Técnicas Avançadas de Programação

Desenvolvimento para a *Web* e Dispositivos Móveis

1º Ano, 2º Semestre

Joana Fialho
E-mail: jfialho@estgv.ipv.pt
Nuno Costa
E-mail: ncosta@estgv.ipv.pt
Carlos Simões
E-mail: csimoes@estgv.ipv.pt

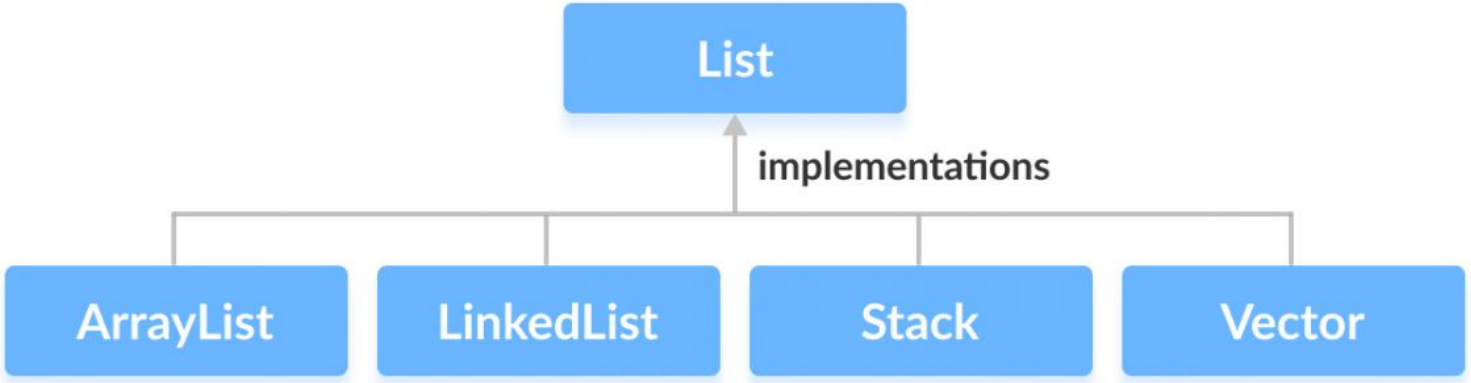
Escola Superior de Tecnologia e Gestão de Viseu
2019-2020

List

Em Java, a **List interface** é uma coleção ordenada (ordered collection) que permite o armazenamento e o acesso sequencial aos seus elementos. Extende a **Collection interface**.

Classes que implementam a **List interface**:

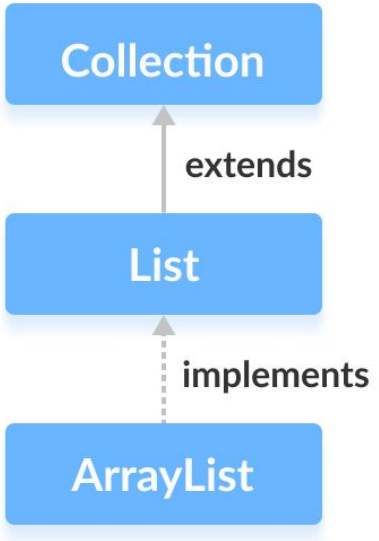
como se trata de uma interface, não se podem "criar" objetos diretamente…




```
// ArrayList implementation of List
List<String> list1 = new ArrayList<>();
```

ArrayList

A classe **ArrayList** (java.util package) permite *arrays* de tamanho adaptável. A diferença entre um *array* normal e um ArrayList em Java é que o tamanho de um array não pode ser alterado, sendo possível adicionar ou remover elementos num ArrayList sempre que desejado (a capacidade de um objeto da classe ArrayList ajusta-se automaticamente quando se adicionam ou removem elementos). A sintaxe apresenta também diferenças:




```
import java.util.ArrayList;

ArrayList<Type> variavel = new ArrayList<Type>();
```

ArrayList

Criar **ArrayList** :

```
// create Integer type arraylist
ArrayList<Integer> Lista1 = new ArrayList<Integer>();

// create String type arraylist
ArrayList<String> Lista2 = new ArrayList<String>();
```

Adicionar elementos a um **ArrayList** : `add()`

```
ArrayList<String> cars = new ArrayList<String>();
cars.add("BMW");
cars.add("Volvo");
cars.add("Ford");
cars.add("Mazda");
```

Mostrar conteúdo de um **ArrayList** :

```
System.out.println("Conteudo de cars: " + cars);
```

ArrayList

Acesso aos elementos de um **ArrayList** :

Para aceder a um elemento de um ArrayList, usamos o método `get()` e a referência ao índice:

```
cars.get(0);
```

Também se podem adicionar elementos usando um índice:

```
cars.add(1,"VW");
```

Alterar um elemento : `set()`

```
cars.set(0, "Opel");
```

Remover elementos de um **ArrayList** :

Remover um elemento - método `remove()` com o índice do elemento:

```
cars.remove(0);
```

Remover todos os elementos - método `clear()`:

```
cars.clear();
```

NOTA: existe ainda o método `removeAll()` mas é menos eficiente.

ArrayList

Exemplo :

```
// Remover elemento com index 2
String str = cars.remove(2);
System.out.println("ArrayList final : " + cars);
System. out.println("Elemento Removido: " + str);
```

Percorrer os elementos de um **ArrayList** :

ciclo `for()`

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

ciclo `for-each`

```
for (String i : cars) {
    System.out.println(i);
}
```

ArrayList

Ordenar Elementos de um **ArrayList** :

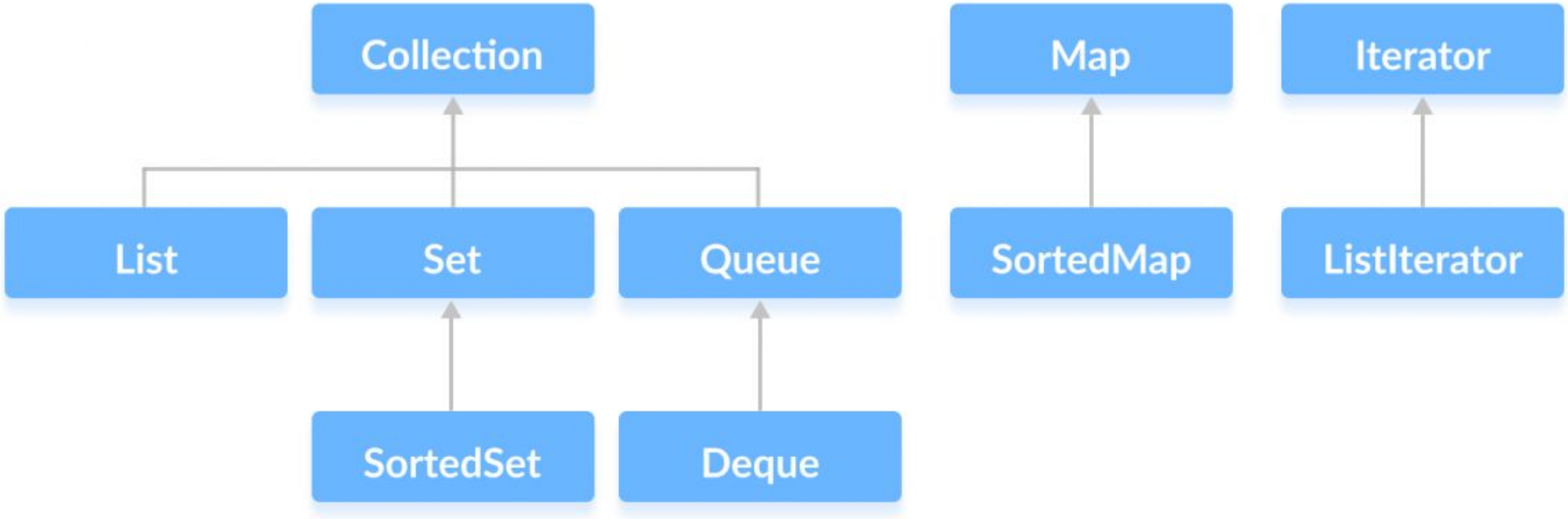
Para ordenar os elementos de um ArrayList pode usar-se o método `sort()` da classe **Collections** (não esquecer de importar a respetiva package).

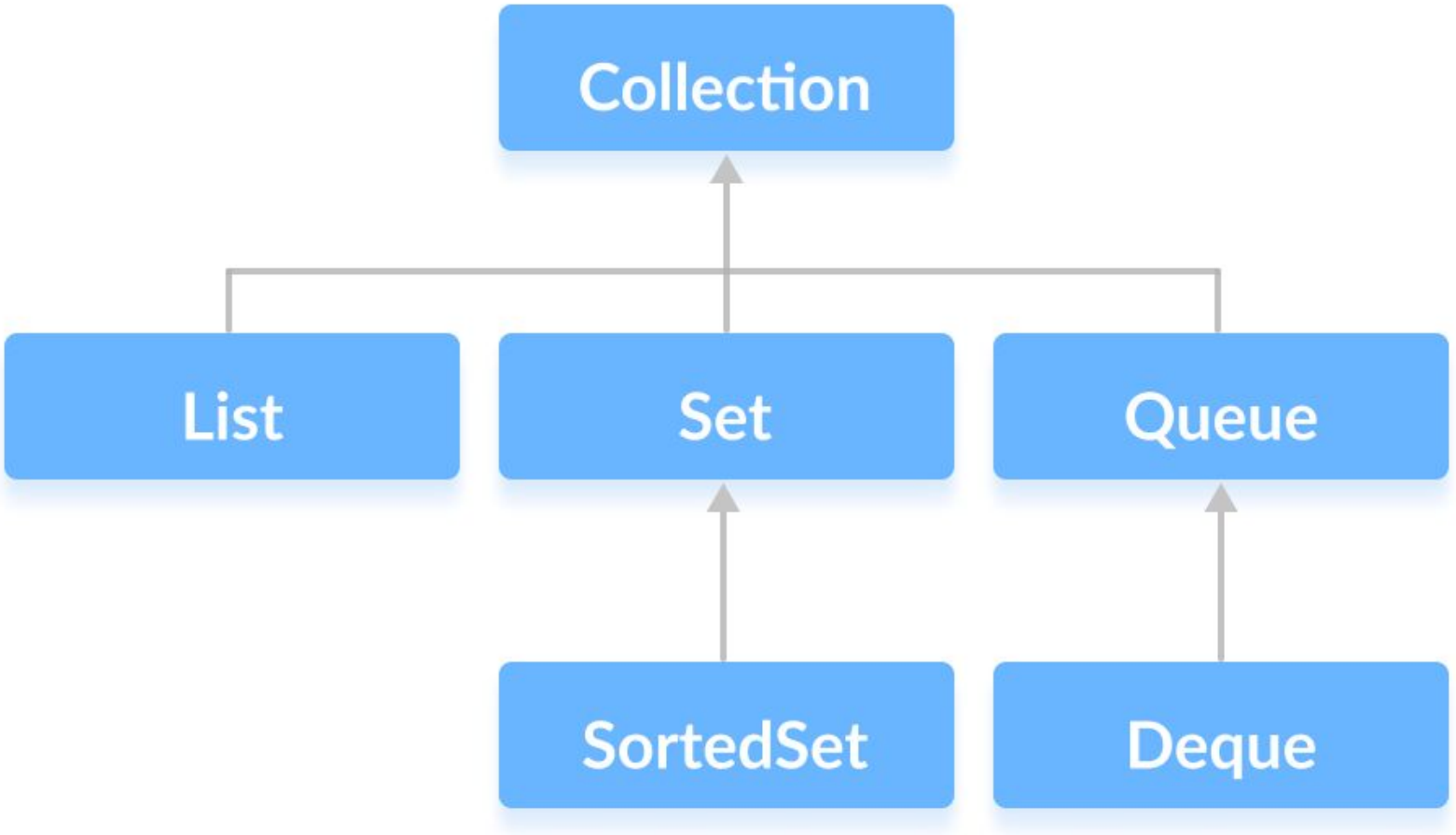
```
import java.util.ArrayList;
import java.util.Collections; //Import Collections class

public class MyClass {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println("ArrayList: " + cars);

    Collections.sort(cars);  // Sort cars
    for (String i : cars) {
      System.out.println(i);
    }
  }
}
```

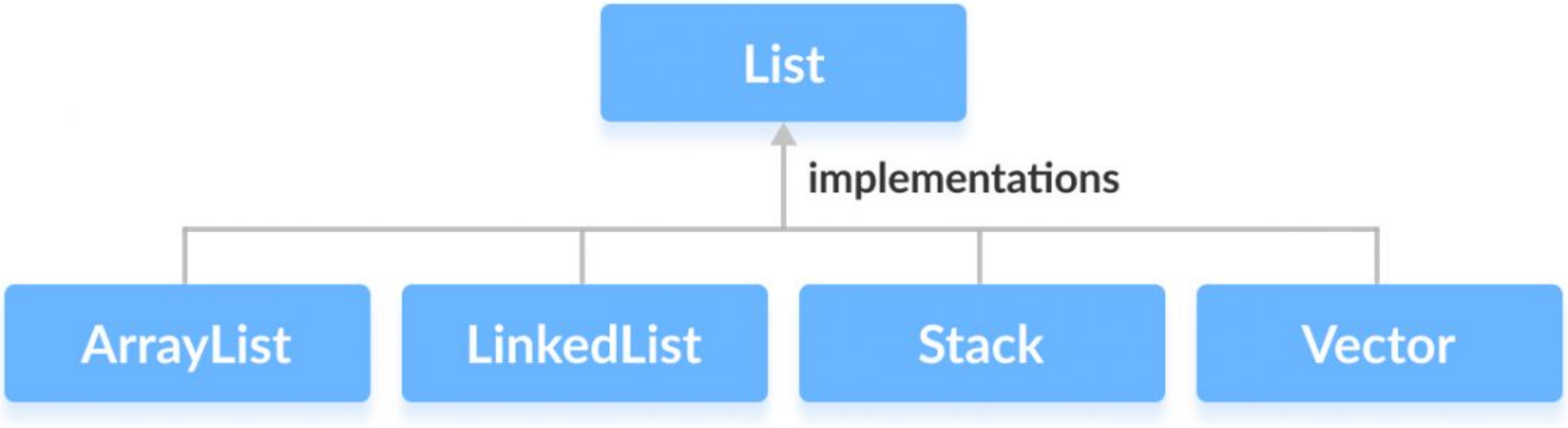
Java Collections Framework




Collection
List
Set
Queue
SortedSet
Deque

In order to use functionalities of the List interface, we can use these classes:

- ArrayList
- LinkedList
- Vector
- Stack

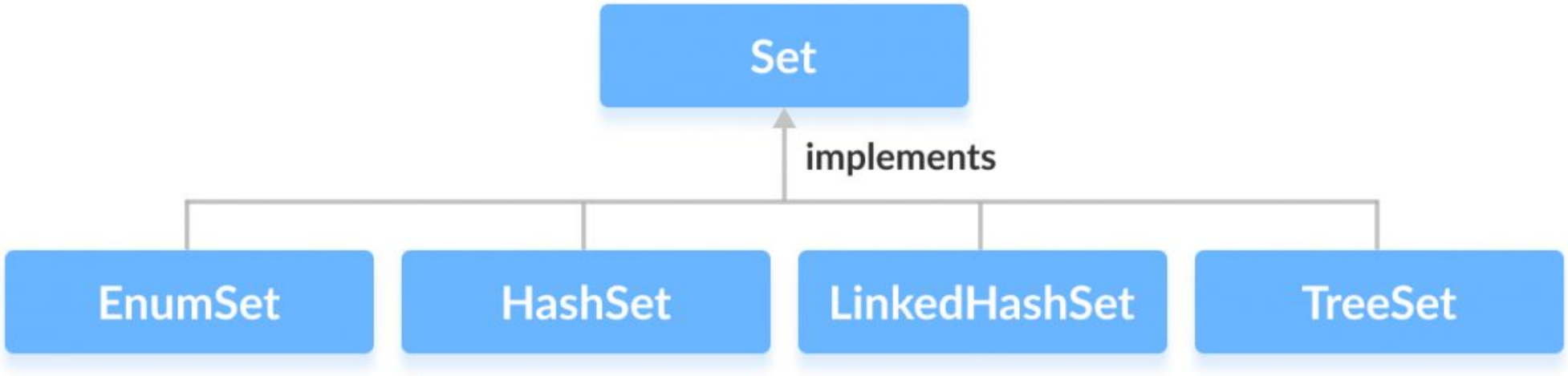


Since Set is an interface, we cannot create objects from it.

In order to use functionalities of the Set interface, we can use these classes:

- HashSet
- LinkedHashSet
- EnumSet
- TreeSet

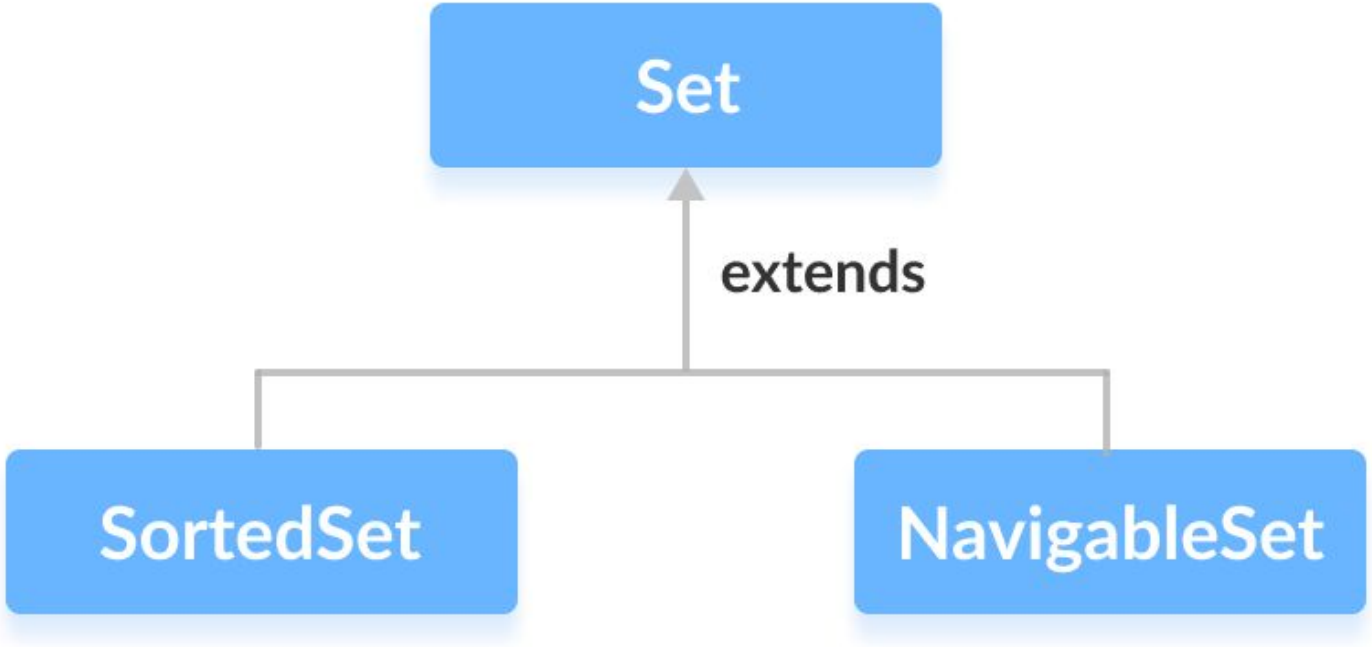
These classes are defined in the Collections framework and implement the Set interface.



The Set interface is also extended by these subinterfaces:

SortedSet

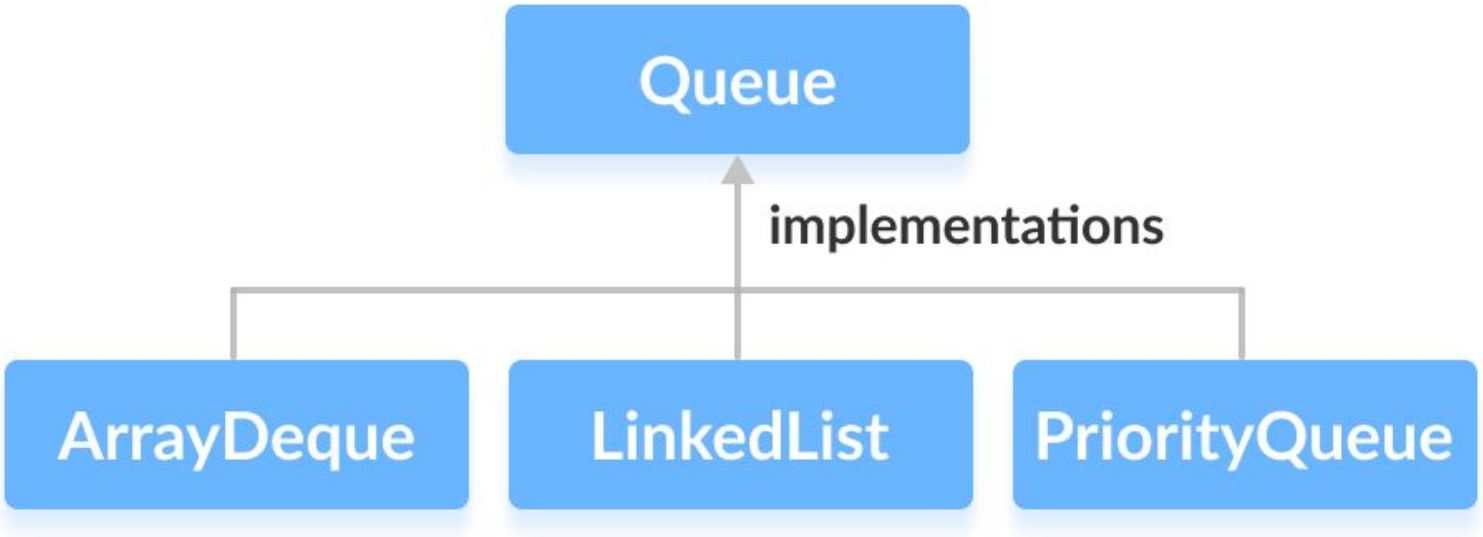
NavigableSet



Since the Queue is an interface, we cannot provide the direct implementation of it.

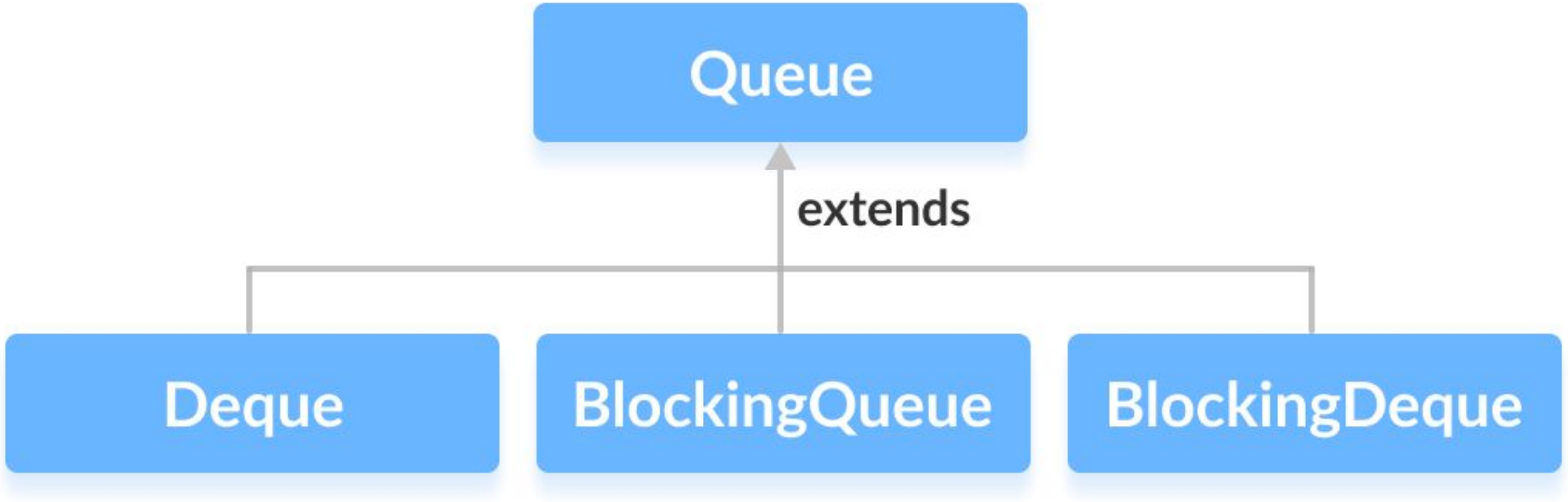
In order to use the functionalities of Queue, we need to use classes that implement it:

- ArrayDeque
- LinkedList
- PriorityQueue



The Queue interface is also extended by various subinterfaces:

- Deque
- BlockingQueue
- BlockingDeque



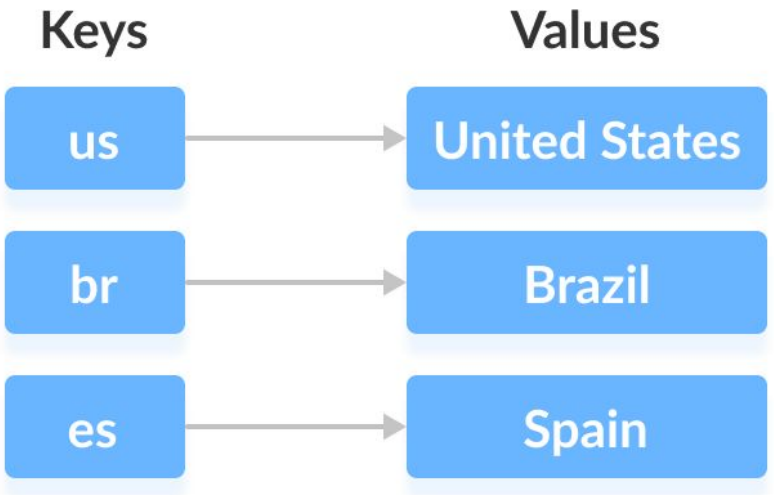
Java Map Interface

The Map interface of the Java collections framework provides the functionality of the map data structure.

It implements the Collection interface.

Java Map Interface

Em Java, os elementos de um **Map** são guardados como pares ***Chave-Valor***. As ***Chaves*** são valores únicos associados a ***Valores***.



É possível aceder e alterar os ***Valores*** usando as ***Chaves*** que lhe estão associadas.

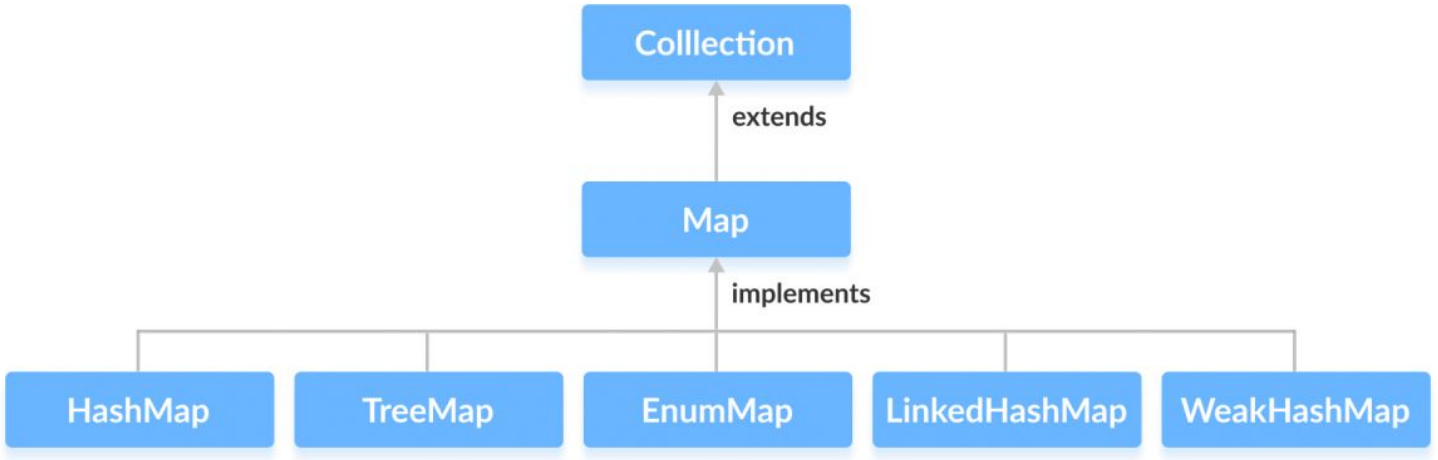
Java Map Interface

São mantidos 3 conjuntos:

- o conjunto das ***Chaves***;
- o conjunto dos ***Valores***;
- o conjunto das associações ***Chave-Valor*** (mapeamento).

Sendo possível aceder às ***Chaves***, ***Valores*** e associações individualmente.

Como o Map é uma interface, não se podem "criar" objetos diretamente… As funcionalidades da Map interface são usadas pelas classes:



Java Map Interface

Como usar a **Map Interface**?

Podemos usar, por exemplo, a classe **`HashMap`** :

```
import java.util.Map;
import java.util.HashMap;

// Map implementation using HashMap
Map<Key, Value> numbers = new HashMap<>();
```

Java HashMap

Um **HashMap** guarda os seus elementos em pares chave-valor, e podemos aceder-lhes por um "índice" de outro tipo (e.g., uma String).

Um objeto é usado como chave (índice) para outro objeto (valor). Podem ser usados tipos diferentes (chaves do tipo string e valores do tipo inteiro), ou do mesmo tipo (chaves do tipo String e valores do tipo String).

Exemplo: Criar um **HashMap** `capitais` com chaves e valores do tipo string:

```
import java.util.HashMap; // import HashMap class

HashMap<String,String> capitais=new HashMap<String,String>();
```

Java HashMap

Adicionar elementos a um **HashMap** : **`put()`**

```
HashMap<String,String> capitais = new HashMap<String,String>();

    // Add keys and values (Country, City)
    capitais.put("England", "London");
    capitais.put("Germany", "Berlin");
    capitais.put("Norway", "Oslo");
    capitais.put("USA", "Washington DC");
    System.out.println(capitais);
```

Aceder aos elementos : **`get()`**

```
capitais.get("England");
```

Java HashMap

Remover um elemento - método **`remove()`** com a chave :

```
capitais.remove("England");
```

Remover todos os elementos - método **`clear()`**

```
capitais.clear();
```

Número de elementos - método **`size()`**

```
capitais.size();
```

Java HashMap

Mostrar conteúdo de um **HashMap**

```
// elementos
System.out.println("HashMap: " + capitais);
// chaves - keySet()
System.out.println("chaves: " + capitais.keySet());
// valores - values()
System.out.println("valores: " + capitais.values());
// mapeamento - entrySet()
System.out.println("Associações chave/valor: " + capitais.entrySet());
```

Output:

```
HashMap: {USA=Washington DC, Norway=Oslo, England=London,
        Germany=Berlin}
chaves: [USA, Norway, England, Germany]
valores: [Washington DC, Oslo, London, Berlin]
Associações chave/valor: [USA=Washington DC, Norway=Oslo,
        England=London, Germany=Berlin]
```

Java HashMap

Percorrer os elementos de um **HashMap**

ciclo `for-each`

exemplo:

```
// mostra chaves
for (String i : capitais.keySet()) {
    System.out.println(i);
}

// mostra valores
for (String i : capitais.values()) {
    System.out.println(i);
}

// mostra chaves e valores
for (String i : capitais.keySet()) {
    System.out.println("key: " + i + " value: " + capitais.get(i));
}
```

Java HashMap

`put()` - inserts the specified key/value mapping to the map
`putAll()` - inserts all the entries from specified map to this map
`putIfAbsent()` - inserts the specified key/value mapping to the map if the specified key is not present in the map
-
`entrySet()` - returns a set of all the key/value mapping of the map
`keySet()` - returns a set of all the keys of the map
`values()` - returns a set of all the values of the map
-
`remove(key)` - returns and removes the entry associated with the specified key from the map
`remove(key, value)` - removes the entry from the map only if the specified key mapped to the specified value and return a boolean value
-
`replace(key, value)` - replaces the value associated with the specified key by a new value
`replace(key, old, new)` - replaces the old value with the new value only if old value is already associated with the specified key
`replaceAll(function)` - replaces each value of the map with the result of the specified function

Java HashMap

`clear()` - Removes all the entries from the map

`containsKey()` - Checks if the map contains the specified key and returns a boolean value

`containsValue()` - Checks if the map contains the specified value and returns a boolean value

`size()` - Returns the size of the map

`isEmpty()` - Checks if the map is empty and returns a boolean value

-