MINMAX

We develop a class MinMax that stores a list of positive integers (including zero), that has getters to get the minimum value, to get the maximum value and to get a copy of the list of integers.

Analysis

Examples

[1,2,3,4,5] → min = 1, max = 5

[2,3,5,1,4] → min = 1, max = 5

We discern the following special situations

Null → We decide to throw an illegal argument exception

[-1] → We decide to throw an illegal argument exception (negative number)

[0,1,2,-3, 2] → We decide to throw an illegal argument exception (negative number)

[] → We decide to throw an illegal argument exception

[1] → We decide to min = 1, max = 1.

Based on these examples we decide to require a list with at least one integer as input. The exceptions get the string 'A list with at least one integer is required' as parameter.

[1,1] → We decide min = 1, max = 1 (in case of equal min/max values)

[1,1, 2,4,4] → min = 1, max = 4

[1,2,4,4] → min = 1, max = 4

The API

MinMax is a class that stores a list with integers and has getters for this list, the minimum and maximum values of these integers in the list.

```
public class MinMax {

@subcontract Happy path {
  @requires xs contains a list with positive integers (0, 1, 2 …) with at least
one member
  @ensures A MinMax object is created, that stores the list with numbers
}
@subcontract Empty list  {
  @requires xs is an empty list
  @signals An illegalArgumentException(A list with at least one positive integer
                                       is required)
}
@subcontract Null value {
  @requires Null as parameter
  @signals An illegalArgumentException(A list with at least one integer
                                       is required)
}
@subcontract xs has negatives in the list {
  @requires One or more members are negative integers in list xs
  @signals An illegalArgumentException(A list with only positive integers
                                       is required)
}
public MinMax(int…xs) throws illegalArgumentException

@subcontract Happy path {
  @requires true
  @ensures \result = The minimum value in the list
}
public int getMin()

@subcontract Happy path {
  @requires true
  @ensures \result = The maximum value in the list
}
public int getMax()

@subcontract Happy path {
  @requires true
  @ensures \result = copy of the list
}
public int[] getIntegers()

}
```

The IPI

We decide to add three attributes
1) intList of type int[]: Representing the list with integers
2) min of type int: Representing the minimum value of intList
3) max of type int: Representing the maximum value of intList

We add the following rules
1) length(intList) >= 1
2) min: is the minimum value in intList
3) max: is the maximum value in intList

The API/IPI becomes (IPI in red):

```
/**
  @desc MinMax is a class that stores a list with integers and has
        a getter for this list, and getters for the minimum and maximum values of
        these integers.
  @represents min = the minimum value in intList
  @represents max = the maximum value in intList
  @represents L = intList

  @inv intList.length >= 1
  @inv all elements of intList are >= 0
*/
```

```
public class MinMax {

@subcontract Happy path {
  @requires xs contains a list with positive integers (0, 1, 2 …) with at least
one member
  @ensures A MinMax object is created, that stores the list with numbers
}
@subcontract Empty list  {
  @requires xs is an empty list
  @signals An illegalArgumentException(A list with at least one positive integer
                                       is required)
}
@subcontract Null value {
  @requires Null as parameter
  @signals An illegalArgumentException(A list with at least one integer
                                       is required)
}
@subcontract xs has negatives in the list {
  @requires One or more members are negative integers in list xs
  @signals An illegalArgumentException(A list with only positive integers
                                       is required)
}
public MinMax(int…xs) throws illegalArgumentException

@subcontract Happy path {
  @requires true
  @ensures \result = The minimum value in the list
}
public int getMin()

@subcontract Happy path {
  @requires true
  @ensures \result = The maximum value in the list
}
public int getMax()

@subcontract Happy path {
  @requires true
  @ensures \result = copy of the list
}
public int[] getIntegers()

}
```

Implementation

We choose to calculate and store the minimum and maximum values in de constructor. To determine the minimum and maximum values, we choose to sort the array (ascending) and to take the first and last value of this array. Other implementations are possible, for example using a stream or a for-loop and determining the minimum and maximum values integer after integer.

The getters are simple.

In an alternative implementation, the getters calculate the min and max value if not calculated before and stores these as attribute values. We did not implement this choose.

```
/**
@desc MinMax is a class that stores an unordered list with integers (L) and has
      getters for L, the minimum and maximum values of these integers of L.
@represents min = the minimum value in intList
@represents max = the maximum value in intList
@represents L = intList
@inv intList.length >= 1
@inv all elements of intList are >= 0
*/

public class MinMax {

  private final int[] intList;
  private final int min;
  private final int max;
  private static final String ERRORMESSAGE =
          "A list with at least one positive integer is required";
  /**
    @subcontract Happy path {
      @requires An unordered list (L) with positive integers (0, 1, 2 …) with at least
                 one member
      @ensures A MinMax object is created, that stores L
    }
    @subcontract Empty list  {
      @requires An empty list
      @signals An illegalArgumentException(A list with at least one positive integer
               is required)
    }
    @subcontract Null value {
      @requires Null as parameter
      @signals An illegalArgumentException(A list with at least one integer
              is required)
    }
    @subcontract Negatives in list {
      @requires One or more members are negative integers
      @signals An illegalArgumentException(A list with at least one integer is
               required)
    }
  */
  public MinMax(int...xs) throws IllegalArgumentException {
         // Robustness
         // null as parameter / zero or one integer in the list
         if (xs == null || xs.length < 1 ) throw new
               IllegalArgumentException(ERRORMESSAGE);
         // if there are negative values in the array
         var min = Arrays.stream(xs).min().getAsInt();
         if (min < 0) throw new IllegalArgumentException(ERRORMESSAGE);

         // Happy path; store the list and determine the min and max values
         this.intList = xs;
         this.min = min;
```

```
            this.max = Arrays.stream(intList).max().getAsInt();
    }

    /**
     @subcontract Happy path {
       @requires true
       @ensures \result = The minimum value in L
     }
    */
    public int getMin() {
      return min;
    }

    /**
     @subcontract Happy path {
       @requires true
       @ensures \result = The maximum value in L
     }
     */
     public int getMax() {
       return max;
     }

    /**
     @subcontract Happy path {
       @requires true
       @ensures \result = a copy of L
     }
     */
     public int[] getIntegers() {
       return Arrays.copyOf(intList,  intList.length);
     }

}
```

Tests

Only blackbox tests are needed (these tests comprise the IPI and whitebox tests. The test cases based on the contracts are (in pseudo code):

```
@subcontract Happy path
// Some test cases are (Actually, one test case is enough)
new MinMax([1,2,3,4,5]) → getMin() == 1, getMax() == 5
new MinMax([1,1]) → getMin() == 1, getMax() == 1
new MinMax([4,2,3,3,5,6,6]) → getMin() == 2, getMax() == 6

@subcontract Empty list
new MinMax([]) → IllegalArgumentException()

@subcontract List with one integer  {
new MinMax([1]) → IllegalArgumentException()

@subcontract null value {
new MinMax(null) → IllegalArgumentException()

@subcontract Negative value
new MinMax([-4,-2,3,3,5,4,6]) → IllegalArgumentException()
```