Java
La généricité

Objectifs

- Découvrir la notion de généricité
- Savoir utiliser les classes génériques proposées par Java
- Être capable de créer des méthodes génériques et des classes génériques

Les Collections

Il existe des alternatives aux tableaux en Java.

Tous les **types** dédiés à accueillir **une multitude de valeur** sont appelés des **Collections**.

On distingue 4 formes de Collections :

- Les listes (List) utilisés partout
- Les ensembles (Set) moins
- les dictionnaires (Map) très utilisé
- les files (Queue) moins

Les Collections

- Les listes (List)

```
public static void main(String[] args) {

    Personne pers1 = new Personne("Cassin", "Etienne");
    Personne pers2 = new Personne("Berger", "Michel");

    List<Personne> personnes = new ArrayList<Personne>();
    personnes.add(pers1);
    personnes.add(pers2);

    for (Personne current : personnes) {
        System.out.println(current.toString());
    }
}
```

On ne fait jamais (ou presque de tableau) toujours des listes

dans une liste, tous les éléments sont de même type.

Les listes ont des tailles dynamiques (alors que les tableaux sont de taille fixe)

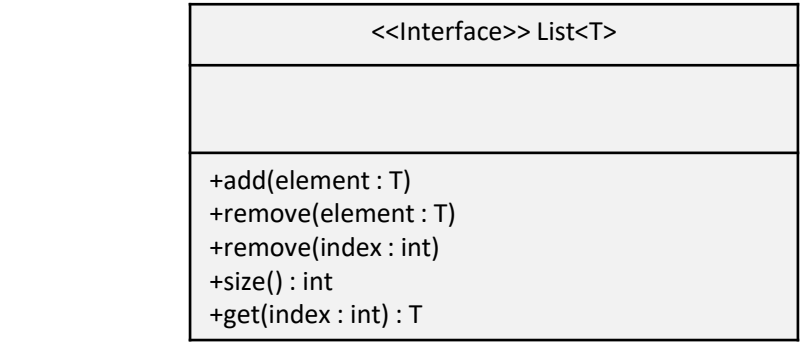
Les Collections

- Les listes (List)

liste=ArrayList

<T> : type générique

```
public static void main(String[] args) {

    Personne pers1 = new Personne("Cassin", "Etienne");
    Personne pers2 = new Personne("Berger", "Michel");

    List<Personne> personnes = new ArrayList<Personne>();
    personnes.add(pers1);
    personnes.add(pers2);

    for (Personne current : personnes) {
        System.out.println(current.toString());
    }
}
```



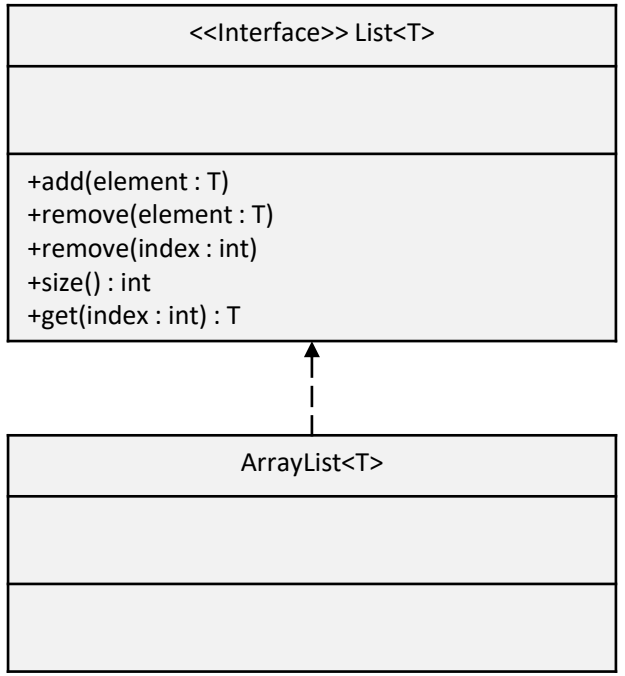
size (nombre d'éléments de la liste)

maListe.get(100) => retourne l'élément n°100 du tableau (commence à 0)

Les Collections

- Les listes (List)

```
public static void main(String[] args) {

    Personne pers1 = new Personne("Cassin", "Etienne");
    Personne pers2 = new Personne("Berger", "Michel");

    List<Personne> personnes = new ArrayList<Personne>();
    personnes.add(pers1);
    personnes.add(pers2);

    for (Personne current : personnes) {
        System.out.println(current.toString());
    }
}
```

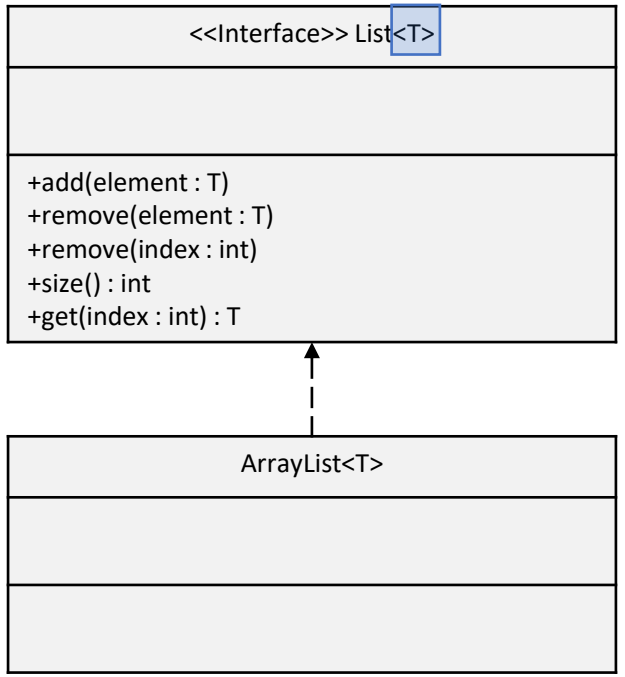


Les Collections

- Les listes (List)

```
public static void main(String[] args) {

    Personne pers1 = new Personne("Cassin", "Etienne");
    Personne pers2 = new Personne("Berger", "Michel");

    List<Personne> personnes = new ArrayList<Personne>();
    personnes.add(pers1);
    personnes.add(pers2);

    for (Personne current : personnes) {
        System.out.println(current.toString());
    }
}
```

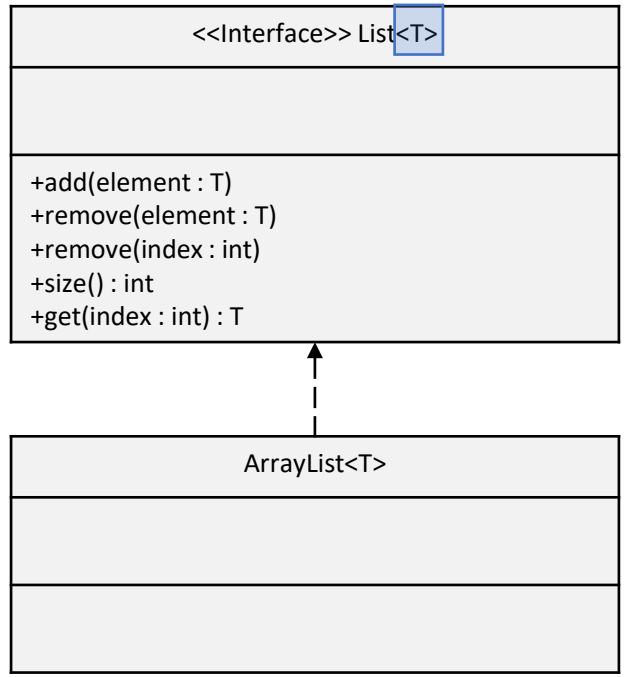
L'opérateur <T> indique un type générique.



Les Collections

- Les listes (List)

L'opérateur <T> indique un type générique.




```
public static void main(String[] args) {

    Personne pers1 = new Personne("Cassin", "Etienne");
    Personne pers2 = new Personne("Berger", "Michel");

    List<Personne> personnes = new ArrayList<Personne>();
    personnes.add(pers1);
    personnes.add(pers2);

    for (Personne current : personnes) {
        System.out.println(current.toString());
    }
}
```

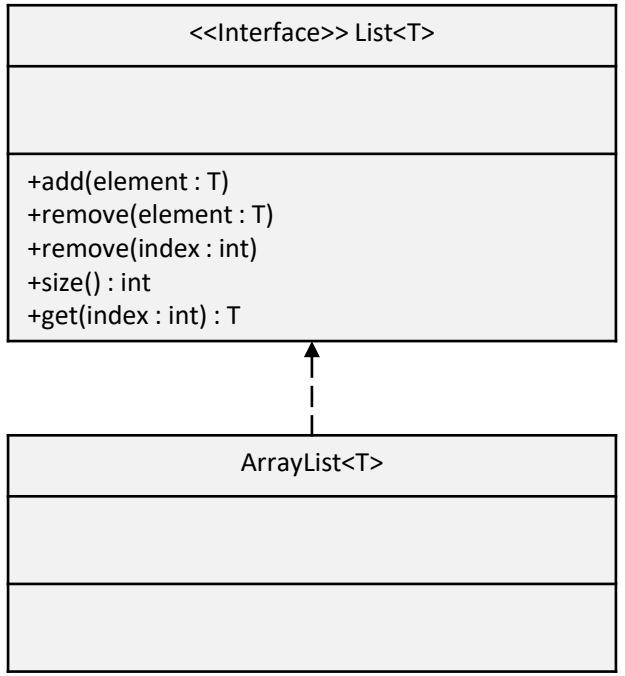
C'est à l'instanciation de la classe que nous préciserons la nature de ce type

Ici une ArrayList

Les Collections

- Les listes (List)

```
public static void main(String[] args) {

    Personne pers1 = new Personne("Cassin", "Etienne");
    Personne pers2 = new Personne("Berger", "Michel");

    List<Personne> personnes = new ArrayList<Personne>();
    personnes.add(pers1);
    personnes.add(pers2);

    for (Personne current : personnes) {
        System.out.println(current.toString());
    }
}
```



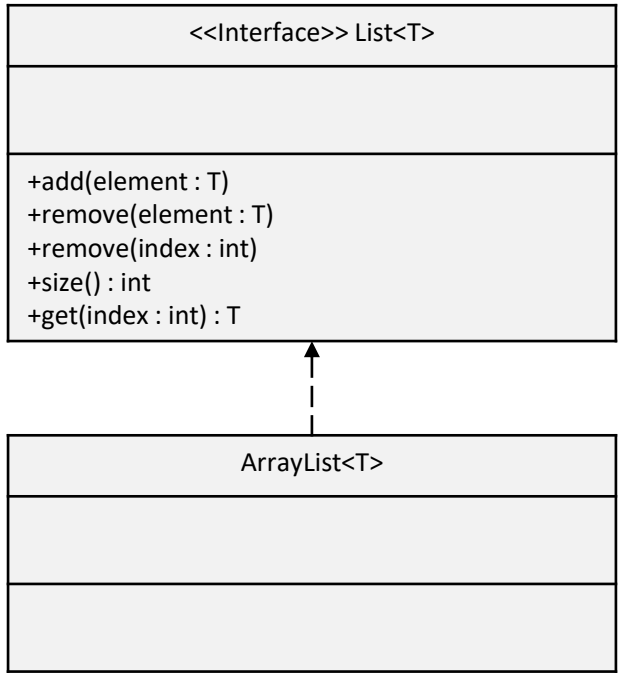
La List est le choix le plus répandu lorsqu'on doit manipuler des collections d'éléments

Les Collections

Quelques détails…

- Les listes (List)

```
public static void main(String[] args) {

    Personne pers1 = new Personne("Cassin", "Etienne");
    Personne pers2 = new Personne("Berger", "Michel");

    List<Personne> personnes = new ArrayList<Personne>();
    personnes.add(pers1);
    personnes.add(pers2);

    for (Personne current : personnes) {
        System.out.println(current.toString());
    }
}
```

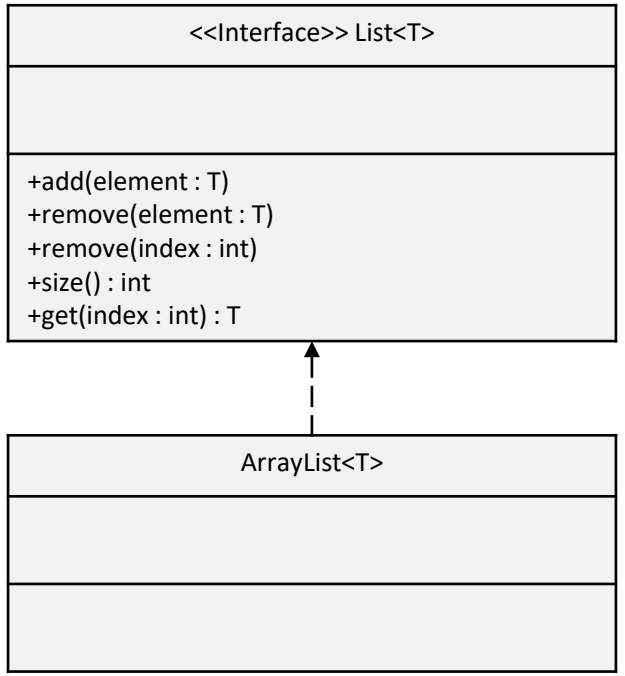


Les Collections

Quelques détails…

- Les listes (List)

```
public static void main(String[] args) {

    Personne pers1 = new Personne("Cassin", "Etienne");
    Personne pers2 = new Personne("Berger", "Michel");

    List<Personne> personnes = new ArrayList<>();
    personnes.add(pers1);
    personnes.add(pers2);

    for (Personne current : personnes) {
        System.out.println(current.toString());
    }
}
```



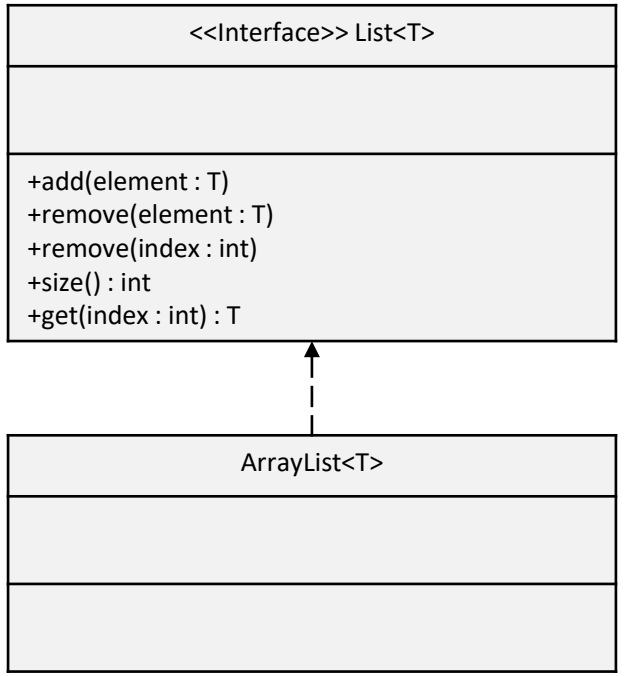
Si mon type est précisé à gauche de l'égalité, je n'ai pas besoin de le repréciser à droite. *C'est l'inférence de type.*

Les Collections

Quelques détails…

- Les listes (List)

```
public static void main(String[] args) {
    List<int> desEntiers = new ArrayList<>();

    for (int current : desEntiers) {
        System.out.println(current);
    }
}
```



Impossible d'utiliser les types primitifs avec le type générique.

Les Collections

Quelques détails…

- Les listes (List)

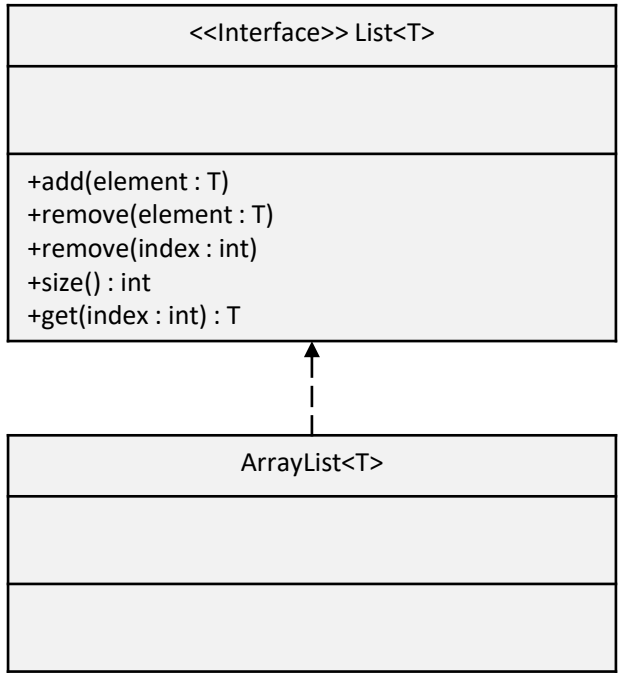
Array:Crée un tableau qui contient nos éléments
.Add() crée un tableau avec une case en plus...

```
public static void main(String[] args) {
    List<int> desEntiers = new ArrayList<>();

    for (int current : desEntiers) {
        System.out.println(current);
    }
}
```

On a le droit de mettre des Integer partout et d'oublier le type int primitif

```
public static void main(String[] args) {
    List<Integer> desEntiers = new ArrayList<>();

    for (int current : desEntiers) {
        System.out.println(current);
    }
}
```

une liste ne peut gérer que des classe (pas des types primitifs)



Impossible d'utiliser les types primitifs avec le type générique.
On passera par des wrappers.

Les Collections

- Les classes dites « wrapper »

String est déjà un objet donc n'est pas un wrapper (un tableau de char)

Type de base	Classe wrapper
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Les Collections

- Les ensembles (Set)

```
public static void main(String[] args) {
    Personne pers1 = new Personne("Sebastien", "Patrick");
    Personne pers2 = new Personne("Kamakawiwo'ole", "Israel");

    Set<Personne> personnesUniques = new HashSet<>();

    personnesUniques.add(pers1);
    personnesUniques.add(pers2);
    personnesUniques.add(pers2);

    for (Personne current : personnesUniques) {
        System.out.println(current);
    }
}
```

Set est une interface
HashSet est une classe concrête

Les Collections

- Les ensembles (Set)

```
public static void main(String[] args) {
    Personne pers1 = new Personne("Sebastien", "Patrick");
    Personne pers2 = new Personne("Kamakawiwo'ole", "Israel");

    Set<Personne> personnesUniques = new HashSet<>();

    personnesUniques.add(pers1);
    personnesUniques.add(pers2);
    personnesUniques.add(pers2);

    for (Personne current : personnesUniques) {
        System.out.println(current);
    }
}
```

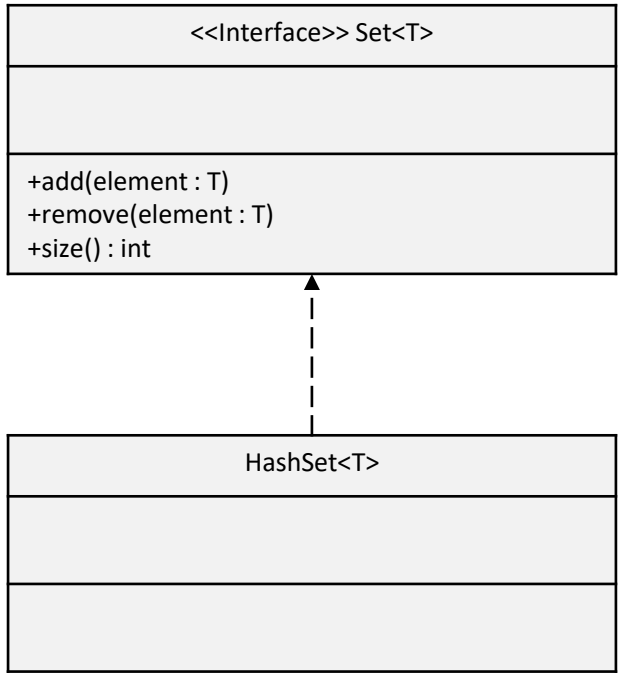
Set est un ensemble: donc pas de doublon, pas d'ordre, pas d'index

```
nom=Kamakawiwo'ole, prenom=Israel
nom=Sebastien, prenom=Patrick
```

Les Collections

- Les ensembles (Set)

```
public static void main(String[] args) {
    Personne pers1 = new Personne("Sebastien", "Patrick");
    Personne pers2 = new Personne("Kamakawiwo'ole", "Israel");

    Set<Personne> personnesUniques = new HashSet<>();

    personnesUniques.add(pers1);
    personnesUniques.add(pers2);
    personnesUniques.add(pers2);

    for (Personne current : personnesUniques) {
        System.out.println(current);
    }
}
```

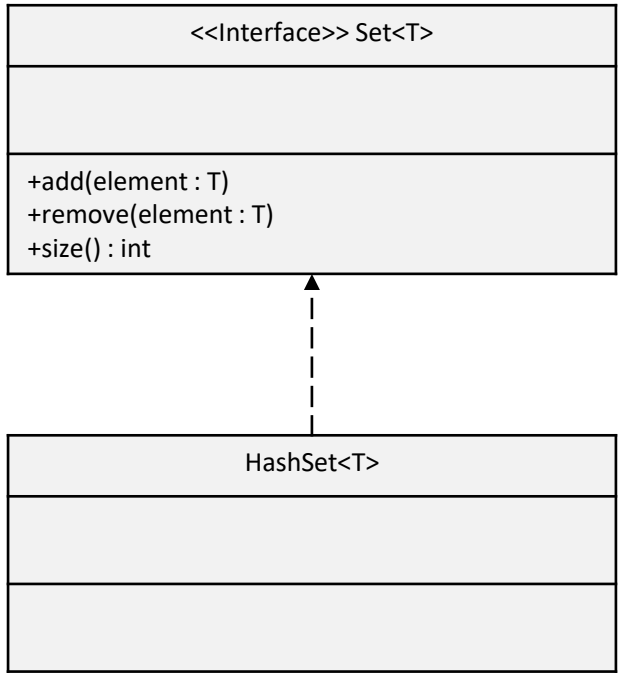



```
nom=Kamakawiwo'ole, prenom=Israel
nom=Sebastien, prenom=Patrick
```

Les Collections

- Les ensembles (Set)

```
public static void main(String[] args) {
    Personne pers1 = new Personne("Sebastien", "Patrick");
    Personne pers2 = new Personne("Kamakawiwo'ole", "Israel");

    Set<Personne> personnesUniques = new HashSet<>();

    personnesUniques.add(pers1);
    personnesUniques.add(pers2);
    personnesUniques.add(pers2);

    for (Personne current : personnesUniques) {
        System.out.println(current);
    }
}
```



nom=K
nom=S

Le Set est le choix le plus répandu lorsqu'on doit manipuler des collections d'éléments *sans doublon et sans notion d'ordre*

Les Collections

- les dictionnaires (Map)

```
public static void main(String[] args) {
    Map<String, String> acronymes = new HashMap<>();
    acronymes.put("JDK", "Java Development Kit");
    acronymes.put("POO", "Programmation Orientee Objet");
    acronymes.put("BO", "Bean Object");
    acronymes.put("ETC", "Et caetera");

    System.out.println(acronymes.get("JDK"));
    System.out.println(acronymes.get("ETC"));
}
```

Ex: Dictionnaire papier

Clé unique

ici on a put pour ajouter au lieu de add

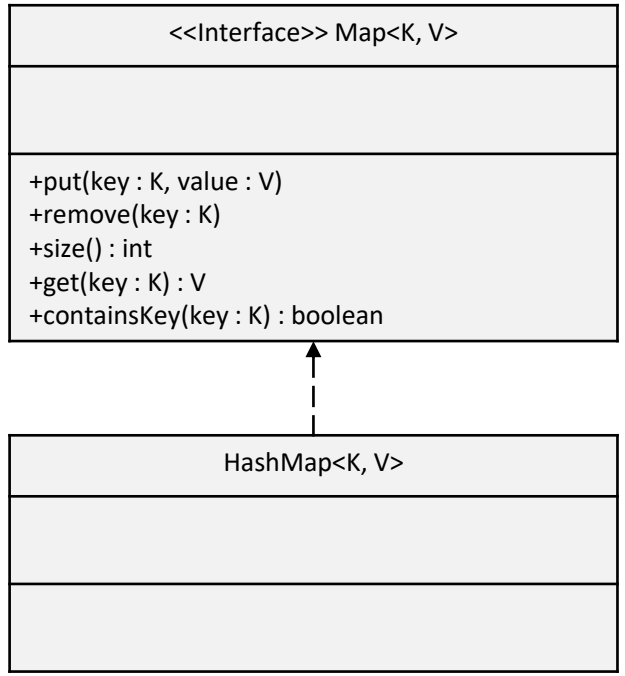
```
Java Development Kit
Et caetera
```

Les Collections

- les dictionnaires (Map)

<K,V> = key,value

```
public static void main(String[] args) {
    Map<String, String> acronymes = new HashMap<>();
    acronymes.put("JDK", "Java Development Kit");
    acronymes.put("POO", "Programmation Orientee Objet");
    acronymes.put("BO", "Bean Object");
    acronymes.put("ETC", "Et caetera");

    System.out.println(acronymes.get("JDK"));
    System.out.println(acronymes.get("ETC"));
}
```



peu adapté aux boucles

```
Java Development Kit
Et caetera
```

On pourrait avoir pour une liste de medecin, une liste de creneaux dsipo
Map<Medecin, List<Creneau>>
Ex: objet Hopital

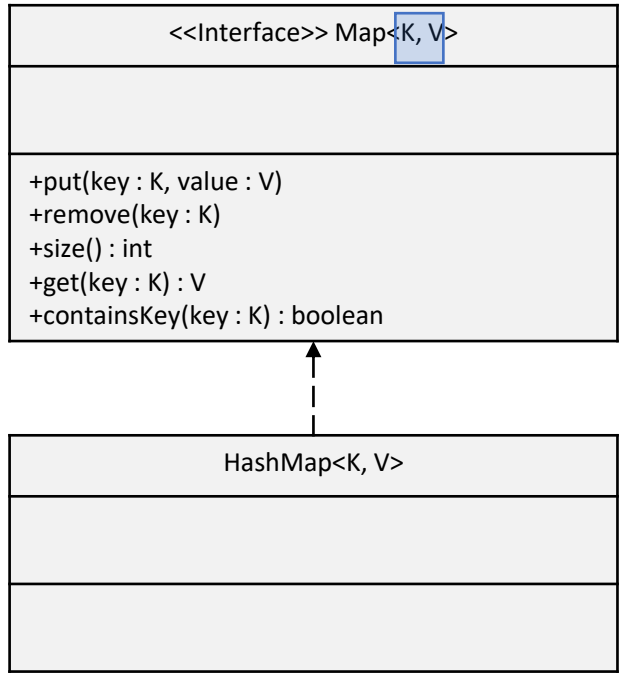
Les Collections

- les dictionnaires (Map)

```
public static void main(String[] args) {
    Map<String, String> acronymes = new HashMap<>();
    acronymes.put("JDK", "Java Development Kit");
    acronymes.put("POO", "Programmation Orientee Objet");
    acronymes.put("BO", "Bean Object");
    acronymes.put("ETC", "Et caetera");

    System.out.println(acronymes.get("JDK"));
    System.out.println(acronymes.get("ETC"));
}
```

```
Java Development Kit
Et caetera
```

Ici, on a 2 types génériques : K (Key) et V (Value).



Les Collections

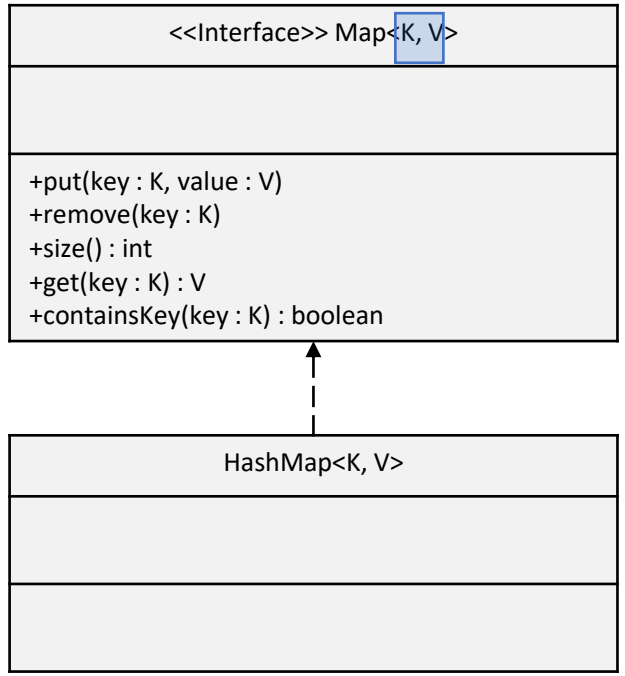
- les dictionnaires (Map)

```
public static void main(String[] args) {
    Map<String, String> acronymes = new HashMap<>();
    acronymes.put("JDK", "Java Development Kit");
    acronymes.put("POO", "Programmation Orientee Objet");
    acronymes.put("BO", "Bean Object");
    acronymes.put("ETC", "Et caetera");

    System.out.println(acronymes.get("JDK"));
    System.out.println(acronymes.get("ETC"));
}
```

L'intérêt du dictionnaire est que la recherche est très rapide (un dictionnaire sait où se trouve la 1ere lettre) exemple le E et trouve rapidement ETC (mieux que dichotomie)

```
Java Development Kit
Et
```

Ici, on a 2 types génériques : K (Key) et V (Value).



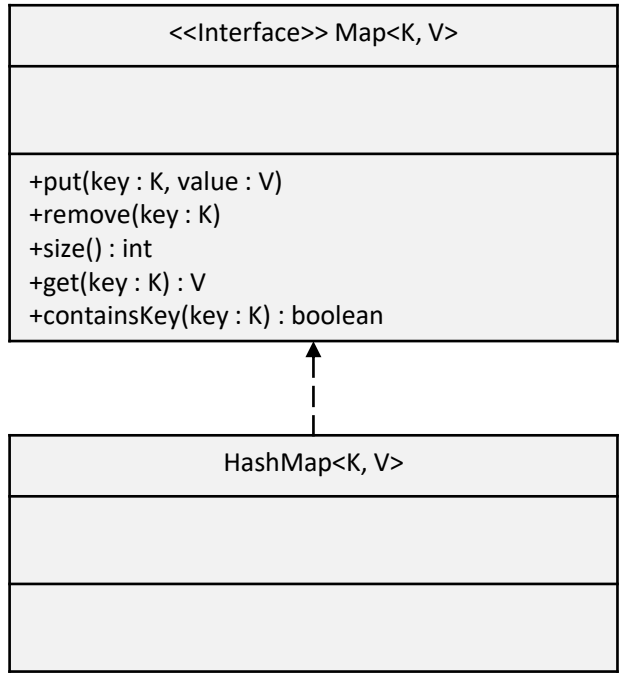
Le Set est le choix le plus répandu lorsqu'on doit manipuler des collections d'éléments *sous la forme de paire clé / valeur*

Les Collections

Un autre exemple

- les dictionnaires (Map)

```
public static void main(String[] args) {
    Map<Adresse, Coordonnees> carte = new HashMap<>();

    Adresse a1 = new Adresse(6, "Rue Charles de Gaulle", "Nantes");
    Adresse a2 = new Adresse(102, "Bd du Massacre", "St Herblain");
    Adresse a3 = new Adresse(1, "Rue du Calvaire", "Nantes");

    Coordonnees c1 = new Coordonnees(47.3008178,-1.5551859);
    Coordonnees c2 = new Coordonnees(47.2321462,-1.5943157);
    Coordonnees c3 = new Coordonnees(47.2155958,-1.5616703);

    carte.put(a1, c1);
    carte.put(a2, c2);
    carte.put(a3, c3);
}
```



Le Set est le choix le plus répandu lorsqu'on doit manipuler des collections d'éléments *sous la forme de paire clé / valeur*

Démo
Créer une LinkedList

Atelier 7
Améliorations

Java
La généricité