```
package ds_arrays_assignment;

import java.util.Random;
import ds_arrays.*;
//import ds_arrays.MyArrayIF;
//import ds_arrays.MyUnorderedArray;

/**
 *
 * @author ogm2
 */
public class DS_Arrays_Assignment {

    public static final int SIZE = 18;

    public DS_Arrays_Assignment() {
    }

    public void run() {
        Random rd = new Random();
        MyArrayIF<Integer> myArray = this.generateNewArray(SIZE);
        myArray.display();
        System.out.println("");
        PlayWithArraysIF pwa = new PlayWithArraysImpl();
        System.out.println("Size of array is " + pwa.findSize(myArray));
        System.out.println("Max value is " + pwa.findLargestValue(myArray));
        System.out.println("2nd largest val is " +
pwa.findSecondLargestValue(myArray));
        System.out.println("\nIntersection with");
        MyArrayIF<Integer> myArray2 = this.generateNewArray(SIZE % 10);
        myArray2.display();
        MyArrayIF<Integer> myArray3 = this.generateNewArray(SIZE * 2);
        myArray3.display();
        myArray = pwa.intersect(myArray, myArray2, myArray3);
        myArray.display();


    }

    public MyArrayIF<Integer> generateNewArray(int size) {
        Integer[] arr = new Integer[size];
        for(int i = 0; i  < size; i++)
            arr[i] = 0;
        MyArrayIF<Integer> myArray = new MyUnorderedArray<>(arr);
        Random rd = new Random();
        for(int i = 0; i < size; i++)
```

```
         myArray.insert(rd.nextInt(101));
      return myArray;
   }
   /**
    * @param args the command line arguments
    */
   public static void main(String[] args) {
      new DS_Arrays_Assignment().run();
   }

}
package ds_arrays_assignment;

import ds_arrays.*;

public interface PlayWithArraysIF <E extends Comparable<E>> {

   /**
    * Determines the size of an array.
    * @param a  the array whose size is determined
    * @return  the size of a
    */
   public int findSize(MyArrayIF<E> a);

   /**
    * Determines the largest value stored in an array.
    * @param a  the array whose max is determined
    * @return  the max value stored in a
    */
   public E findLargestValue(MyArrayIF<E> a);

   /**
    * Determines the second largest value stored in an array.
    * @param a  the array whose second largest value is determined
    * @return  the second largest value stored in a
    */
   public E findSecondLargestValue(MyArrayIF<E> a);

   /**
    * Finds all the values that are common to three different arrays.
    * HINT: you are allowed to modify a1 so that it ends up storing common
    * values only.
    * @param a1  the first array to intersect
    * @param a2  the second array to intersect
    * @param a3  the third array to intersect
    * @return  the intersection of a1, a2, and a3
    */
   public MyArrayIF<E> intersect(MyArrayIF<E> a1, MyArrayIF<E> a2,
MyArrayIF<E> a3);

}

package ds_arrays_assignment;

import ds_arrays.MyArrayIF;
import ds_arrays.MyUnorderedArray;


public class PlayWithArraysImpl<E extends Comparable<E>> implements
PlayWithArraysIF<E> {

   public PlayWithArraysImpl() {
   }

   @Override
   public int findSize(MyArrayIF<E> a) // O(n)
   {
      int i=0;
      try{



         //a.getElementAt(i);
         while(a.getElementAt(i)!=null)
         {
            i++;
         }

   }catch(IndexOutOfBoundsException E){}
      return i;
   }


   @Override
   public E findLargestValue(MyArrayIF<E> a) { //O(N)
      E temp=a.getElementAt(0);
      try{
         int i=1;

         while(a.getElementAt(i)!=null)
         {
           if(a.getElementAt(i).compareTo(temp)>0)
```

```
            {
               temp=a.getElementAt(i);
            }
            i++;

          }
       }catch(IndexOutOfBoundsException E){}
          return temp;
       //throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public E findSecondLargestValue(MyArrayIF<E> a)// 2(N)-> O(N)
    {
       E temp=a.getElementAt(0);
       E temp2=a.getElementAt(0);
       try{

       int i=1;
       int f=1;

       E r=this.findLargestValue(a);
       while(a.getElementAt(f)!=null)
       {

if((a.getElementAt(f).compareTo(temp2)>0)&&(a.getElementAt(f).compareTo(r)<0
))
          {
             temp2=a.getElementAt(f);
          }
          f++;
       }
//          if(a.getElementAt(i).compareTo(temp2)>0)
//          {
//             temp2=a.getElementAt(i);
//          }
//          i++;
//          return temp2;
//       }
//
//       while(a.getElementAt(f)!=null)
//       {
//          if((a.getElementAt(f).compareTo(temp)>0) &&
a.getElementAt(f).compareTo(temp2)<0)
//              temp=a.getElementAt(f);
//          f++;
//       }

   }catch(IndexOutOfBoundsException E){}
       return temp2;
       //throw new UnsupportedOperationException("Not supported yet.");
   }

   @Override
   public MyArrayIF<E> intersect(MyArrayIF<E> a1, MyArrayIF<E> a2,
MyArrayIF<E> a3) {
      try{
       int n1=this.findSize(a1);
       int n2=this.findSize(a2);
       int n3=this.findSize(a3);

       //int n4=0;
       //MyArrayIF<E> a4;
       for(int i=0; i<=n1; i++) // iterating through Array 1
       {
         if(a2.find(a1.getElementAt(i))!=-1 && a3.find(a1.getElementAt(i))!=-1)
         {

         }
          else{
             a1.delete(a1.getElementAt(i));
             i--;
          }
       }
   }catch(IndexOutOfBoundsException E){}
       return a1;
//throw new UnsupportedOperationException("Not supported yet.");
   }



}
```