```java
package com.newton;

import com.newton.tests.Test;

public class Main {
    public static void main(String[] args) {
        Test test = new Test();

        test.executeTest();
    }
}

```

File - /Users/newtonneto/IdeaProjects/heap-with-array/src/com/newton/tests/Test.java

```java
package com.newton.tests;

import com.newton.exceptions.EmptyHeapExceptions;
import com.newton.resources.ArrayHeap;

public class Test {
    public void executeTest() {
        ArrayHeap heap = new ArrayHeap(3);

        System.out.println("isEmpty? " + heap.isEmpty());

        try {
            System.out.println("size " + heap.size());
        } catch (EmptyHeapExceptions e) {
            System.out.println(e);
        }

        try {
            System.out.println("min " + heap.min());
        } catch (EmptyHeapExceptions e) {
            System.out.println(e);
        }

        heap.insert(2);
        heap.insert(5);
        heap.insert(6);
        heap.insert(9);
        heap.insert(7);
        heap.insert(8);
        heap.insert(10);
        heap.insert(4);

        heap.print();

        System.out.println("isEmpty? " + heap.isEmpty());
        System.out.println("size " + heap.size());
        System.out.println("min " + heap.min());
```

```
38
39          heap.removeMin();
40          heap.removeMin();
41          heap.removeMin();
42          heap.removeMin();
43          heap.removeMin();
44          heap.removeMin();
45          heap.removeMin();
46          heap.removeMin();
47          try {
48              heap.removeMin();
49          } catch (EmptyHeapExceptions e) {
50              System.out.println(e);
51          }
52      }
53  }
54
```

File - /Users/newtonneto/IdeaProjects/heap-with-array/src/com/newton/resources/ArrayHeap.java

```java
package com.newton.resources;

import com.newton.exceptions.EmptyHeapExceptions;
import com.newton.interfaces.IArrayHeap;

public class ArrayHeap implements IArrayHeap {
    private Integer capacity;
    private Integer[] heap;
    private Integer next_space;

    public ArrayHeap(Integer capacity) {
        this.capacity = capacity;
        this.heap = new Integer[this.capacity];
        this.next_space = 1;
    }

    @Override
    public void insert(Integer element) {
        if (this.next_space >= this.capacity) {
            System.out.println("Allocating more space...");
            this.resize();
        } {
            this.heap[this.next_space] = element;
            // Verifica se o elemento inserido precisa ser realocado
            upHeap(this.next_space);
            this.next_space++;
        }
    }

    @Override
    public void upHeap(Integer index) {
        // O pai do elemente inserido sempre será seu indice dividido por 2, caso o resultado seja
        // fracionado, considerar apenas a parte inteira
        Integer father_index = index / 2;

        // O indice do pai tem que ser maior que zero, pois a raiz é o indice 1
```

```
37          if (father_index > 0 ) {
38              //Se o valor do index for menor que o
    valor do seu pai, realizar troca
39              if (this.heap[index] < this.heap[
    father_index]) {
40                  Integer temporary_storage = this.
    heap[father_index];
41                  this.heap[father_index] = this.heap
    [index];
42                  this.heap[index] =
    temporary_storage;
43
44                  upHeap(father_index);
45              }
46          }
47      }
48
49      @Override
50      public Integer removeMin() throws
    EmptyHeapExceptions {
51          if (this.isEmpty()) {
52              throw new EmptyHeapExceptions("
    removeMin(): Empty Heap");
53          }
54
55          System.out.println("Estado atual da heap: "
    );
56          this.print();
57
58          Integer min = this.min();
59          // Substitui o primeiro elemento pelo
    ultimo
60          this.heap[1] = this.heap[this.next_space -
    1];
61          // Remove o conteudo do ultimo indice
    utilizado
62          this.heap[this.next_space - 1] = null;
63          // O indice limpo será o proximo espaço
    vago
64          this.next_space--;
65
```

```
66          System.out.println("Primeira remoção: ");
67          this.print();
68
69          // Reordena a heap
70          downHeap(1);
71
72          return min;
73      }
74
75      @Override
76      public void downHeap(Integer index) {
77          // Verica se o indice passado é valido
78          if ((2 * index) < this.next_space && (2 *
    index + 1) < this.next_space) {
79              // Verifica qual o menor dos dois
    filhos
80              if (this.heap[2 * index] < this.heap[2
     * index + 1]) { // O filho esquerdo é menor
81                  // Verifica se o indice atual é
    maior que o seu filho esquerdo
82                  if (this.heap[index] > this.heap[2
     * index]) {
83                      Integer temporary_storage =
    this.heap[index];
84                      this.heap[index] = this.heap[2
     * index];
85                      this.heap[2 * index] =
    temporary_storage;
86
87                      System.out.println("Down Heap
    : ");
88                      this.print();
89
90                      downHeap(2 * index);
91                  }
92              } else { // O filho direito é menor
93                  // Verifica se o indice atual é
    maior que o seu filho direito
94                  if (this.heap[index] > this.heap[2
     * index + 1]) {
95                      Integer temporary_storage =
```

```
 95 this.heap[index];
 96                     this.heap[index] = this.heap[2
     * index + 1];
 97                     this.heap[2 * index + 1] =
    temporary_storage;
 98
 99                     System.out.println("Down Heap
    : ");
100                     this.print();
101
102                     downHeap(2 * index + 1);
103                 }
104             }
105         }
106     }
107
108     @Override
109     public Integer size() {
110         return this.next_space - 1;
111     }
112
113     @Override
114     public Boolean isEmpty() {
115         return (this.heap[1] == null);
116     }
117
118     @Override
119     public Integer min() throws
    EmptyHeapExceptions {
120         if (this.isEmpty()) {
121             throw new EmptyHeapExceptions("min():
    Empty Heap");
122         }
123
124         return this.heap[1];
125     }
126
127     @Override
128     public void print() {
129         for (int index = 1; index < this.
    next_space; index++) {
```

```
130             System.out.print(this.heap[index] +
    " ");
131         }
132         System.out.println("");
133     }
134
135     @Override
136     public void resize() {
137         Integer[] resized_heap = new Integer[this.
    capacity * 2];
138
139         for (Integer index = 1; index < this.
    next_space; index++) {
140             resized_heap[index] = this.heap[index
    ];
141         }
142
143         this.capacity = this.capacity * 2;
144         this.heap = resized_heap;
145     }
146
147
148 }
149
```

```java
package com.newton.exceptions;

public class EmptyHeapExceptions extends
RuntimeException {
    public EmptyHeapExceptions(String error) {
        super(error);
    }
}

```

```java
package com.newton.interfaces;

import com.newton.exceptions.EmptyHeapExceptions;

public interface IArrayHeap {
    void insert(Integer element);
    void upHeap(Integer index);
    Integer removeMin() throws EmptyHeapExceptions;
    void downHeap(Integer index);
    Integer size();
    Boolean isEmpty();
    Integer min() throws EmptyHeapExceptions;
    void print();
    void resize();
}

```