Programming challenges in Java

6. Using collections in Java

Nacho Iborra

IES San Vicente

Table of Contents

Programming challenges in Java

1. Introduction

If we want to create an undetermined sequence of elements, we need to use a collection. There are several types of collections in Java, and we must choose the most appropriate depending on the problem that we are trying to solve.

The main types of collections are:

- Lists: an ordered sequence of elements, that can be of any type (integers, strings, objects...). Each elements has a specified index in the collection, starting at 0. There are some concrete types of lists such as *ArrayList* (simple linked list), *Stack...*
- Maps: a sequence of key-value pairs, in which every value is associated to its corresponding key. There is no "user-controlled" order in this structure. Data is order according to a hash value from the keys, so we can't make sure that information will be displayed in a given order. The most usual type to deal with maps is *HashMap* class, but there are some other types.
- Sets: a sequence of unordered elements, in which none of them can be repeated. In this case, there is no numeric index to find each element, we can only iterate over the collection to explore all the elements.

In this session, we are going to focus on two of these collection types: Lists and Maps.

2. Working with lists

The **List** interface in Java defines all the operations that can be performed by any concrete type of list, such as ArrayLists, Vectors or Stacks, among others. In this tutorial, we are going to focus on working with ArrayLists, because they are the most usual list type.

2.1. Creating lists

If we want to create a list, we should use a generic list, this is, a parameterized list in which we specified the data type that will be stored in it. For instance, if we want to create a list of strings, we can do it this way:

```
List<String> myStrings = new ArrayList<>();
```

But we can also create any type of list, such as:

```
List<Integer> myNumbers = new ArrayList<>();
List<Book> myBooks = new ArrayList<>();
```

2.2. Adding elements to the list

The most usual way to add an element to a list is the *add* method. Let's add an element to each list created in previous subsection:

```
myStrings.add("Hello");
myNumbers.add(2);
myBooks.add(new Book("Ender's game", "Orson Scott Card"));
```

2.3. Getting and exploring the elements

We can get an element from the list with *get* method, indicating the position of the element in the list (starting by 0).

```
String s = myStrings.get(2);
```

We can also explore the whole list with a *for* loop, and determine the total length of the list with *size* method:

```
for (int i = 0; i < myBooks.size(); i++)
{
    Book b = myBooks.get(i);
    System.out.println(b);
}
```

2.4. Removing an element

We can remove an element from a list with the *remove* method, indicating its position.

```
myBooks.remove(3);
```

We can also use the *remove* method indicating the object to be removed, but in this case all the elements of the list need to be compared between them (we need to override the *equals* method).

2.5. Other useful operations

There are some other useful operations that we can do with lists:

- `add (pos, element)` inserts the element at the specified position of the list
- `contains(element)` checks if the list contains the specified elements. Again, we need to override the *equals* method of the element class to make sure that this method will work.
- `indexOf(element)` determines the position of the given element (as long as we override the *equals* method, as well).
- `set (pos, element)` changes the value of the element at position *pos* with the new *element* given as a parameter.

3. Working with maps

The **Map** interface in Java provides some useful methods to deal with maps or dictionaries. We will apply these methods with *HashMap* class.

3.1. Creating maps

If we want to create a map, we must specify the data type for both the key (typically a string or an integer) and the value (any object type). For instance:

```
HashMap<String, Book> myLibrary = new HashMap<String, Book>();
```

3.2. Adding elements to the map

In order to add any element to the map, we need to specify both the key and the value in the *put* method. If the element already exists, then its old value will be replaced with the new one:

```
myLibrary.put("11A228ABC4", new Book("Ender's game", "Orson Scott Card"));
```

We can also check if the map contains the element before putting the new value, with *containsKey* method:

```
if (!myLibrary.containsKey("11A228ABC4"))
    myLibrary.put("11A228ABC4", new Book("Ender's game", "Orson Scott Card"));
```

3.3. Getting and exploring the elements

We can get an element from the map with *get* method, indicating the key of the element.

```
Book b = myLibrary.get("11A228ABC4");
```

We can also explore the whole map with:

```
for(String key : myLibrary.keySet())
{
    Book b = myLibrary.get(key);
    System.out.println(b);
}
```

3.4. Removing an element

We can remove an element from a map with the *remove* method, indicating its key.

```
myLibrary.remove("11A228ABC4");
```

3.5. Try yourself: Liga de pádel

Try to solve this challenge by using either lists or maps.