Collections

Collection types

In this unit we are going to explain what collections are, and how Java manages dynamic data. We'll see the difference between dynamic and static data, and the different Java types to deal with dynamic collections.

1. Static vs dynamic

In previous units we have worked with elements such as arrays or strings, along with basic Java types (integers, doubles and so on). All these types are called **static** types because their size is established when we declare the corresponding variables, and it never changes along the program execution.

> **NOTE**: remember that, although strings can change their size by adding or removing pieces of text, Java actually creates a brand new string with every operation.

On the other hand, **dynamic** types can change their size along program execution. For instance, if we have a list of books, we can dynamically add or remove books from the list, without having to manually increase or decrease any counter, nor re-assigning the whole collection to a new variable.

The main advantage of static types is the efficiency: as we know the size of the variable when we declare it, Java does not need to care about checking the size every time we use the variable, so programs are faster. However, the main drawback of dealing with these types that they are too rigid, and if we need to increase the previously established size, code can become quite tricky.

2. Main collection types in Java

Java has some different types to deal with dynamic data. Most of them belong to `java.util` package, and can be classified into these main categories:

- **Lists**: an ordered sequence of data, usually indexed. We can think of lists as a dynamic version of arrays: we can access an element by its numeric position, sort them, and many other operations. Besides, there are some specific types of lists in which only certain operations are allowed, such as *stacks* or *queues*.
- **Maps**: also known as *dictionaries* or *hash tables*, they store data that can be retrieved by a given key, just like we do when we look for a word definition in a dictionary: we look for a key (the word) and then we get its definition.
- **Sets**: they store unordered data with the sole premise that there are no repeated elements.
- **Trees**: they store data connected in some sort of parent-child relationship. From one tree node we can go to its children. As we will see in later documents, this provides some useful advantages when dealing with certain information.

Depending on the application we are developing, we may choose one of these types (or some of them). In the following documents we are going to explain how to manage each kind of collection, and the main operations that can be done with them.