Lab 4. Implementing ADT using array

Theme. In this lab, you will:

- use **set** to implement a Java application
- use a Java interface to specify the operations of the set data structure
- implement set intersection using an array implementation of set
- practise the use of generic types
- practise the use of enhanced for (for-each) statement

Key concepts: implementing set ADT, set operations, using the set data structure

Required file(s): `lab4.zip`

1 Getting started...

1. In a web browser, download the archive `lab4.zip` from Blackboard and extract its contents into your `Eclipse` workspace for this module.

2. Start up `Eclipse`.

3. Making use of the contents of your extracted archive, create a new Java project named **lottery_ArraySet** in your `Eclipse` workspace using the contents of your extracted archive.

2 Working with Sets

A set is a collection of *unique* values *without* any particular order. A set can be modelled using a linear structure, so long as we ignore the element order within the linear structure.

Lottery is an application that is ideal for using sets as its main data structure.

In a lottery session, players buy one or more lottery ticket during a purchase period. Each lottery ticket contains 6 *unique* numbers that are within a predefined range. This range is set by the lottery organiser, e.g. 1 to 59 in the UK National Lottery. The purchase period ends just before the draw takes place. Immediately after the purchase period has ended, 6 numbers within the predefined range are drawn by the lottery organiser. The players with all 6 numbers on their lottery ticket win the jackpot (i.e. a big prize).

The archive `lab4.zip` contains a partially-implemented lottery application. Your task is to complete the given implementation according to the following requirements. A successful completion enables the simulation of running a lottery session to take place.

Hint: The rough locations where you are expected to add your Java code and relevant hints for accomplishing the tasks have been marked throughout the given Java programs. Look out for **block comments** that include a sequence of *four* exclamation marks, i.e.:

```
/* !!!!  ...  ...  */
```

The locations where you are expected to pay particular attention on the given Java code have also been annotated. Answer the given questions, if any. Look out for **block comments** that include a sequence of *four* plus signs, i.e.:

```
/* ++++ ...  ...  */
```

1. Class `AbstractSet<T>` implements interface `SetADT<T>`. This class has method `addAll(SetADT<T>)`, but we cannot use it in our **lottery_ArraySet** project because its implementing interface `SetADT<T>` does not specify this method.

   Add the missing abstract method to the interface `SetADT<T>` to allow a set to add all given elements in another set to itself.

2. Add the abstract method `intersection(SetADT<T>)` to interface `SetADT<T>` for specifying set intersection. The result of set intersection is a new `SetADT<T>` object.

3. Add the abstract method `removeRandom` to interface `SetADT<T>` for removing and returning a random item from the set.

4. Complete the implementation of method `expandCapacity` in class `ArraySet<T>`. This method enables an `ArraySet<T>` object to act as an unbounded set.

5. Complete the implementation for the copy constructor `ArraySet<T>(ArraySet<T>)` in class `ArraySet<T>`. This constructor creates a new `ArraySet<T>` object and initialises it with the same elements as the given set object.

Hint: Arrays can be cloned effectively using their `clone()` method. The result is a **new array** which is identical to the array on which we perform the clone operation.

6. Complete the implementation for method `union(SetADT<T>)` in class `ArraySet<T>`.

7. Complete the implementation for method `intersection(SetADT<T>)` in class `ArraySet<T>`.

Hint: Further instructions about how to implement this method can be found in the source code for Unit 4 (cf. the ZIP archive `unit04_ArrayADTs.zip`). If you would like to approach implementing this method differently, do note which approach will result in a more efficient set intersection.

8. The class `ArraySet<T>` implements the interface `SetADT<T>`. This class has the method `difference` (as specified in `SetADT<T>`) which was not implemented in the given code listing for the class `ArraySet<T>`.

   Write the method `difference`.

9. The implementation for the constructor `LotteryTicket(int, Lottery)` in the class `LotteryTicket` is incomplete. Add the missing code.

10. The implementation for the constructor `LotteryTicket(int, Lottery, SetADT<Integer>)` in the class `LotteryTicket` is incomplete. Add the missing line of code to the constructor.

11. There are numerous bits of code missing from the class `Lottery`, e.g.:

    - initialising various instance variables,
    - the body of methods `sellTicket()` and `sellTicket(SetADT<Integer>)`,
    - an iteration routine for drawing winning numbers, and
    - an iteration routine to check who has won.

    Add the missing implementation to this class.

3 Testing

Now test your implementation to see if it meets the above requirements. You may use the given `main(String[])` method in each of your modified class to perform quick, but limited, unit testing.

Run *at least two consecutive* lottery sessions. In each lottery session, try buying a few Lucky Dip tickets and at least one normal ticket. Run the lucky draw and check who has won. To help the checking process, the check win operation also shows all tickets for this lottery session.

You may like to increase the range of numbers available for this lottery application from 7 to 14, then to 49, 59, etc and see if anyone wins the jackpot.

4 Further Challenges

1. Consider the class `ArraySet<T>`. All parameters and return values of the methods in this class are of the type `SetADT<T>`. Could we have used the type `ArraySet<T>` instead of `SetADT<T>`?
   Explain why using `SetADT<T>` here is considered to be a better object-oriented design and hence, more beneficial.

2. In the given lottery application, we used a set to keep the integers that represent our set of potential lottery winning numbers. Rather than using the primitive type `int`, the reference type `Integer` is used in our set collection, i.e.:

```
1    availableNumbers = new ArraySet<>(largest);
```

   However, later on we add primitive `int` values to this set collection, i.e.:

```
1    // populate the set with all valid lottery numbers
2    for(int i=1; i <= largest; i++)
3    {
4          availableNumbers.add(i);
5    }
```

   (a) Why did we need to use the reference type `Integer` in the first place? Could we have used the primitive type `int` instead?

   (b) Why can we assign an `int` value to an `Integer` object without getting a compilation error? What must have happened "behind the scene" **before** the assignment could take place?

3. Consider the following implementation of class `ArrayIterator`:

```
1  package dsa;
2
3  import java.util.Iterator;
4
5  /**
6   * An array implementation of an iterator over a collection.
7   *
8   * @param <T>
9   */
10 public class ArrayIterator<T> implements Iterator<T> {
11     private T[] contents;  // the elements to be iterated over
12     // the index of the element that is to be returned in the next() operation
13     private int cursor;
14     private int limit;
15
16     /**
17      * Constructor: constructs an iterator (which is backed by an array)
18      *  for iterating over the elements in the specified array.
19      *
20      * @param array
21      * @param size
22      */
23     public ArrayIterator(T[] array, int size) {
24         contents = array;
25         limit = size;
26         cursor = 0;
27     }
28
29     @Override
30     public boolean hasNext() {
31         return cursor < limit;
32     }
33
34     @Override
35     public T next() {
36         if (!hasNext()) {
37             throw new
38                 IllegalStateException("ArrayIterator: no next element");
39         } else {
40             return contents[cursor++];
41         }
42     }
43 }
```

(a) Interface `java.util.Iterator` includes method `remove`, but `ArrayIterator` does not include a definition of method `remove`. Would this cause a compilation error? Justify your answer.

(b) Consider the definition of class `ArraySet` in this practical. A programmer has decided to add method `retainAll(SetADT<T>)` in `ArraySet` with the following implementation:

```
1    public boolean retainAll(SetADT<T>  A) {
2        boolean hasBeenModified = false;
3        Iterator<T> iter = this.iterator();
4        while (iter.hasNext()) {
5            if (!A.contains(iter.next())) {
6                iter.remove();
7                hasBeenModified = true;
8            }
9        }
10       return hasBeenModified;
11   }
```

What would happen when method `retainAll` is invoked?

(c) Write an improved implementation for method `retainAll`.