object ==
null");
        synchronized(mutex) {
            while(size == capacity) {
                mutex.wait();
            }
            queue[in] = object;
            ++size;
            in = (in + 1) % capacity;
            mutex.notify();
        }
    }
    @Override
    public T take() throws InterruptedException {
        synchronized (mutex) {
            while(isEmpty()) {
                mutex.wait();
            }
            @SuppressWarnings("unchecked")
            T result = (T) queue[out];
            queue[out] = null;
            size--;
            out = (out + 1) % capacity;
            mutex.notify();
            return result;
        }
    }
    @Override
    public boolean isEmpty() {
        synchronized (mutex) {
            return size == 0;
        }

    }
    @Override
    public void clear() {
        synchronized(mutex) {
            in = out = 0;
            queue = new Object[capacity];
            size = 0;
            mutex.notify();
        }
    }
    @Override
    public int remainingCapacity() {
        synchronized(mutex) {
            return capacity - size;
        }
    }
    public int size() {
        synchronized(mutex) {
            return size;
        }
    }
}
```