

A3P/IPO 2023/2024

Révisions

HashMap

- HashMap \approx « tableau associatif » qui contient un ensemble d'associations clé \rightarrow valeur
- Classe du JDK \Rightarrow `import java.util.HashMap;`
- Plus général qu'un tableau classique qui impose le type `int` pour les indices (clé, valeur) \Rightarrow 2 types objets
- Donc **pas** `typeValeurs[]` mais
`HashMap<typeClés, typeValeurs> vMaHM;`
- `vMaHM.put (maClé, maValeur);`
- `quelleValeur = vMaHM.get (maClé);`

HashMap : exemple

- Plutôt que de stocker les sorties d'une pièce dans un tableau de 4 Rooms :
tab[0] serait le nord ? **tab[2] l'est ou l'ouest ?**
- Associons chaque sortie Room à la direction "North" ou "South" ...
- Donc dans la classe Room : un attribut
`HashMap<String, Room> aSorties;`
- `aSorties.put("North", vCuisine);`
- `quelleRoom = aSorties.get("North");`
- `null` si la clé n'est pas présente

Set, keySet, for each, remove

- `keySet` est une fonction de `HashMap` qui retourne l'ensemble des clés :

```
?vMesClés? = vMaHM.keySet();
```

- Donc pour `aSorties`, un ensemble de `String`, classe du JDK : `Set<String> vMesClés;`

- On peut facilement parcourir tous ses éléments :

```
for ( String vS : vMesClés ) {  
    S.o.p( vMaHM.get(vS).getDesc() );  
} // boucle for each
```

- On peut aussi écrire `vMaHM.remove(vS);`

Révisions

- Résumés de cours
- Partie cours des TP et TD, + sujets
- **A SAVOIR EXPLIQUER** du projet
- Finir les TP
- Finir les TD et les passer sur machine

Séquence 1

- **Les classes** : modèle pour construire des objets, = attributs (privés) + méthodes (publiques)
- **Les attributs** : nom/type/valeur, types primitifs (valeur) et objets (référence)
- **Les méthodes** : fonction (*return type*), procédure (void), constructeur (spécial, rôle)
- **minuscules/Majuscules** : règles de nommage standard
- **Les variables** : attribut/paramètre/variable locale
- **this** : objet courant

Séquence 2

- **Les ordres** : attributs, constructeurs, méthodes, instructions, paramètres
- **Les commentaires** : `//` fin de ligne, `/**` javadoc, multilignes, `@ */`
- **Définition de méthode** : signature + corps, fonction (typeRetour, `return`) et procédure (`void`)
- **Les constructeurs** : plusieurs, par défaut, naturel, quelconque
- **Duplication de code** : constructeur appelle un autre : `this()` ; en 1^{ère} instruction
- **Création d'objets** : déclaration/création, instance/instancier, `new`, paramètres
- **Accesseurs & modificateurs** : rôles, fonction/procédure, paramètre ou pas
- **Expressions et instructions** : valeur ou pas, types, affectation/comparaison, terminaison/affichage
- **Tests** : `if`, `if else`, parenthèses, une ou plusieurs instructions
- **Récursivité** : méthode qui s'appelle elle-même, relation de récurrence, test d'arrêt, paramètre qui change ; `return` ;

Séquence 3 (1/2)

- **Appel de méthode** : `objet.méthode(paramètres)`,
`expression/instruction`, `new Classe(paramètres)`
- `return expression;` `return;`
- `null`, `NullPointerException`
- `if (condition) return true; else return false;`
dans une fonction booléenne
- `if non indépendants, else`
- `x=y;` `x == y` `x.equals(y)`
- `System.out.println` `System.out.print`
- `private`, protection autour de la classe et non de chaque objet

Séquence 3 (2/2)

- Tableaux
- Les boucles `for` et `while`
- Limitations du type `int` et du type `double`
- Type primitif \neq Type objet (*reference type*)
- Expressions : entières, réelles, booléennes, `String`, `Cercle`, ...
- Instruction \neq expression
- Méthodes statiques = méthodes de classe (et non méthodes d'instance)
- Comment écrire une méthode d'après un énoncé ?

Séquence 4

- **Attributs statiques** = attributs de classe = variables de classe (partagés => en un seul exemplaire).
- **Constantes** => attributs statiques
`private static final int NOM_EN_MAJ = 1000;`
- **héritage** entre classes, seulement si « est une sorte de »
- **signature** de la classe suivie de `extends ClasseMère`
- **type déclaré / type constaté** (compilation / exécution)
- Pourquoi `Animal vA1 = new Oiseau();`
et pas `Oiseau vO1 = new Animal();` ?
- **Redéfinition** d'une méthode dans une sous-classe
- **Object**, mère de toutes les classes =>
`extends Object` automatique, pourtant un seul `extends` !?
- **Constructeur naturel** dans une sous-classe :
combien de paramètres ?

Séquence 5

- **vocabulaire** : Polymorphisme.
- **classe abstraite** => non instanciable
- constructeurs **inutiles** ?
- **méthode abstraite** = signature sans corps => non exécutable
- méthode non abstraite **peut appeler** une méthode abstraite
- **interface** => ni attribut ni constructeur,
que des méthodes abstraites
- **implements** n'est pas incompatible avec
`extends UneSuperClasse`
- **Exemples** d'interfaces dans le JDK :
- Il est possible de **créer** ses propres interfaces
- **Héritage d'interface** (multiple !)
- Nouveau mot java : `unObjet instanceof UnType`

Séquence 6

- **Framework Collections** = ensemble de classes abstraites ou non, d'interfaces, et d'algorithmes.
- >tableaux : - **seulement des objets** + extensible + propriétés/méthodes
- Primitifs interdits => classes enveloppes, auto-(un)boxing, généricité systématique => `Coll<TypeE>` (à la déclaration *ET* à la création)
- Interface la plus haute = `Collection`
`List` extends `Collection`, `AbstractList` implements `List`,
`ArrayList` / `LinkedList` extends `AbstractList`, `Collections`
- `Iterator<TypeE>` permet de parcourir une `Collection<TypeE>`, fournit 3 méthodes : `hasNext`, `next`, `remove`
- Les parcours : simples ou non ?
 - Simple => boucle **for each** (fonctionne aussi sur les tableaux)
 - Non simple (ou jusqu'à java 1.4.2) => boucle `while` avec un itérateur