TD 6 IJA
Structures de données JAVA

Sommaire

- Types de base
  - Types primitifs
  - Classes des types primitifs (wrappers)
- Structures de collection
  - Tableaux
  - L'interface collection
  - Vector et Hashtable
- Remarque: toute classe est une structure de données

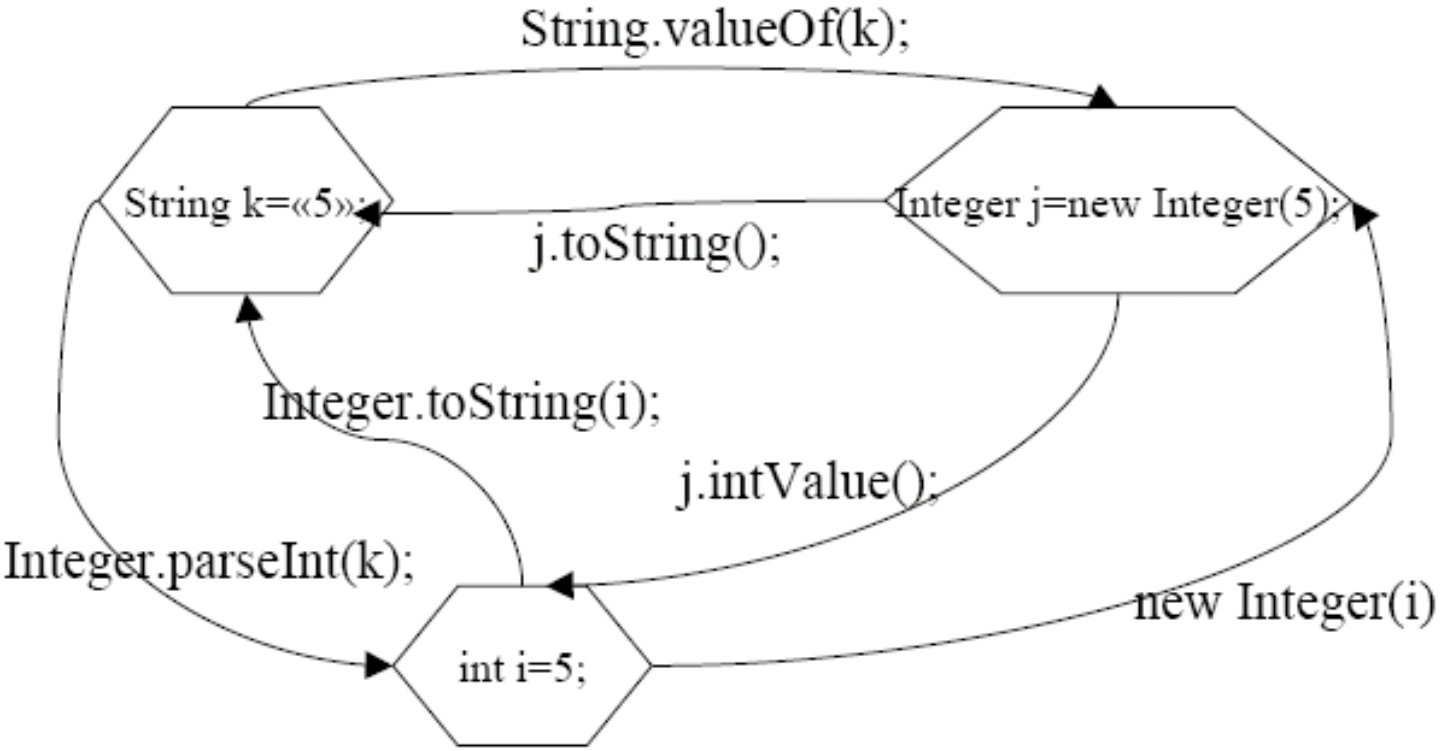
Les types primitifs

- Types
  - byte : 1 octet
  - short : 2 octets
  - int : 4 octets
  - long : 8 octets
  - float : 4 octets
  - double : 8 octets
  - boolean : true/false
  - char: 2 octets (en unicode)

- Un type primitif ne prend jamais de majuscule

Classes des types primitifs (Wrappers)

- Classes qui encapsulent des types primitifs
- Exemples: String, Integer, Boolean, Long…
- Possibilité de conversion: Exemple



Exercice : Wrappers

- public static void main(String args[])
  - args est un tableau de chaînes de caractères qui contient les mots frappés à l'appel de la machine virtuelle
  - exemple : java Truc toto tata
    - args[0] contient "toto", args[1] contient "tata"
- Exercice
  - Ecrire une classe Additionne qui additionne deux entiers donnés en paramètre au lancement de la machine virtuelle
  - exemple : java Additionne 2 5

Structures de collection

- Collection (conteneur) = ensemble générique d'objets
  - Main de bridge (collection de cartes), répertoire de fichiers, répertoire téléphonique, …
- Les objets peuvent être soumis à des contraintes
  - Ordre (liste), entrées uniques (ensemble), …

Les tableaux

- Un tableau JAVA est une structure collectant un ensemble (dont on connaît sa cardinalité d'avance) d'objets de même type
- Déclaration d'un tableau

  type[] unTableau; (type peut être n'importe quel objet ou type de base)
- Instanciation
  - unTableau = new type[10];
- Accès aux éléments
  - unTableau[7] = new type();
  - System.out.println(unTableau[7]);

L'interface JAVA « Collection »

- Séparation de l'interface d'une collection de son implémentation (ex: file d'attente) polymorphisme
- Les principales méthodes:
  - add(Object o), contains(Object o), iterator(), remove(Object o)
- Iterator pour parcourir une collection
  - hasNext(), next(), remove()
- Quelques extensions de l'interface Collection
  - List, Set
  - ArrayList, TreeSet, Vector

Vector & Hashtable

- Vector
  - Une classe de collection qui engendre un tableau d'objets
  - Méthodes: add(Object o), get(int index), remove(int index)…
  - Gère une taille dynamique du tableau
- Hashtable
  - Une classe de collection qui gère un ensemble de couples (clé, valeur)
  - Méthodes: put(Object key, Object value), get(Object key), remove(Object key)
  - Recherche transparente par rapport à l'utilisateur

Exercice: LIFO

1) Ecrire une classe cliente de la LIFO

2) Définir la classe pile (LIFO).

Cette classe contient les méthodes :

```
public Object push(Object item) ;
public Object pop ();
```

3) Modifiez la classe pour y intégrer les exceptions sur push et pop

Exercice: Gestion d'étudiants avec un vecteur

0) Créer une classe ClientEtud, qui instancie 10 objets de la classe Etudiant (nom, prénom, num étudiant). Chaque objet doit avoir un nom différent. Au moins un de ces objets aura comme nom « toto » et l'autre « steph »

1) Vous insérez ces dix objets dans une classe GesEtud, qui répond à la méthode void ajout(Etudiant etu). Pour pouvoir stocker les étudiants, la classe repose sur un Vector interne, listEtud.

2) Dans la classe ClientEtud, rechercher « steph », avec la méthode recherche(String nom) que vous implanterez dans GestEtud. Quelle est la complexité de la méthode de recherche ?

Exercice: Gestion d'étudiants avec une Hashtable

0) Copiez la classe GesEtud en GesEtudHash
1) Consultez la documentation sur la Hashtable
2) Remplacez l'attribut interne par une Hashtable
3) Dans la classe ClientEtud, rechercher « steph », avec la méthode recherche(String nom) que vous implanterez dans GestEtud. Quelle est la complexité de la méthode de recherche ?
4) Intégrer la gestion d'exceptions dans GestEtudHash et dans la classe cliente