Data structures

New Austrian Coding School

Programming - memo

Lists

Lists are a particular kind of collection that have the following properties:

- The elements are **ordered**.
- There can be **repeated** elements.

How to create them

```
List<ELEMENT-TYPE> elements = new ArrayList<>();
```

Examples

```
List<Integer> numbers = new ArrayList<>();
```

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

What to do with them

Trait	Description
add(element)	Appends the specified element to the end of this list.
get(position)	Returns the element at the specified position in this list. Position needs to be a primitive **int** value.
remove(position)	Removes the element at the specified position in this list. Position needs to be a primitive **int** value.
indexOf(element)	Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element. It uses the **equals** method of the element to find it.
remove(element)	Removes the first occurrence of the specified element from this list, if it is present. It uses the **equals** method of the element to find it.
contains(element)	Returns **true** if this list contains the specified element. It uses the **equals** method of the element to find it.
isEmpty()	Returns **true** if this list contains no elements.
size()	Returns the number of elements in this list.

Sets

Sets are a particular kind of collection that have the following properties:

- The elements are **unordered**. Internally they are indexed to find them faster.
- There **cannot** be any **repeated** elements. Otherwise the indexing would not work.

How to create them

```
Set<ELEMENT-TYPE> elements = new HashSet<>();
```

Examples

Set<Integer> numbers = **new** HashSet<>();	Set<String> words = **new** HashSet<>();

What to do with them

Trait	Description
add(element)	Adds the specified element to this set if it is not already present
remove(element)	Removes the specified element from this set if it is present. It uses the **equals** method of the element to find it.
contains(element)	Returns **true** if this set contains the specified element. It uses the **equals** method of the element to find it.
isEmpty()	Returns **true** if this set contains no elements.
size()	Returns the number of elements in this set.

Maps

Maps are a particular kind of data structure that have the following properties:

- They connect keys and values.
- There cannot be repeated keys. There can be repeated values
- The type of the key and the value can be different.

How to create them

Map<ELEMENT-TYPE1, ELEMENT-TYPE2> pairedElements = **new** HashMap<>();

Examples

Map<String, String> dictionary = **new** HashMap<>();	Map<String, Integer> ages = **new** HashMap<>();

What to do with them

Trait	Description
put(key, value)	Associates the specified value with the specified key in this map
get(key)	Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key. It uses the **equals** method of the element to find it.
containsKey(element)	Returns **true** if this map contains a mapping for the specified key. It uses the **equals** method of the element to find it.
isEmpty()	Returns **true** if this map contains no key-value mappings.
size()	Returns the number of key-value mappings in this map.
keySet()	Returns a Set with the keys contained in this map.
values()	Returns a Collection with the values contained in this map.
entrySet()	Returns a Set with the *Entry* mappings contained in this map. An *Entry* object contains one key together with its corresponding value.

Optionals

Optionals are a particular kind of data class that have the following properties:

- They can contain one element.
- They can be empty.
- They warn that the value a method returns could be non-existing.
- They ensure trust because when they are not used it means method always return what they promise.

How to create them

```
Optional<ELEMENT-TYPE> optional = Optional.of(element);
Optional<ELEMENT-TYPE> optional = Optional.empty();
```

Examples

Optional<String> shoeBox = Optional.*of*(**"Shoes"**);	Optional<String> shoeBox = Optional.*empty*();

What to do with them

Trait	Description
isPresent()	Return **true** if there is a value present, otherwise **false**.
get()	If a value is present in this Optional, returns the value, otherwise throws *NoSuchElementException*.

Internet reference

You can find much more information about these data structures at the official Oracle online documentation for Java 8: Lists, Sets, Maps and Optionals.