1

CONCEPTION LOGICIELLE

Présentée par: Mme HASSAM-OUARI Kahina
Email: kahina.hassam@yncrea.fr
Bureau: T336
Département: **O**rganisation , **M**anagement et **I**nformatique

2

Résultats d'apprentissage

Penser, concevoir et programmer en Objet

3

Historique du langage Java…

- Le père du projet Java est James Gosling et Patrick Naughton (employés de sun mycrosystems)
- Idée initiale : étendre le langage C++
- Puis écriture du premier compilateur Java en Langage C.
- Fin 1994, premier compilateur Java écrit en Java (Arthur Van Hoff)
- La société Sun a été ensuite rachetée en 2009 par la société Oracle qui détient et maintient désormais Java.

4

Quelles applications peut on réaliser en java ?

- Oracle propose plusieurs plateformes de java, selon les types d'applications qu'on veut réaliser:

- JAVA SE (Standard Edition):
  - des applications, sous forme de fenêtre ou de console ;
  - des applets, qui sont des programmes Java incorporés à des pages Web ;
- J2ME (Java 2 Micro Edition)
  - des applications pour appareils mobiles, comme les smartphones, domotique..;

- avec J2EE (Java 2 Enterprise Edition, maintenant JEE) ;
  - des sites web dynamiques,

- et bien d'autres : J3D pour la 3D. . .

5

Architecture de JAVA SE (Standard Edition):

Le **Java Development Kit** (JDK): Possède
- Les outils pour écrire du code java
- Des outils pour compiler du code Java pour le transformer en *bytecode destiné à la JVM* .
- JRE

- Le *Java Runtime Environment (JRE):*
  - *JRE Library: bibliothèques utilisées par les programmes Java*
  - *JVM(Java Virtual Machine):*
    - *se charge d' interpréter le bytecode généré à la compilation du programme java,*
    - La JVM dépend de l'architecture matérielle.

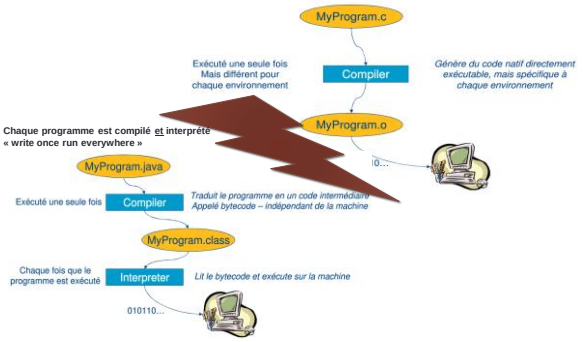
6

Propriétés du langage Java

- Portabilité
- Orienté Objet
- Robuste (peu de bidouillages autorisés)
- Sûr (ByteCode Verifié)

7

Déploiement d'un programme C et en java ?



8

L'IDE (Environnement de Développement Intégré) *eclipse pour développer en java*

Le manuel d'installation d'eclipse se trouve sur itlearning dans le cours de conception logicielle

9

Principes et concepts de base du langage Java

10

Vocabulaire Java… les variables

- Une variable est un endroit de la mémoire auquel on a donné un nom de sorte que l'on puisse y faire facilement référence dans le programme
- Une variable possède :
  un nom + une **valeur** + un **type**
- La valeur d'une variable peut changer (varier) au cours du temps et de l'exécution du programme. **Par contre son type ne change pas !**

11

Vocabulaire Nom des variables

- Le 1er caractère d'un nom peut être:

AUTORISES	INTERDITS
Lettre	Chiffre
$	Signe de ponctuation
_	

- Le nom ne dois jamais contenir :
  - des espaces
  - des caractères internationaux (accents, etc.)

12

Vocabulaire Java… Types primitifs

- Java fournit plusieurs types prédéfinis : les types primitifs, leur nombre est limité

En français	Type en java	Valeurs possibles
Nombre entier	byte	-128 à 127
	short	-32768 à 32767
	int	-2^{31} à $2^{31}-1$
	long	-2^{63} à $2^{63}-1$
Nombre flottant (à virgule)	float	$1.4*10^{-45}$ à $3.4*10^{38}$
	double	$4.9*10^{-324}$ à $1.8*10^{308}$
Caractère	char	tous?
Vrai ou faux	boolean	true ou false

13

Variable en Java…
Déclaration et initialisation

Type ET nom ET valeur

Type	Exemple de création et initialisation
`int`	`int a = 12;`
`short`	`short b = 32;`
`long`	`long c = 200L;`
`byte`	`byte d = 10;`
`double`	`double e = 10.5;`
`float`	`float f = 23.2323f;`
`char`	`char g = 't';`
`boolean`	`boolean h = true;`

14

Vocabulaire Java…
Type chaine de caractères

- Les chaînes de caractères sont essentielles et omniprésentes dans les programmes informatiques
- Pas de type primitif « String » ☹
- En java, c'est la classe « String » qui permet de créer des chaine de caractères ☺

15

Opérateur pour les String
La concaténation avec le « + »

- **Déclaration**
```
String s1, s2;
```
- **Initialisation**
```
s1 = "Je pense";
s2 = "donc je suis";
```
- **Déclaration et initialisation en une ligne**
```
String s3 = "Hello";
```
- On peut « concaténer » (mettre bout à bout) deux Strings pour n'en faire qu'une seule.
```
String s4 = "Je pense"+ s2;
```

L'affichage de s4 à l'écran donnerait : Je pensedonc je suis

16

Opérateur pour les String
La méthode equals

- Pour comparer deux chaines de caractères, il faut utiliser la fonction java *equals()*.
- Cette fonction renvoie :
  - true : si les deux chaines de caractères sont identiques
  - false : si elles ne le sont pas

Exemple:

```
String chaine1, chaine2;
boolean res;
chaine1= "Bonjour";chaine2="Bonjour";
res=chaine1.equals(chaine2);
```

17

Grammaire Java
Opérateurs numériques 1/2

Niveau	Symbole	Signification
1	()	Parenthèse
2	*	Produit
	/	Division
	%	Modulo
3	+	Addition ou concaténation
	-	Soustraction

```
int i1 = 4;
int i2 = i1 + 4;
int i5 = (i1 % i2) - 5 * 4;
```

1 2 3

18

Grammaire Java
Opérateurs numériques 2/2

- Comparer des valeurs numériques :

Opérateur	Exemple	Renvoie TRUE si
>	v1 > v2	v1 plus grand que v2
>=	v1 >= v2	Plus grand ou égal
<	v1 < v2	Plus petit que
<=	v1 <= v2	Plus petit ou égal à
==	v1 == v2	égal
!=	v1 != v2	différent

```
int i1 = 4;
int i3 = i1+ 4;
boolean resultat = i1 < i3;
```

19

Grammaire Java
Opérateurs logiques

- Opérateurs logiques:

Opérateur	Usage	Renvoie TRUE si
&&	expr1 && expr2	expr1 et expr2 sont vraies
||	expr1 || expr2	Expr1 ou expr2, ou les deux sont vraies
!	! expr1	expr1 est fausse
!=	expr1 != expr2	si expr1 est différent de expr2

```
int i1 = 4;
int i3 = i1 + 4;
boolean inferieur = i1 < i3;
boolean res = inferieur && (i3>0);
```

20

Grammaire Java
Instructions et blocs

- Une instruction
  - Une seule par ligne
  - Est presque toujours suivie par un « ; »
  - Réalise un traitement particulier
  - Renvoie éventuellement le résultat d'un calcul
  - Il en existe plusieurs types: déclaration, assignation...etc
- Un bloc
  - Est une suite d'instructions entre accolades « { » et « } »
  - Doit toujours être refermé (autant de « { » que de « } »)
  - Délimite la portée des variables
  - Suit (presque) toujours la déclaration de classe et méthodes
  - Définit aussi le contenu des boucles et structures conditionnelles

21

Structure d'une classe en java
Exemple Simple

```
public class nomDeVotreClasse {
//propriétés (des attributs)

//1 ou 0 méthode principale
    public static void main(String[] args) {
    //ici mettre le code sans les//

    }

//d'autres méthodes de la classe

}
```

Il n'existe qu'un seul main par application

22

Les entrées- sorties

- Les composants d'entrées/sorties sont très divers:
  - claviers,
  - écrans,
  - imprimantes,
  - disques souples ou durs,
  - souris,
  - mais aussi tout appareil que l'on souhaite piloter par ordinateur

23

Les sorties Affichage à l'écran

- **Ecriture:** Communiquer des valeurs vers l'extérieur (utilisateur). Se fait en général via un affichage à l'écran. En Java, cet affichage se fait sur **la console**.
- Elle permet :
  1. D'Afficher un texte :
  ```
  System.out.println("Texte à afficher");
  ```
  2. D'Afficher la valeur d'une variable :
  ```
  System.out.println(nom_de_la_variable);
  ```
  3. De mélanger de texte et des valeurs :
  ```
  System.out.println("Texte"+nom_de_la_variable+
  "texte" +nom_de_la_variable);
  ```

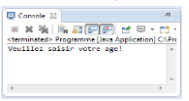
24

Les sorties Par l'exemple

Programme en java:

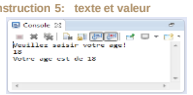
```
public class Programme {

 public static void main(String[] args) {
   int age;    1
   System.out.println("Veuillez saisir
votre age!"); 2
       age=18; 3
       System.out.println(age); 4
     System.out.println("Votre age est de "+age);5
   }

}
```

Instruction 2: affichage du texte



Instruction 4: affichage de la valeur

Instruction 5: texte et valeur



25

Les entrées Saisie de données

- **Lecture**: Entrer des valeurs externes (utilisateur) pour qu'elles soient utilisées dans la suite du programme. Par exemple par une saisie au clavier.

- Pour chaque type primitif il existe une méthode de récupération de données **(sauf les char) :**
  - nextLine(): pour récupérer une String
  - nextInt(): pour récupérer un entier (**int**),
  - nextDouble(): pour récupérer un double (**double**),
  - nextFloat(): pour récupérer un réel (**float**),
  - nextBoolean(): pour récupérer un booléen (**boolean**)

26

Les entrées Syntaxe

- Syntaxe pour lire des données clavier :

```
java.util.Scanner sc= new java.util.Scanner(System.in);
System.out.println("Pour saisir un booléen (true /false)");
boolean b=sc.nextBoolean();

java.util.Scanner sc1= new java.util.Scanner(System.in);
System.out.println("Pour saisir un entier");
int b=sc1.nextInt();

java.util.Scanner sc2= new java.util.Scanner(System.in);
System.out.println("Pour saisir une chaine de caractères");
String b=sc2.nextLine();

java.util.Scanner sc3= new java.util.Scanner(System.in);
System.out.println("Pour saisir un float");
float b=sc3.nextFloat();
```

Les entrées Syntaxe

- Syntaxe pour lire des données clavier :

```
java.util.Scanner sc= new java.util.Scanner(System.in);
System.out.println("Pour saisir un booléen
(true /false)");
boolean b=sc.nextBoolean();

java.util.Scanner sc1= new
java.util.Scanner(System.in);
System.out.println("Pour saisir un entier");
int b=sc1.nextInt();

java.util.Scanner sc2= new
java.util.Scanner(System.in);
System.out.println("Pour saisir une chaine de
caractères");
String b=sc2.nextLine();

java.util.Scanner sc3= new
java.util.Scanner(System.in);
System.out.println("Pour saisir un float");
float b=sc3.nextFloat();
```

1-Créer une variable de type Scanner
2- Mettre une phrase pour guider l'utilisateur dans sa saisie.
3- Selon le type qu'on veut saisir utiliser la méthode adéquate

27

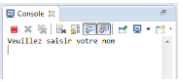
28

Les entrées
Par l'exemple

Programme en java:

```
public class Programme {

   public static void main(String[] args) {
      java.util.Scanner sc =
          new java.util.Scanner(System.in);
      System.out.println("Veuillez saisir
votre nom"); 1
      String  nom=sc.nextLine(); 2
      System.out.println("vous avez saisi
"+nom); 3
   }

}
```

Instruction 1 et 2: affichage du texte+ apparition d'un curseur pour la saisir



Instruction 3: affichage de la saisie utilisateur

29

Les entrées
Génération aléatoires..

Pour mettre une valeur aléatoire dans une variable aléatoirement (non saisie par l'utilisateur):

1. On crée une variable Random:

```
java.util.Random generateur = new java.util.Random(System.currentTimeMillis());
```

2. Selon le type des valeurs qu'on souhaite générer, on appelle la méthode correspondante sur la variable crée:

Appel	Génère
generateur.nextBoolean() ;	true ou false
generateur.nextInt(25) ;	Un entier compris entre [0,24]
generateur.nextDouble() ;	Un double compris entre [0.0,1.0[
generateur.nextFloat() ;	Un float compris entre[0.0, 1.0f[

30

Grammaire Java
Structures conditionnelles

- Si *condition* alors … : la structure **if**

```
if (CONDITION_VRAIE) {

}
```

- Si *condition* alors … sinon … la structure **if…else**

```
if (CONDITION_VRAIE) {
} else {
}
```

- Si *condition* alors … sinon si condition alors…sinon si condition alors…sinon..

```
if (CONDITION_VRAIE) {

} else if (CONDITION_VRAIE) {
    }
else if (CONDITION_VRAIE) {

} else{ }
```

31

Exercice
Votre premier programme java

- Ecrire un programme java qui demande à un étudiant 3 valeurs correspondant aux notes de 3 UEs.
- Si la moyenne de ses 3 notes est supérieure à 10 alors on affiche que l'étudiant: « GUE validé », sinon « GUE non validé »

32

Structures itératives

33

Structures itératives
Les types

- Elles sont de 2 types:
  1. À bornes non définies: utilisées lorsque le nombre d'itérations est **inconnu**,
     - Boucle **while**
     - Boucle **do...while**
  2. À bornes définies: utilisées lorsque le nombre d'itérations est **connu**,
     - Boucle **for**

34

Structures itératives
La boucle while

Fonction: répéter une suite d'instructions tant qu'une condition est vraie

Syntaxe :

```
while (conditionVraie){
    instructions;
}
```

Remarques :

- Si la condition **est fausse dès le départ**, les instructions ne sont **jamais** exécutées
- Si la condition est toujours vraie, le programme ne s'arrêtera jamais

35

Structure itérative
boucle do… while

Fonction: exécuter une suite d'instructions ***au moins une fois*** et la répéter jusqu'à ce que la condition soit vraie

Syntaxe :

```
do{
   instructions;
}while(conditionVraie);
```

Remarque: Les instructions sont exécutées au moins une fois

36

Structure itérative à borne définie
La boucle For..

Fonction: répéter un bloc d'instructions un certain nombre de fois.

Syntaxe:

- Un for Incrémenté:

```
for(int i=valInitiale;i<valeurFinale;i++) {
   instructions;
}
```

- Un for décrémenté

```
for(int i=valeurFinale;i>=valInitiale;i--) {
   instructions;
}
```

37

Structures de données
Tableaux

38

Structures de données
Les tableaux

Un tableau est un ensemble **ordonné** et **homogène** de valeurs :

- Ordonné car les cases mémoires composants un tableau sont numérotés à partir de 0
- Homogène car un tableau possède des valeurs du même type.

39

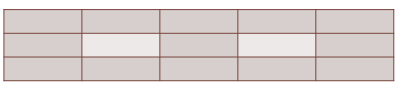
Les tableaux

-Structure de données permettant d'effectuer un même traitement sur des données de même type.

-Il y a deux types de tableaux:

- Tableau à **une** dimension

- Tableau à **deux** dimensions



40

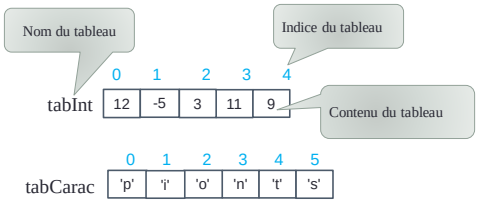
Définition d'un tableau

- On définit un tableau par:
  - Son nom
  - Ses dimensions 1 ou 2
  - Sa taille
  - Le type de données qu'il contient

- Un tableau **doit être déclaré, initialisé, et rempli**

41

Contenu d'un tableau à 1 dimension



Remarques :

- L'indexation des tableaux commence toujours à 0

42

Manipulation d'un tableau à 1 dimension en java
Exemple: Tableau d'entiers

- Déclaration d'un tableau d'entiers :

```
int[] tabValeur;
```

- Création d'un tableau

```
tabValeur= new int[10];
```

- On peut déclarer et créer un tableau dans une seul instruction

```
int[] tabValeur= new int[10];
```

type des éléments — Nom du tableau — Taille du tableau

- La capacité du tableau est fournie par l'instruction nomDuTableau.length

```
int longueur = tabValeur. length;
```

43

Affichage et affectation

```
public class Exemple{

  public static void main(String[] args ){
     int[] tabVal = new int[8];
  java.util.Scanner entree =new java.util.Scanner Scanner(System.in);
     for (int i = 0; i < 8; i++) {
      System.out.println("veuillez saisir une valeur à l'indice "+i);
         tabVal[i] = entree.nextInt();
        System.out.println("la valeur saisie est"+tabVal[i]);
       }
   }
}
```

44

Exercice

- Ecrire un programme java dont lequel 10 valeurs aléatoires sont générées et stockées dans un tableau à 1 dimension. Ensuite, le programme affiche: la somme, le minimum, le maximum et la moyenne de ces valeurs.

45

Les sous-programmes ou les méthodes

46

Notion de Sous-programme

- Sous-programme : petit programme qui s'exécute à l'intérieur d'un autre programme et qui réalise un traitement spécifique.
- Deux cas principaux justifiants leurs utilisations:
  - Répétition d'un même traitement dans le programme principal
  - Organisation et lisibilité du code
- Sous-programme = méthode en Java/langage objet

48

Création d'une méthode
Exemple de x^n

programme

```
public class Cours4 {
   //calcul x^n
   public static void main(String[] args) {
      // variables
java.util.Scanner entree = new java.util.Scanner(System.in);
      int x, n, res = 1;
      //code
      System.out.println("Nombre ?");
      x = entree.nextInt();
      System.out.println("Puissance ?");
      n = entree.nextInt();

      for (int i=0;i<n;i++){
         res = res*x;
      }
      System.out.println("Résultat "+ res);
   }
}
```

Programme complet qui va calculer x puissance n (x et n sont des entiers) à partir des saisies de l'utilisateurs.

49

Création d'une méthode
Exemple de x^n

programme

```
public class Cours4 {
   //calcul x^n
   public static void main(String[] args) {
      // variables
java.util.Scanner entree = new java.util.Scanner(System.in);
      int x, n, res = 1;
      //code
      System.out.println("Nombre ?");
      x = entree.nextInt();
      System.out.println("Puissance ?");
      n = entree.nextInt();

      for (int i=0;i<n;i++){
         res = res*x;
      }
      System.out.println("Résultat "+ res);
   }
}
```

Programme complet qui va calculer **x** puissance **n** à partir des saisies de l'utilisateurs

Bout de code qu'on souhaite ré-utiliser

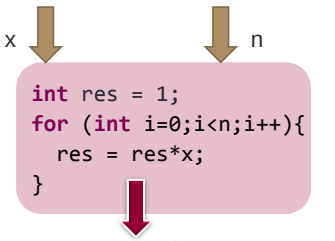
50

Caractériser ce sous-programme

```
for (int i=0;i<n;i++){
  res = res*x;
}
```

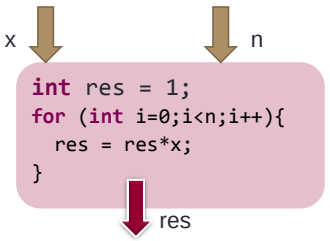
51

Caractériser ce sous-programme



52

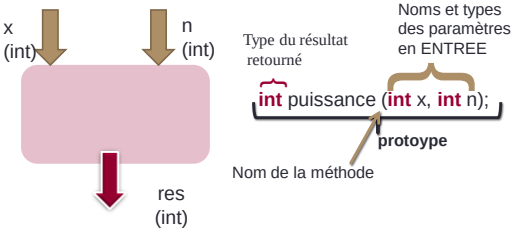
Caractériser ce sous-programme



3 paramètres:

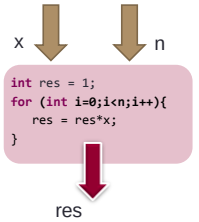
- 2 paramètres *en entrée* : n et x (type int)
- 1 paramètre en *sortie*: res (type int)

Paramètres e/s et prototype d'une méthode
Exemple de puissance



Prototype d'une méthode = entête de la méthode suivi du symbole de fin d'instruction.

Du prototype à la méthode




```
int puissance (int x, int n){
    int res = 1;
    for (int i=0;i<n;i++){
            res = res*x;
    }
    return res;
}
```

Mot clef du langage pour signifier que la méthode **retourne une copie de la valeur de res**

Méthode qui ne retourne rien

- Une méthode peut ne pas retourner une valeur
  - Cette méthode provoque un effet de bord (modification de l'environnement: écriture dans un fichier…).
- Syntaxe:
  - **A la place du type de retour mettre un *void***
  - **Enlever le mot clé *return***

void nomMethode (types et noms des paramètres)

Méthode qui ne retourne rien

- Exemple d'une méthode sans retour:

```
public void helloWorld(){
      System.out.println("Bonjour tout le monde");
}
```

- Le mot clef ***void*** veut dire pas de type de retour, donc par extension pas de retour.

Appel d' une méthode

- Une fois la méthode définie, celle-ci pour être utilisée, doit être appelée dans une méthode (exemple pg principal)

- On distingue 2 types d'appels:
  a) Quand la méthode retourne un résultat, celui-ci doit être stocké dans une variable:
     - double resultat = puissance(4,5);
  b) Quand la méthode ne retourne pas de résultat, pas besoin de stocker le résultat dans une variable:
     - helloWorld();

Portée de variables – Exemple

```
public class Cours4 {
   static String texte1 = " puissance ";

   public static int puissance(int x, int n){
      int variableLocale = 1;
      for (int i=0;i<n;i++) {variableLocale = variableLocale*x};
      return variableLocale;
   }

   static  String texte2 = " = ";

   //calcul x^n
   public static void main(String[] args) {
       int x1 = 2, n1 = 3;
       System.out.println(x1+texte1+n1+texte2+ puissance(x1,n1));
   }
}
```

Portée de variables –Solution

```
public class Cours4 {
   static String texte1 = " puissance ";

   public static int puissance(int x, int n){
      int variableLocale = 1;
      for (int i=0;i<n;i++) {variableLocale = variableLocale*x};
      return variableLocale;
   }

   static  String texte2 = " = ";

   //calcul x^n
   public static void main(String[] args) {
       int x1 = 2, n1 = 3;
       System.out.println(x1+texte1+n1+texte2+ puissance(x1,n1));
   }
}
```

Portée des variables

- Une variable est caractérisée par un nom, un type, mais aussi une portée.
- On distingue :
  - les **variables globales :** utilisables partout dans le programme et ses différentes méthodes,
  - les **variables locales:** utilisables que dans la méthode où elles ont été déclarées.

Exercices:

- Exercice 1:
  - Ecrire une méthode java qui permet de vérifier l'existence d'une chaine de caractères saisie par l'utilisateur dans un tableau
  - Appelez la méthode dans un programme principal

D'autres Structures de données utiles

Les collections
Création 1

Je définis quelle structure de données je souhaite utiliser

```
NomCol<Type> nomVariable= new NomCol<Type>();
```

Je dis à Java que tous les éléments de la collection sont du type Type (entiers, nombres à virgule…)

Je nomme ma collection

Les collections
Création 1

- Type peut prendre les valeurs suivantes

Si je veux des	Je mets le mot
int	Integer
double	Double
boolean	Boolean
char	Character
String	String

65

Les collections
Création d'une Liste

- NomCol peut prendre les valeurs suivantes

Si je veux	Je mets
une liste	LinkedList ArrayList

Création d'une liste d'entiers:

```
LinkedList<Integer> maListe= new LinkedList<Integer>();
ArrayList<Integer> maListe= new ArrayList<Integer>();
```

65

66

Les collections
Création d'une pile

- NomCol peut prendre les valeurs suivantes

Si je veux	Je mets
une liste	LinkedList ArrayList
une pile	Stack

Création d'une pile d'entiers:

```
Stack<Integer> maPile= new Stack<Integer>();
```

66

67

Les collections
Création d'une HashSet

- NomCol peut prendre les valeurs suivantes

Si je veux	Je mets
une liste	LinkedList ArrayList
une pile	Stack
un ensemble	HashSet

Création d'un ensemble d'entiers:

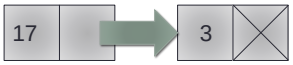
```
HashSet<Integer> monSet= new HashSet<Integer>();
```

67

68

Les collections
Modèle théorique

- Composée de maillons ordonnées
- Chaque maillon est un couple :
  - Variable
  - Pointeur vers le maillon suivant



- Donc taille infinie (contrairement aux tableaux) : il suffit de rajouter un maillon

69

Les collections : listes

Si je veux	Je mets
une liste	LinkedList ArrayList

- Les LinkedList Java fonctionnent quasiment sur le modèle théorique : ce sont des listes (doublement) chainées, et la queue de la liste pointe vers sa tête (= anneau)

- Les ArrayList reposent sur des tableaux automatiquement redimensionnés au besoin.

70

Les listes
Méthodes

boolean	**add(Object o)**	Ajouter un objet à la collection.
void	**add(int index, Object o)**	Ajouter un objet à la liste en position donnée.
Object	**get(int index)**	Retourne l'élément à la position index
boolean	**isEmpty()**	Tester si la collection est vide.
boolean	**contains(Object o)**	Tester si la collection contient l'objet indiqué.
int	**size()**	Retourne la taille de la collection
void	**clear()**	Vider la collection
Object	**remove(int index)**	Retourne l' objet retiré à la position index de la collection.
boolean	**set(int index, Object element)**	Remplace l'élément à la position index.

71

Les listes
Méthodes

boolean	**add(Object o)**	Ajouter un objet à la collection.
void	**add(int index, Object o)**	Ajouter un objet à la liste en position donnée.

Exemple :

```
LinkedList<Integer> maListe= new LinkedList<Integer>();
maListe.add(8);
maListe.add(0,10)
```

72

Les listes
Méthodes

Object	**get(int index)**	Retourne l'élément à la position index

Exemple:

```
LinkedList<Integer> maListe= new LinkedList<Integer>();
maListe.add(8);

int valeurRecuperee= maListe.get(0);
```

73

Les listes
Méthodes

boolean	**isEmpty()**	Tester si la collection est vide.

Exemple:

```
LinkedList<Integer> maListe= new LinkedList<Integer>();
maListe.add(8);
boolean res= maListe.isEmpty();
```

74

Les listes
Méthodes

boolean	**contains(Object o)**	Tester si la collection contient l'objet indiqué.

Exemple:

```
LinkedList<Integer> maListe= new LinkedList<Integer>();
maListe.add(8);
maListe.add(20);
maListe.add(18);
maListe.add(9);
boolean res= maListe.contains(20);
```

75

Les listes
Méthodes

int	**size()**	Retourne la taille de la collection

Exemple :

```
LinkedList<Integer> maListe= new LinkedList<Integer>();
maListe.add(8);
maListe.add(8);
maListe.add(20);
maListe.add(18);
maListe.add(9);

int taille= maliste.size();
```

76

Les listes
Méthodes

void	**clear()**	Vider la collection

Exemple:

```
LinkedList<Integer> maListe= new LinkedList<Integer>();
maListe.add(8);
maListe.add(20);
maListe.add(17);
int res= maListe.size();
System.out.println(res);    //Affichera
maListe.clear();
System.out.println(maListe.size()); //Affichera
```

77

Les listes
Méthodes

Object	**remove(int index)**	Retourne l' objet retiré à la position index de la collection.

Exemple:

```
LinkedList<Integer> maListe= new
LinkedList<Integer>();
maListe.add(8);
maListe.add(20);
maListe.add(20);
int val= maListe.remove(2);
maListe.remove(1);//peut être utilisée comme ça
```

78

Les listes
Méthodes

boolean	**set(int index, Object element)**	Remplace l'élément à la position index.

Exemple:

```
LinkedList<Integer> maListe= new LinkedList<Integer>();
maListe.add(8);
maListe.add(20);
maListe.add(20);
maListe.set(0,9);
```

81

Les collections : piles
Modèle théorique

- Dernier arrivé, premier sorti
  "Last In, First Out" (LIFO)
- Structure ordonnée et homogène

82

Les collections : piles
Méthodes de piles

- Les méthodes qui permettent de manipuler les piles sont :

Object	**push(Object o)**	Ajouter au sommet de la pile
Object	**pop()**	Retirer le sommet de la pile
boolean	**isEmpty()**	Tester si la collection est vide.
boolean	**contains(Object o)**	Tester si la collection contient l'objet indiqué.
int	**size()**	Retourne la taille de la collection

83

Les collections : piles
Méthodes de piles

Object	**push(Object o)**	Ajouter au sommet de la pile

Exemple:

```
Stack<String> maPile= new Stack<String>();
maPile.push("Bonjour");
maPile.push("hei");
maPile.push("ça va?");
```

84

Les collections : piles
Méthodes de piles

Object	**pop()**	Retirer le sommet de la pile

Exemple:

```
Stack<String> maPile= new Stack<String>();
maPile.push("Bonjour");
maPile.push("hei");
maPile.push("ça va?");
mapile.pop();
```

85

Les collections : piles
Méthodes de piles

boolean **isEmpty()**	Tester si la collection est vide.

Exemple:

```
Stack<String> maPile= new Stack<String>();
maPile.push("Bonjour");
maPile.push("hei");
maPile.push("ça va?");
mapile.pop();
boolean res=maPile.isEmpty();
```

86

Les collections : piles
Méthodes de piles

boolean **contains(Object o)**	Tester si la collection contient l'objet indiqué.

Exemple:

```
Stack<String> maPile= new Stack<String>();
maPile.push("Bonjour");
maPile.push("hei");
maPile.push("ça va?");
mapile.pop();
boolean res1=maPile.contains("BONJOUR");
```

87

Les collections : piles
Méthodes de piles

int **size()**	Retourne la taille de la collection

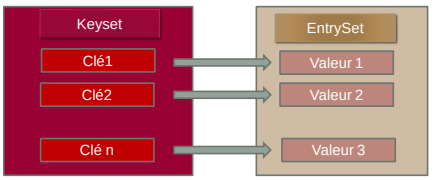
Exemple:

```
Stack<String> maPile= new Stack<String>();
maPile.push("Bonjour");
maPile.push("hei");
maPile.push("ça va?");
mapile.pop();
boolean res1=maPile.contains("BONJOUR");
int taille= maPile.size();
```

88

Les maps ou table de hachage

- Structure de données qui fonctionne selon le principe de paires Clé / Valeur
- Implémentation : HashMap



89

Les maps : HashMap

Méthodes

boolean put(K key, V value)	Ajouter un objet à la collection.
boolean isEmpty()	Tester si la collection est vide.
boolean containsKey(Object o)	Tester si la clé existe
boolean containsValue(Object o)	Tester si la valeur existe
int size()	Retourne la taille de la collection
void clear()	Vider la collection
boolean remove(Object o)	Retirer la clé + valeur si la clé existent

90

Information

- Pour plus d'infos sur la syntaxe du langage java, voir le lien suivant:
  - http://introcs.cs.princeton.edu/java/11cheatsheet/