JAVA ET LE WEB

Licence Devops
JEE
2018/2019

Fayçal BRAÏKI
etudiantspsp@free.fr

Sommaire

I. Sérialisation et désérialisation
II. La généricité
III. Les conteneurs et algorithmes
IV. L'utilitaire javadoc

Sérialisation et désérialisation

- Le développeur peut avoir besoin de sauvegarder l'état d'un objet pour stockage sur une mémoire de masse ou à des fins de transmission sur un réseau

- Pour ce faire, le langage JAVA introduit les notions de sérialisation et désérialisation
- Cette méthode est utilisé par RMI (Remote Method Invocation) ou encore par les Javabeans

- Les opérations de sérialisation et de désérialisation sont assimilables aux opérations de lectures/écritures de flux vus précédemment

Sérialisation et désérialisation

- Ces opérations nécessitent que la classe d'objets implémente l'interface *Serializable* impliquant l'import du package java.io.

- Cela permet de disposer des méthodes :
  - *private void writeObject(Objet_à_serialiser)*
  - *private void readObject()*

- De la même manière que les autres flux (fichiers, clavier,…), la sérialisation/désérialisation nécessite l'ouverture et la fermeture de flux d'entrée et de sortie

- Ces derniers seront respectivement de type *ObjectInputStream et ObjectOutputStream*

Rappel sur les Entrées/Sorties

- Les flux d'entrée
  - *La lecture d'un flux d'entrée suit le schéma suivant :*
    1. Ouverture du flux : elle se produit à la création d'un objet de la classe *InputStream* en précisant au moment de l'appel au constructeur quel élément externe (fichier, clavier,…) est à relier au flux
    2. Lecture des données : les données du flux sont lues à l'aide d'une méthode dépendante du flux de données ouvert (*ici readObject())*
    3. Fermeture du flux : lorsqu'il n'y a plus besoin du flux ouvert, il peut être fermé à l'aide la méthode close()

Rappel sur les Entrées/Sorties

- Les flux de sortie
  - *L'écriture d'un flux de sortie suit le schéma suivant :*
    1. Ouverture du flux : elle se produit à la création d'un objet de la classe *OutputStream*
    2. Écriture des données : les données du flux sont écrites à l'aide d'une méthode dépendante du flux de données ouvert (ici *writeObject())*
    3. Fermeture du flux : lorsqu'il n'y a plus besoin du flux ouvert, il peut être fermé à l'aide la méthode close()

Exemple

- Sérialisation d'une voiture

```
Voiture a=new Voiture (15,160,5,5,"bleu");
//sérialisation
FileOutputStream fs = new FileOutputStream("C:\\Users\\voiture.dat");
ObjectOutputStream os = new ObjectOutputStream(fs);
os.writeObject(a);
os.close();

//désérialisation

Voiture z;
FileInputStream fis = new FileInputStream("C:\\Users\\voiture.dat");
ObjectInputStream ois = new ObjectInputStream(fis);
z = (Voiture) ois.readObject();
ois.close();
```

LA GENERICITE

- *Concept*

  - *Nous avons pu nous rendre compte que le java était un langage fortement « typé »*

  - *Néanmoins, certains mécanismes permettent de s'affranchir partiellement de cette contrainte :*
    - surcharge des méthodes
    - classes abstraites
  - *Cependant, même avec l'aide de ces mécanismes, obligation nous est faite de prototyper et définir les méthodes en y incluant les types*

LA GENERICITE

- *L'objectif de la généricité en java est de définir des classes, des interfaces (et donc des méthodes) indépendantes du type des données manipulées et donc de fournir du code réutilisable*

- *Il s'agit aussi de limiter autant que possible les opérations de cast à partir de la classe Object*

- *Exemple de classe générique : classe Pile contenant des objets quelconques et les méthodes associées permettant des les manipuler (empiler, dépiler, etc…)*

LA GENERICITE

- Mise en œuvre de la généricité en java
  - Le type devient un « paramètre » de la classe/interface ou de la méthode
  - Le code adéquat est généré au moment de la compilation par inférence de types
  - Syntaxe :

```
//T indique un type changeant (peut y en avoir plusieurs)
 public class MaClasse<T>
 {…}
```

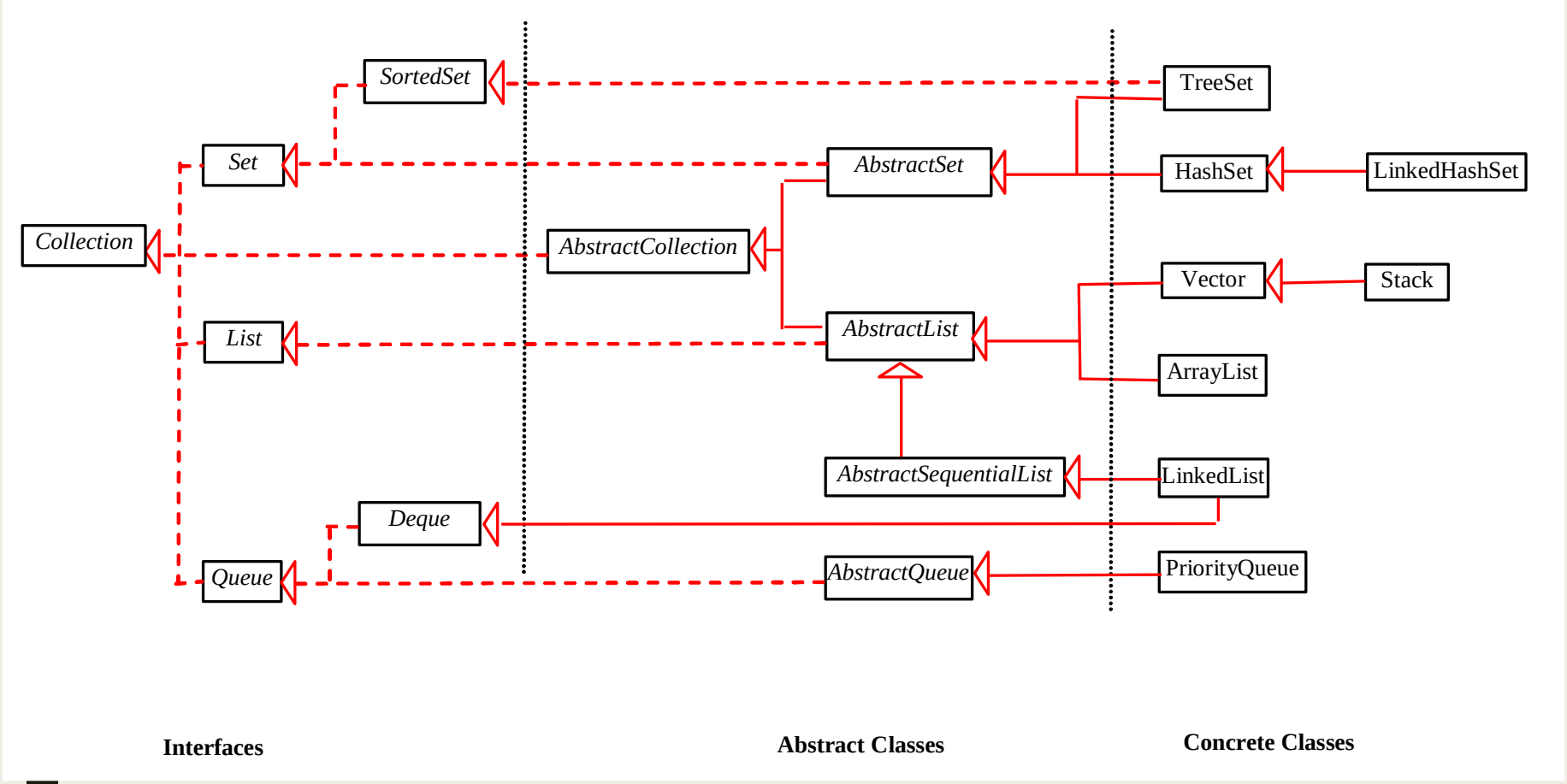
LA GENERICITE

- Exemple de classe générique

LES CONTENEURS ET ALGORITHMES

- JAVA met à disposition du développeur les principales structures de données et algorithmes par le biais de la bibliothèque java.util :
  - les vecteurs dynamiques (ArrayList et Vector)
  - Les listes (LinkedList)
  - Les piles (Stack)
  - Tri, recherche de min/max,…
- Les structures de données héritent toutes de l'interface *Collection*

LES CONTENEURS ET ALGORITHMES



LES CONTENEURS ET ALGORITHMES

- Les vecteurs dynamiques : la classe *ArrayList*
  - Structure de données implémentant un tableau dynamique d'objets
  - Les opérations de base :
  - La construction : ArrayList tab=new ArrayList();
  - L'ajout : tab.add(element);
  - La suppression d'un élément : tab.remove(i);
  - L'accès : tab.get(i);

LES CONTENEURS ET ALGORITHMES

- Les vecteurs dynamiques : la classe *Vector*
  - Structure de données implémentant un tableau dynamique d'objets avant java 2
  - Obsolète mais dans la pratique toujours utilisée…

LES CONTENEURS ET ALGORITHMES

- Les listes : la classe *LinkedList*
  - Implémentation de liste (doublement) chaînée
  - Les opérations de base :
  - La construction : LinkedList liste=new LinkedList();
  - L'ajout : liste.add(element);
  - La suppression d'un élément : liste.remove(i);

  Ou liste.removeFirst() ou liste.removeLast()
  - Le parcours : liste.next(); ou liste.previous(); ou liste.getFirst() ou liste.getLast()

LES CONTENEURS ET ALGORITHMES

- Les piles : la classe *Stack*
  - Implémentation de pile (LIFO)
  - Les opérations de base :
  - La construction : Stack pile=new Stack();
  - L'ajout : pile.push(element);
  - La suppression d'un élément : pile.pop()
  - Le parcours : pile.peek();
  - Test de vacuité : pile.empty();

LES CONTENEURS ET ALGORITHMES

- Les algorithmes
  - Implémentation de pile (LIFO)
  - Les opérations de base :
  - La construction : Stack pile=new Stack();
  - L'ajout : pile.push(element);
  - La suppression d'un élément : pile.pop()
  - Le parcours : pile.peek();
  - Test de vacuité : pile.empty();

L'UTILITAIRE JAVADOC

- Il s'agit d'un outil intégré qui permet de générer automatiquement la documentation du code
- La documentation générée est au format html
- Elle contient :
  - La description des membres
  - Des liens permettant de naviguer entre les classes
  - Des informations sur l'héritage
- Les commentaires doivent respecter un certain formalisme

L'UTILITAIRE JAVADOC

- **Les principaux tags disponibles**

@author	pour préciser l'auteur de la fonctionnalité
@docRoot	chemin relatif qui mène a la racine de la doc
@deprecated	indique que le membre ou la classe est dépréciée
@link	permet de creer un lien
@param	pour décrire un paramètre
@return	pour décrire la valeur de retour
@see	pour ajouter une section "Voir aussi"
@since	depuis quelle version la fonctionnalité est disponible
@throws	pour indiquer les exceptions lancées par les méthodes
@version	pour indiquer la version de la classe