PAL Worksheet Sets CSC20

Topics: Collection Interface, Set Interface
Ref: CSC20 PAL WS Inheritance
CSC20 PAL Polymorphism and Interfaces

The Collection Interface

A Collection is an object that stores a group of objects called elements. Collections use data structures to store and manage data. They are categorized by the types of elements that they store, the operations they allow you to perform on these elements, and the speed and efficiency of these operations. Some examples of collections are List, Set, Map, Stack, Queue etc.

Java provides a large and useful group of collections that allow store, access, search, sort and manipulate data in a variety of ways. Together, these collections and classes are known as the Java Collections Framework. This frame work is contained largely in java.util.*

Sets

One of the limitations with linear data structures like lists is that searching for an element takes a long time – you have to look over all preceding elements before locating one. They can also store duplicate elements, which is not desirable for certain applications.

The Set Interface in Java is a Collection and is implemented as either a HashSet or a TreeSet. It uses data structures which are quick to search and do not store any duplicates, just was we would expect a mathematical set to do. Since Set inherits from the Collection interface, it's implementations are bound to implement all the methods of the Collection interface. You can see a list of some useful Collection methods in Chapter 11 of the BJP textbook. Some examples are *add, remove, size* etc.

The implementation of Set that we will use is HashSet. It is based on a *hash table* structure which makes searching, adding, and removing all very quick operations.

For example, it would be more appropriate to use a Set for a social network application instead of a LinkedList, because every user mandates their own unique profile. Duplicates of any profile would be misleading and chaotic.

We can declare a Set by its HashSet implementation and add items to it as such:

```
Set<Integer> integerSet = new HashSet<>();
integerSet.add(2);
integerSet.add(4);

integerSet.add(2); // this duplicate item will not be stored
```

Ex.1) Write a Java method which takes in an array of integers, (returns nothing)then uses both a LinkedList and a Set to store unique elements (i.e. without duplication). You will see that it is simpler with the Set than the LinkedList.

Ex.2) Write a Java method which uses a Set to determine the number of unique email addresses stored in a database. Email addresses should be provided as an array of Strings. It returns an integer representing the number of unique email addresses.

Sets also provide set operations with built-in methods like: setA.addAll(setB) for set union, setA.retainAll(setB) for set intersection, and setA.removeAll(setB) for set difference.

For example, here is how to perform set union, which provides the set of all elements which are in set A, set B, or both. Note there will be no duplicates stored.

```
HashSet<Integer> setA = new HashSet<>(Arrays.asList(2,4,6,8));
HashSet<Integer> setB = new HashSet<>(Arrays.asList(6,8,10,12));

setA.addAll(setB); // setA is now [2, 4, 6, 8, 10, 12]
```

Q3) We have two databases of email addresses and want to determine which addresses are common to both. Write a method which takes in two String arrays as input and finds and prints their common elements, using HashSet. Its return type is void.