Opgaveløsninger
(sæt 4)

Opgave 12: 6.8 (1 point)

```
import java.util.*;

public class Stack {
    public void push(Object obj) {
        list.addLast(obj);
    }

    public void pop() {
        list.removeLast();
    }

    public Object top() {
        return list.getLast();
    }

    public Object topAndPop() {
        return list.removeLast();
    }

    public boolean isEmpty() {
        return list.isEmpty();
    }

    public void makeEmpty() {
        list.clear();
    }

    private LinkedList list = new LinkedList();
}
```

Opgave 13: 6.9 (1 point)

```
import java.util.*;

public class Queue {
    public void enqueue(Object obj) {
        list.addLast(obj);
    }

    public Object dequeue() {
        return list.removeFirst();
    }

    public boolean isEmpty() {
        return list.isEmpty();
    }

    public void makeEmpty() {
        list.clear();
    }

    private LinkedList list = new LinkedList();
}
```

Opgave 14: 6.12 (2 point, ikke-obligatorisk)

(a)

`findMin` returnerer det forreste element i det sorterede array.

`deleteMin` returnerer det aktuelt forreste element i det sorterede array efter at have parallelforskudt de øvrige elementer en tak fremad i arrayet. Antallet af elementer, `size`, reduceres med 1.

`insert` bestemmer først indsættelsespunktet og parallelforskyder de efterfølgende elementer en tak bagud i arrayet. Herefter sættes elementet ind på indsættelsespunktet. Antallet af elementer, `size`, øges med 1. Arrayet udvides dynamisk efter behov.

(b)

`findMin`: O(1)

`deleteMin`: O(N)

`insert`: O(N)

Bemærkning: Hvis elementerne opbevares i arrayet i faldende orden, kan `deleteMin` implementeres med tidskompleksitet O(1).

(c)

```
public class PriorityQueue {
    public void insert(Comparable obj) {
        if (size == items.length) {
            Comparable[] old = items;
            items = new Comparable[2 * items.length + 1];
            for (int i = 0; i < old.length; i++)
                items[i] = old[i];
        }
        int i = size - 1;
        while (i >= 0 && obj.compareTo(items[i]) < 0) {
            items[i + 1] = items[i];
            i--;
        }
        items[i + 1] = obj;
        size++;
    }

    public Comparable findMin() {
        return items[0];
    }

    public Comparable deleteMin() {
        Comparable result = items[0];
        size--;
        for (int i = 0; i < size; i++)
            items[i] = items[i + 1];
        return result;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public int size() {
        return size;
    }

    public void makeEmpty() {
        size = 0;
    }

    private int size = 0;
    private static final int DEFAULT_CAPACITY = 10;
    private Comparable[] items = new Comparable[DEFAULT_CAPACITY];
}
```

Opgave 15: 6.13 (2 point, ikke-obligatorisk)

(a)

`findMin` finder det mindste element og returnerer dette.

`deleteMin` finder og returnerer det aktuelt mindste element efter at have indsat det bagerste element på fundne elements plads. Antallet af elementer, `size`, reduceres med 1.

`insert` indsætter elementet som det bagerste af de nuværende elementer. Antallet af elementer, `size`, øges med 1. Arrayet udvides dynamisk efter behov.

(b)

`findMin`: O(N)

`deleteMin`: O(N)

`insert`: O(1)

(c)

```
public class PriorityQueue {
    public void insert(Comparable obj) {
        if (size == items.length) {
            Comparable[] old = items;
            items = new Comparable[2 * items.length + 1];
            for (int i = 0; i < old.length; i++)
                items[i] = old[i];
        }
        items[size++] = obj;
    }

    public Comparable findMin() {
        Comparable min = null;
        for (int i = 0; i < size; i++)
            if (min == null || items[i].compareTo(min) < 0)
                min = items[i];
        return min;
    }

    public Comparable deleteMin() {
        int minIndex = -1;
        Comparable min = null;
        for (int i = 0; i < size; i++)
            if (min == null || items[i].compareTo(min) < 0) {
                minIndex = i;
                min = items[i];
            }
        if (minIndex == -1)
            return null;
        items[minIndex] = items[--size];
        return min;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public int size() {
        return size;
    }

    public void makeEmpty() {
        size = 0;
    }

    private int size = 0;
    private static final int DEFAULT_CAPACITY = 10;
    private Comparable[] items = new Comparable[DEFAULT_CAPACITY];
}
```