German University in Cairo
Media Engineering and Technology
Prof. Dr. Slim Abdennadher
Dr. Wael Abouelsaadat
Dr. Mohammed Abdel Megeed

Introduction to Computer Programming, Spring Term 2017
Practice Assignment 8

Discussion: 6.5.2017 - 11.5.2017

Exercise 8-1 Arrays: Smallest Value, Largest Value, Arithmetic Mean

a) Write a Java program to compute the smallest as well as the largest value in a series of numbers.

b) Modify the program to print the smallest as well as the largest values and their positions.

c) Modify the program to find the arithmetic mean of the numbers.

Solution:

```
class Arith {
  public static void main ( String[] args ) {
    int[] array = { -20, 19, 1, 5, -1, 27, 19, 5 } ;
    int   max;
    int min;
    int sum = 0;
    int average;
    max = array[0];
    min = array[0];


    for ( int index=0; index < array.length; index++ )
    {
      if ( array[ index ] > max )
        max = array[ index ];

      if ( array[ index ] < min )
        min = array[ index ];

      sum += array[ index ];

    }
    System.out.println("The maximum of this array is:" + max );
    System.out.println("The minimum of this array is:" + min );
    int aver = sum/array.length;
    System.out.println("The arithmetic mean of this array is:" + aver );
  }
}
```

Exercise 8-2 Palindrome
To be discussed in tutorials

Write a Java program that tests whether the elements of an array consist of a palindrome. A palindrome is any word which is the same forward and backward, e.g, radar, noon, anna, ...

Assume that the words are stored in an array of characters, e.g. the word `radar` will be stored in the array `{'r','a','d','a','r'}`

Solution:

```
public class Palindrome {

    public static void main(String[] args) {

        char[] w = {'R', 'A', 'D', 'A', 'A'};
        int n = w.length;
        boolean b = true;
        for (int i = 0; (i<n/2 &&  b == true); i++)
            b = w[i] == w[n-1-i];
        System.out.println(b);
    }

}
```

Exercise 8-3 Union and Intersection
To be discussed in labs

Write Java methods `Union` and `Intersection` that, given two arrays of integers, prints the union and intersection of the two arrays. Assume that the arrays do not consist of duplicates. For example, given:

- Array1 = 12, 32, 14, 35, 89, 16, 120
- Array2 = 9, 12, 8, 17, 120, 35, 36

your class should print the following:

- For `Union`:

  ```
  12 32 14 35 89 16 120 9 8 17 36
  ```

- For `Intersection`:

  ```
  12 35 120
  ```

Solution:

```
class UnionIntersect {
  public static void intersect(int[] A, int[] B) {

    for (int i = 0; i < A.length; i++) {
      for (int j = 0; j < B.length; j++) {
        if (A[i] == B[j])
          System.out.print(A[i] + "␣");
      }
    }
  }

  public static void union(int[] A, int[] B) {
    for (int i = 0; i < A.length; i++) {
      System.out.print(A[i] + "␣");
    }
    for(int j = 0; j < B.length; j++){
```

```
      boolean unique = true;
      for(int i = 0; i < A.length; i++) {
        if(A[i] == B[j])
          unique = false;
      }
      if(unique)
        System.out.print(B[j] + " ");
    }
  }

  public static void main(String[] args) {
    int[] A = {1, 2, 3, 4};
    int[] B = { 1, 4, 2, 6};
    intersect(A, B);
    System.out.println();
    union(A, B);
  }
}
```

Another Solution:

```
class UnionIntersect2 {
  public static void intersect(int[] A, int[] B) {
    int [] result = new int[(A.length<B.length)?A.length:B.length];
    int elemCount = 0;
    for (int i = 0; i < A.length; i++) {
      for (int j = 0; j < B.length; j++) {
        if (A[i] == B[j])
          result[elemCount++] = A[i];
      }
    }
    for(int i = 0; i < elemCount; i++)
      System.out.print(result[i] + " ");
  }

  public static void union(int[] A, int[] B) {
    int [] result = new int[A.length+B.length];
    int elemCount = 0;
    for (int i = 0; i < A.length; i++) {
      result[elemCount++] = A[i];
    }
    for(int j = 0; j < B.length; j++){
      boolean unique = true;
      for(int i = 0; i < result.length; i++) {
        if(B[j] == result[i])
          unique = false;
      }
      if(unique)
        result[elemCount++] = B[j];
    }
    for(int i = 0; i < elemCount; i++)
        System.out.print(result[i] + " ");
  }

  public static void main(String[] args) {
    int[] A = {1, 2, 3, 4};
    int[] B = {1, 4, 2, 6};
    intersect(A, B);
```

```
    System.out.println();
    union(A, B);
  }
}
```

Exercise 8-4 Split Array
To be discussed in labs

Write a method `void split(int array[], int pivot, int size)` which partitions a given array into two parts: one with all elements with values `<= pivot` and the other one with all elements with values `> pivot`. The array should be partitioned without the use of a new array and using the minimum number of steps, i.e. the resulting array should not be sorted. For example, if the array is

13	-42	8	35	-7	46	28	-19

and the `pivot` is 10, then the resulting array should be

-19	-42	8	-7	35	46	28	13

in which all elements less than the `pivot` are separated from those greater than the `pivot`.

Solution:

```
class Split {
  static void split(int array[], int pivot, int size) {
    int i = 0;
    int j = size - 1;
    while (i <= j) {
      if (array[i] > pivot && array[j] <= pivot) {
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
        i++;
        j--;
      }
      else if (array[i] > pivot) {
        j--;
      }
      else {
          if(array[j] <= pivot)
                i++;
        else
            {
                j--;
                i++;
            }
      }
    }
  }

  public static void main(String args[]) {
    int[] anArray = {13, -42, 8, 35, -7, 46, 28, -19};
    split(anArray, 10, 8);
    for (int i = 0; i < anArray.length; i++)
      System.out.print(anArray[i] + " ");
  }
}
```

Exercise 8-5 Subset

Write a Java method `subset` that takes two arrays of integers as parameters and **returns** `true` if and only if the first array is a subset of the second array, otherwise the method should return `false`. Assume that the arrays do not consist of duplicates. For example:

```
subset({1,2,3}, {1,2,3,5,6}) returns true
subset({1,2,3}, {2,4,5,1,3}) returns true
subset({}, {1,2,3,5,6}) returns true
subset({1,2,3}, {2,4,5,1}) returns false
```

Write a `main` method to test your program. The `main` method should display either

```
Array 1 is a subset of  Array 2
```

or

```
Array 1 is not a subset of Array 2
```

Solution:

```
class Subset
{

public static boolean member(int x, int[] a) {
    int n = a.length;
    for (int i = 0; i < n; i++) {
        if (x == a[i]) return true;
    }
    return false;
}


public static boolean subset(int[] sub, int[] sup) {
    int m = sub.length;
    for (int i = 0; i < m; i++)
        if (!member(sub[i], sup)) return false;
    return true;
}



    public static void main (String[] args) {
        int[] a = {1,2,6};
        int[] b = {1,2,6,3,7,4,8,5};

        if (subset(a,b))
           System.out.println("Array 1 is contained in Array 2");
        else
           System.out.println("Array 1 is not contained in Array 2");
    }
}
```

Exercise 8-6 MiniString - Final Spring 2010
To be discussed in tutorials

The `String` class is one of the most useful classes among those provided by the Java Standard Class Library, but how does it actually work?

The purpose of the exercise is to implement a similar class.

Write a class called `MiniString`; like `String` objects, objects that belong to the `MiniString` class represent ordered sequences of characters. Also like `String` objects, valid character indices in a `MiniString` range from 0 (inclusive) to the number of characters in the `String` (exclusive); that is, the i^{th} character of a `MiniString` has index $i - 1$.

`MiniString` objects MUST keep track of the character sequences they represent using an array (of `char`); you **ARE NOT ALLOWED** to use regular `String`s within the `MiniString` class under ANY circumstances. Your `MiniString` should provide the following public methods:

- A constructor that takes no parameters, and initializes the newly-created `MiniString` so that it represents an empty character sequence, that is, one that contains no characters.
- A second constructor, which takes as a parameter an array of `char`, and initializes the newly created `MiniString` so that it represents the sequence of characters currently contained in the array passed as parameter. The contents of the character array are copied, so that any subsequent modification of the character array does not affect the newly-created `MiniString`
- A method called `length()`, which takes no parameters and returns a value of type `int` representing the length of this `MiniString`, that is, the number of characters it contains.
- A method called `charAt()`, which takes as a parameter a value of type `int`, and returns a value of type `char` representing the character in the `MiniString` at the position given by the parameter.
- A method called `concat()`, which takes as a parameter a `MiniString`, and returns a new `MiniString` representing the concatenation of the `MiniString` this method is called on and the `MiniString` parameter, in that order. For example, if the `MiniString` this method is called on represents the character sequence "`CSEN`", and the parameter `MiniString` represents the character sequence "`202`", then the method should return a new `MiniString` representing the character sequence "`CSEN202`". If the `MiniString` passed as parameter is `null`, then the method MUST return a reference to the `MiniString` on which this method is called.
- A method called `equals()`, which takes as a parameter a `MiniString`, and returns a value of type `boolean`. This method returns `true` if the `MiniString` this method is called on represents the same character sequence as the `MiniString` parameter (that is, both contain the same number of characters in the same order), `false` otherwise. If the `MiniString` passed as parameter is `null`, then the method MUST return `false`.
- A `main` method that creates at least two `MiniString` objects and calls all methods described above.

Solution:

```
public class MiniString {
    char[] list;

    public MiniString() {
        // Creates an empty MiniString.
        list = new char[0];
    }
    public MiniString(char[] c) {
        // Creates a MiniString with the same chars as in c
        list = new char[c.length];

        for(int i = 0; i < c.length; i++)
            list[i] = c[i];
    }

    public int length() {
        //returns the length of the MiniString
```

```
    return list.length;
}

public char charAt(int i) {
    //returns the char at index i within the MiniString
    return list[i];
}
public MiniString concat(MiniString m) {
    //creates a new String as a result of concatenating
    // this MiniString and MiniString m and
    // returns a reference to this if m is null

    if (m==null)
        return this;

    else {
        char[] r = new char[m.list.length + this.list.length];

        for (int i =0; i<list.length; i++)
            r[i] = list[i];

        int k = 0;

        for (int j = list.length; j<r.length; j++)
        {
            r[j] = m.list[k];
            k++;
        }

        return new MiniString(r);
    }
}

public boolean equals(MiniString m) {
    //compares two miniStrings and return false if they are not equal
    // and true otherwise

    if (m == null)
        return false;

    if (m.list.length != this.list.length)
        return false;

    for (int i = 0;i<list.length; i++ )
        if (m.list[i]!= this.list[i])
            return false;

    return true;
}




public static void main (String args []) {
    MiniString m = new MiniString();
```

```
        char [] a = {'S','l','i','m'};
        MiniString s = new MiniString(a);

        char [] b = {'C','S','E','N','2','0','2'};
        MiniString l = new MiniString(b);

        System.out.println("m.length() is " + m.length());
        System.out.println("the first char in a is "+ s.charAt(0));

        MiniString newM = m.concat(l);

        System.out.println("newM.length() is "+ newM.length());
        System.out.println(l.equals(null));
    }
}
```