ISEN ALL IS DIGITAL! OUEST | yncréa

Java

Collections

Introduction

- Le langage Java contient de base des conteneurs de données génériques et prêts à l'emploi appelés ***collections***.
- Les collections se divisent en plusieurs types. Pour chaque type, on a :
  - Une interface d'utilisation abstraite (une liste d'opérations)
  - Des implémentations concrètes, avec des caractéristiques différentes (performances, données triées ou non, etc…)
- Dans ce cours, nous allons focaliser sur les types suivants :
  - Les listes
  - Les ensembles
  - Les tableaux associatifs

Les listes

- Dans une liste, les données sont ordonnées et indicées.
- Une liste peut contenir des doublons.

Indices	0	1	2	3	4	5	6
Valeurs	“Cthulhu”	“Dagon”	“Azathoth”	“Hastur”	“Ulthar”	“Nyarlathotep”	“Ubbo-Sathla”

- En Java, le concept de liste est modélisé par l’interface List.
- Il existe deux implémentations de cette interface :
  - ArrayList, basé sur des tableaux
  - LinkedList, basé sur les listes chaînées

Création d'une liste

- Les listes peuvent contenir des instances de n'importe quelle classe. Ceci est matérialisé par un ***type générique*** <>.
- Pour créer une liste, on précise la classe des éléments de la liste entre <>, qui sera la même pour toutes les cases.
- Exemple : liste (chaînée) de chaînes de caractères :

```
LinkedList<String> anciensDieux = new LinkedList<String>();
```

- Si vous voulez mettre un type primitif dans une liste, il faut indiquer la classe enveloppante associée à ce type.

```
LinkedList<int> notes;
```

✗ **Erreur**

```
LinkedList<Integer> notes;
```

✓ **OK**

Exemple de manipulation d'une liste

```
LinkedList<String> grandsAnciens = new LinkedList<String>();
// remplissage de la liste
grandsAnciens.add("Cthulhu");
grandsAnciens.add("Nyarlathotep");
grandsAnciens.add("Dagon");
// affichage des éléments de la liste
for(String nom : grandsAnciens){
    System.out.println("Ftaghn " + nom);
}
```

Exemple de manipulation d'une liste

```
LinkedList<String> grandsAnciens = new LinkedList<>();
// remplissage de la liste
grandsAnciens.add("Cthulhu");
grandsAnciens.add("Nyarlathotep");
grandsAnciens.add("Dagon");
// affichage des éléments de la liste
for(String nom : grandsAnciens){
    System.out.println("Ftaghn " + nom);
}
```

On peut omettre le type ici, il est inféré par le compilateur à partir du type de la variable.

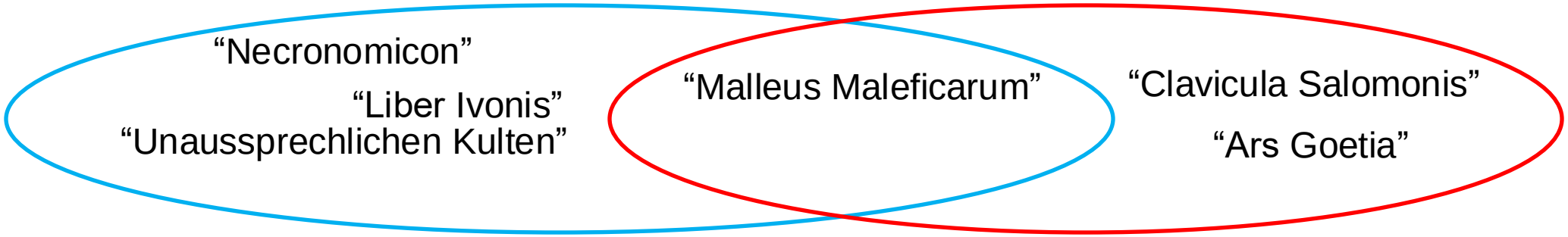
ArrayList ou LinkedList ?

- Pour choisir entre ArrayList et LinkedList, vous devez identifier les opérations les plus fréquentes sur votre liste, et comparer les complexités temporelles associées
- Par exemple :

Opération	Complexité temporelle	
	LinkedList	ArrayList
Accès à un élément quelconque	$O(n)$	$O(1)$
Ajout / suppression d'un élément	$O(1)$	$O(n)$

Les ensembles : Set

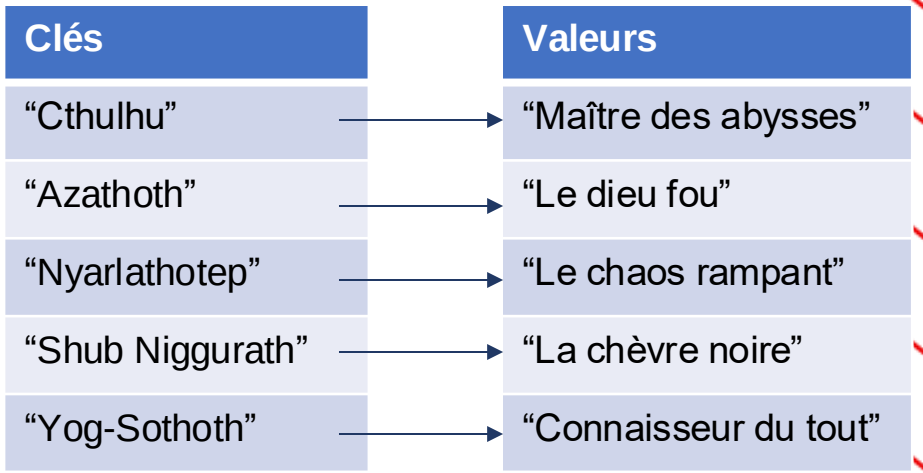
- L'interface Set modélise en Java le concept d'ensemble mathématique
  - Les valeurs d'un Set sont uniques
  - Set définit des opérations ensemblistes comme l'union, l'intersection, etc.



- Il existe deux implémentations de Set en Java :
  - HashSet, dans laquelle les données ne sont pas triées
  - TreeSet, dans laquelle les données sont triées

Les tableaux associatifs : Map

- L'interface Map modélise le concept de tableau associatif en Java (aussi appelés dictionnaires ou tables de hachage)
- Dans une Map, les valeurs sont identifiées par une clé unique. À noter que n'importe quel type peut servir de clé.
- Il existe deux implémentations de Map en Java :
  - HashMap, dans laquelle les données ne sont pas triées
  - TreeMap, dans laquelle les données sont triées



Exemple de manipulation d'une map

```
HashMap<String, String> greatOldOnes = new HashMap<>();
// Remplissage de la HashMap avec des paires clé-valeur
greatOldOnes.put("Cthulhu", "Maître des abysses");
greatOldOnes.put("Nyarlathotep", "Le Chaos Rampant");
greatOldOnes.put("Azathoth", "Le Dieu fou");
greatOldOnes.put("Yog-Sothoth", "Connaisseur du tout");
greatOldOnes.put("Shub-Niggurath", "La Chèvre Noire");
greatOldOnes.put("Dagon", "Seigneur des Profondeurs");
// Affichage d'une description pour un Grand Ancien particulier
String ancientOne = "Cthulhu";
if (greatOldOnes.containsKey(ancientOne))
    System.out.println(ancientOne+":"+greatOldOnes.get(ancientOne));
// Récupération de tous les noms des grands anciens
for (String name : greatOldOnes.keySet())
    System.out.println(name);
```

HashMap ou TreeMap?

- Pour choisir entre HashMap et TreeMap , vous devez identifier les opérations les plus fréquentes sur votre liste, et comparer les complexités temporelles associées
- Par exemple :

Opération	Complexité temporelle	
	HashMap	TreeMap
Recherche	O(1)	O(log n)
Ajout/Suppression	O(1)	O(log n)
Parcours	O(n)	O(n) mais trié