Java

Dr Maurice Djibril Faye

Java™

Contenu pédagogique

- **Introduction**
- **Types primitifs de base et opérateurs**
- **Structures de contrôle**
- **Programmation Orientée Objet : notions de classes, objets, champs et méthodes**
- **Tableaux en Java**
- **La gestion des chaînes**
- **L'Héritage : classe de base et classe dérivée, polymorphisme, classes abstraites et interfaces**
- **Les Exception : gestion des erreurs légères et graves**
- **Les Threads : gestion des processus.**

Contenu pédagogique

- **Les Threads : gestion des processus.**
- **Les Swing : interfaces graphiques, gestion des événements utilisateurs.**
- **Les entrées sorties : accès au disque, manipulation de fichiers et répertoires, filtres**
- **Collections et algorithmes : vecteurs dynamiques, listes chaînées, ensembles, tables associatives.**
- **JDBC : accès aux bases de données, manipulation de source de données, gestion des transactions.**

Bibliographie

- Java La synthèse – Des concepts objet aux architectures Web, G. Clavel, et al., Dunod, 2000.
- Introduction à la programmation objet en Java, J.Brondeau, Dunod, 1999.
- Le langage Java – Programmer par l'exemple, T. Leduc,D. Leduc, Technip, 2000.
- Algorithmique et programmation objet en Java, V. Granet, Dunod, 2001.
- Initiation à l'algorithmique objet, A. Cardon, C. Dahancourt, Eyrolles, 2001.
- Java Les Cahiers du Programmeur Java, E. Pubaret, Eyrolles, 2004.
- Le langage Java, Irène Charon, Hermes, 2006.

Bibliographie

- Sites : (2017)
  - http://www.oracle.com/technetwork/java/
    - Documentation sur les produits java, liens de téléchargement, JDK8, Netbeans avec JDK8
  - https://netbeans.org/downloads/
    - Java SE 8
  - http://www.java.com/
  - http://download.oracle.com/javase/tutorial/
    - http://docs.oracle.com/javase/tutorial/getStarted/index.html
  - http://docs.oracle.com/javase/8/docs/api/
    - Documentation sur les api de la JSE 8

Bibliographie

- Support de cours :
  - Le cours de Java du Pr Moussa LO, Université Gaston Berger de St-Louis.
  - Implémentation d'IHM en Java avec Swing ,Thierry Duval

Généralités sur Java

- Langage orienté objet développé par Sun en 1990.
  - Sun racheté par Oracle en 2010
- Syntaxe proche de C++
- Java est portable :
  - Code Java écrit sur un PC, s'exécute sur Téléphones, routeurs, appareils sur lesquels une JVM est installée, sans ajustement
  - « write once , run anywhere (WORA) »
  - « write once , run Every where (WORE) »
- Grande richesse des APIs proposées :
  - interfaces graphiques pour clients lourds et riches, bases de données, réseau, XML, ...

Généralités sur Java

- Beaucoup de technologies associées :
  - Java EE, Java ME, Java Card, Java TV, Java 3D,
- Plate-forme évolutive :
  - Java 1.0, 1.1, Java 2, 1.3, 1.4, Java 5, Java 6, Java 7, Java 8
  - Java Community Process (JCP - jcp.org/) :
    - coordonne l'évolution du langage et des technologies associées, spécifications,

Généralités sur Java

- Java : langage interprété qui utilise une machine virtuelle
  - Machine virtuelle : simulation d'un ordinateur réel
- La "compilation" (en fait traduction en Bytecode) d'un code source Java (.java) permet d'obtenir un code exécutable (le bytecode) par la machine virtuelle Java. Ce bytecode (.class) est indépendant de la plate-forme
- La machine virtuelle Java est capable de comprendre les instructions contenu dans le bytecode et de les exécuter effectivement dans l'environnement réel courant.

Généralités sur Java

Comparé au C++ (compilé)

- C++ : langage compilé
  - la compilation du source permet d’obtenir des instructions natives exécutables.
  - le code exécutable obtenu est très performant
  - Peut ne pas s'exécuter si on change d'environnement (matériel, SE)
    - Faut gérer ces contraintes pendant la phase de programmation

Outils pour développer en Java choix 1/3

- Le Java (Software) Development Kit (JDK)

(http://www.oracle.com/technetwork/java/)

référence pour développer en Java.

comporte tous les outils indispensables à la réalisation et l'exécution d'un programme Java

totalement gratuit

ne comporte pas de véritable environnement de

développement.

Outils pour développer en Java choix 1/3

Le Java (Software) Development Kit (JDK)

- http://www.oracle.com/technetwork/java/
- Version 8 (JDK8) . Comporte JRE, "compilateur " (javac), et lanceur d'application/exécution (Java)

- référence pour développer en Java.
- comporte tous les outils indispensables à la réalisation et l'exécution d'un programme Java
- totalement gratuit
- ne comporte pas de véritable environnement de

développement.

Outils pour développer en Java choix 2/3

Freeware ou shareware

- offrent une interface graphique au JDK
- Ajoutent des outils supplémentaires (générateurs d'interface, …)
  - exemple : Eclipse, NetBeans

Outils pour développer en Java choix 3/3

Environnement de développement professionnel

- très puissants
- très chers…

  – exemples: JBuilder, JCreator

Installer JDK : Vérifiez

- Si vous avez déjà installé java

Vous pouvez vérifier les versions du JRE et de la JDK en utilisant les commandes suivantes :

java -version ==> version de la JRE

javac -version ==> version JDK

Les versions des la JRE et de la JDK **doivent être les mêmes** (sinon vous pourrez avoir une erreur de version plus tard) .

Installer JDK : Vérifiez

- S' il y a plusieurs versions de Java installées :

sur Ubuntu :

Vous pouvez afficher la liste de toutes les versions de JRE installés et en choisir un avec la commande suivante :

```
sudo update-alternatives --config java
```

Vous pouvez afficher la liste de toutes les versions de JDK installés et en choisir un avec la commande suivante :

```
sudo update-alternatives --config javac
```

Installation de la JDK8

http://www.oracle.com/technetwork/java/

Mettre à jour les variables d'environnement (Linux , windows)

https://docs.oracle.com/cd/E19182-01/820-7851/inst_cli_jdk_javahome_t/

https://docs.oracle.com/javase/tutorial/essential/environment/paths.html

Pour Tester l'installation de la JDK

Ouvrez un éditeur de texte (notepad) et saisir le code suivant :

```
/**
 * Premier programme Java pour afficher Bonjour.
 *
 */
class Bonjour {
    public static void main(String[] args) {
        System.out.println("Bonjour!"); // affiche la chaîne Bonjour.
    }
}
```

Pour Tester l'installation de la JDK

- Enregistrez le fichier source dans un répertoire avec le nom Bonjour et l'extension .java
  - Bonjour.java
- Ouvrir un terminal , se placer dans le repertoire contenant le fichier source
  - Compiler : javac Bonjour.java
    - Vérifier l'existence d'un fichier Bonjour.class
  - Exécuter : java Bonjour

Installation Netbeans

- Installer Netbeans 8 JSE
  - https://netbeans.org/downloads/

Notions de programmation orientée objet

- Classe, encapsulation
- Objet/instance
- Héritage
- polymorphisme

Notions de programmation orientée objet

Qu’est ce qu’une classe d’objets ?

- Classe :
  - généralisation de la notion de type
  - Définition des caractéristiques communes (données et comportements) d'un ensemble d’éléments similaires.==> classe pour décrire les mammifères.
  - Regroupe les données/attributs et les méthodes/comportements qui manipulent ces données ⇒ encapsulation

Notions de programmation orientée objet

- Encapsulation :
  - « En programmation, l’encapsulation désigne le principe de regrouper des données brutes avec un ensemble de routines permettant de les lire ou de les manipuler. Ce principe est souvent accompagné du masquage de ces données brutes1 afin de s’assurer que l’utilisateur ne contourne pas l’interface qui lui est destiné. L’ensemble se considère alors comme une boîte noire ayant un comportement et des propriétés spécifiés. » wikipedia
  - oblige le monde extérieur à utiliser les interfaces/méthodes spécifiées pour manipuler les données, sans savoir comment l'objet a été implémenté
  - ⇒ conséquence : L’implémentation peut être modifiée sans changer le comportement extérieur de l’objet. Cela permet donc de séparer la spécification du comportement d’un objet, de l’implémentation pratique de ces spécifications.
  - ⇒ sécurité car contrôle de l'accès aux données

Notions de programmation orientée objet

Qu’est ce qu’une classe d’objets ?

- Classe = regroupement d’objets similaires (appelés instances)
  - Toutes les instances d’une classe ont les mêmes attributs et opérations
    - Abstraction des caractéristiques non communes
- Classes sans instance
  - Classes abstraites :
    - Certaines (au moins une) opérations peuvent être abstraites/virtuelles (non définies)
  - Interfaces :
    - Pas d’attribut et toutes les opérations sont abstraites/virtuelles

Notions de programmation orientée objet

- Qu’est-ce qu’un objet ?
  - Une instance/exemplaire d'une classe (des valeurs spécifiques pour les attributs)
  - Objet = Etat + Comportement + Identité
- Etat d’un objet :
  - Ensemble de valeurs décrivant l’objet
  - Chaque valeur est associée à un attribut (propriété)
  - Les valeurs sont également des objets (⇒ liens entre objets)
- Comportement d’un objet :
  - Ensemble d’opérations que l’objet peut effectuer
  - Chaque opération est déclenchée par l’envoi d’un message
    - Exécution d’une méthode

Notions de programmation orientée objet

- Qu’est-ce qu’un objet ?
  - Objet = Etat + Comportement + Identité
- Identité d’un objet :
  - Permet de distinguer les objets indépendamment de leur état
  - 2 objets différents peuvent avoir le même état
  - Attribuée implicitement à la création de l’objet
  - L’identité d’un objet ne peut être modifiée
  - NB : identité est différente du nom de la variable qui référence l’objet

Notions de programmation orientée objet

Relation de spécialisation/généralisation entre classes

- Une classe A est une spécialisation d'une classe B si tout attribut/opération
  - de B est également attribut/opération de A
- Implémentation par héritage

Notions de programmation orientée objet

Héritage :

- permet la ré-utilisabilité et l’adaptabilité des objets ⇒ gain de productivité
- est basé sur des classes "filles" qui héritent des caractéristiques de leur(s) "mère(s)".
- Une classe filles possède les mêmes caractéristiques que ses classes mères et peut bénéficier de caractéristiques supplémentaires à celles de ces classes mères. Bien sur, toutes
- les méthodes de la classe héritée (fille) peuvent être redéfinies ⇒ autre comportement.
- Une classe fille peut éventuellement devenir à son tour classe mère ⇒ si le programmeur n’a pas défini de limitation

Notions de programmation orientée objet

Polymorphisme :

- dérive directement du principe d'héritage.
- un objet va hériter des attributs et méthodes de ses ancêtres. Mais un objet garde toujours la capacité de pouvoir redéfinir une méthode.
- concept de polymorphisme ⇒ choisir en fonction des besoins, quelle méthode ancêtre appeler, et ce au cours même de l'exécution.
  - Choix de la méthode non définie de manière statique mais en cours d'exécution en fonction du type|classe.

Définition d'une classe

```
public class NomDeLaClasse {
    déclarations des champs (attributs, propriétés)
    déclarations des méthodes
}
```

Définition d'une classe

```
public class NomDeLaClasse {
  //déclarations des champs (attributs, propriétés)
   [public,private(defaut),protected] [static] [final] Type|classe nom att_1 ;
   [public,private,protected][static] [final] Type|classe nom att_2 ;
   ----------------------------------------------------------------
   [public,private,protected][static] [final] Type nom att_n ;
  //déclarations des méthodes
  [public,private,protected] [static]  [Type|classe] nomMethode_1 ([paramètres] ){
   //instructions } ;
    -------------------------------------------------------------------------------------
  [public,private,protected] [static] [Type|classe]  nomMethode_n ([paramètres] ){
   //instructions } ;

}
```

Définition d'une classe

```
public class Fraction {
    // champs
public int num ;
public int den ;
   // méthodes
public void afficher () {
        if (this.num % this.den == 0)
              System.out.println(this.num/this.den) ;
        else
              System.out.println(this.num+”/”+this.den) ;
     }
}
```

Définition d'une classe

```
public class Fraction {
    // champs
public int num ;
public int den ;
    // méthodes
public void afficher () {
        if (this.num % this.den == 0)
            System.out.println(this.num/this.den) ;
        else
            System.out.println(this.num+"/"+this.den) ;
    }
}
```

Définition d'une classe

- Pointeur interne/auto-référence

Il peut se révéler indispensable pour un objet de pouvoir se référencer lui-même.

Pour cela, toute instance dispose d’un pointeur interne vers elle-même ⇒ this

Définition d'une classe

- Pointeur interne/auto-référence : this
  - est une pseudo-variable qui permet :
    - de faire référence à l’objet courant (celui qu’on est en train de définir).
    - ou de désigner ses attributs ou ses méthodes.

Définition d'une classe

- this est une pseudo-variable qui permet :

  - de faire référence à l'objet courant (celui qu'on est en train de définir).
  - ou de désigner ses attributs ou ses méthodes.
  - est généralement utilisé pour lever l'ambiguïté sur un identificateur lorsque le paramètre d'une méthode porte le nom de l'attribut de l'objet qu'il est chargé de modifier.==> exemple

Définition d'une classe

Pointeur interne/auto-référence : this

Exemple :

```
void fixerNumerateur (int num) {
        this.num = num ;
}
```

Définition d'une classe

Utilisation d’une classe

```
public class FractionUtil {
    public static void main (String[] args) {
        Fraction f = new Fraction();
        f.num = 8; f.den = 5;
        f.afficher();
}
}
```

- Instanciation avec l’opérateur new
- Java s’occupe de la destruction des objets non utilisés

(mécanisme de garbage collector). pas besoin de “destructeurs”.

Définition d'une classe

- Différents types de méthodes

quelques méthodes particulières à la Programmation Orientée Objet.

- Constructeur
  - sert à construire l’objet en mémoire. Réserve un emplacement en mémoire pour le nouvel objet,
  - initialise les attributs de cet objet, c'est-à-dire leur donner une valeur initiale, et rendre cet objet.
  - se charge de mettre en place les données, d’associer les méthodes avec les attributs et de créer le diagramme d’héritage de l’objet, de mettre en place toutes les liaisons entre les ancêtres et les descendants
- En Java, les constructeurs portent exactement le nom de la classe qui les contient.
- Chaque constructeur possède une liste de paramètres, éventuellement vide.

Définition d'une classe

- Destructeur
  - Un destructeur est une procédure particulière dont le rôle est de détruire un objet existant.
  - Les destructeurs ont principalement deux tâches :
    - libérer les ressources utilisées par l'objet (par exemple, fermer un fichier ou une connexion réseau) et
    - libérer l'emplacement réservé en mémoire pour l'objet.
  - En Java, il n'existe pas de destructeur car la libération de la mémoire est gérée automatiquement ⇒ ramasse-miettes/ garbage collector ⇒ on peut l'appeler direcement avec :
    - java.lang.System.gc(); //public static void gc()
  - Un ramasse-miettes (en anglais : garbage collector) est un dispositif qui libère automatiquement la mémoire des objets qui ne sont plus accessibles.

Définition d'une classe

- méthode main()

Premier sous-programme exécuté au lancement d’une application Java.

- Signature:

```
public static void main (String[] args) {
...
}
```

- Le paramètre est un tableau de chaînes utilisé pour les arguments transmis à partir de la ligne de commande.

Définition d'une classe

- méthode main()

```
public class Facture
{
public static void main(String[] args)
{
    int x = Integer.parseInt(args[0]);
    int y = Integer.parseInt(args[1]);
   System.out.println("la somme de "+x+" et"+y+" est: "+(x+y));
}
}
```

Définition d'une classe

- méthode main()
  - Utilisation argument ligne de commande : sur un terminal

    java Facture 12 14

    Sortie ==> la somme de 12 et 14 est: 26

  - Avec un IDE
    - Netbeans :
      - projet actif (Run ⇒ set main project ==> choisir son projet) ; ensuite Run ⇒ Set project configuration ⇒ Customize et saisir les arguments séparés par des espaces dans le champs Arguments. Compiler et exécuter le projet, pas le fichier
    - eclipse
      - Run Configurations ⇒ onglet Arguments pour préciser les arguments

Définition d'une classe

- Accesseur
  - est une méthode qui permet d'accéder en lecture ou en écriture à la valeur d'un attribut.==> getters/setters
  - Par convention :
    - le rôle des accesseurs se limite à rendre, pour ceux en lecture, ou à modifier, pour ceux en écriture, la valeur de l'attribut concerné, et rien de plus.
    - sont des méthodes comme les autres en Java.
    - ils portent le nom de get ou set suivi du nom de l'attribut concerné.

Définition d'une classe

- Redéfinition
  - technique qui permet, dans une classe fille, de donner une nouvelle définition d'un membre déjà présent dans une classe mère.
  - elle permet notamment de donner une implémentation plus précise des méthodes dans les classes spécialisés (qui ont hérité).
  - Par exemple, si on a un classe Parallélogramme qui définit une méthode aire, alors la classe Carré, qui hérite de Parallélogramme, peut redéfinir aire pour lui donner un code plus adapté.

Définition d'une classe

- Surcharge
  - La surcharge est la capacité que possède une classe d'avoir plusieurs membres portant le même nom.
  - Tous les langages à objets n'autorisent pas la surcharge, considérée comme très pratique par certains et source de problèmes par d'autres.
  - Java :
    - dispose de la surcharge notamment pour les méthodes et les constructeurs
    - lorsque deux méthodes ou deux constructeurs portent le même nom, c'est le nombre et le type de leurs paramètres (signature) qui permettent alors de les différencier
    - ==>les constructeurs portent tous forcément le même nom

Packages prédéfinies

- Les classes sont regroupées par thèmes dans des packages. Exemples :
- java.lang : types de données et éléments de base du langage (classes Object, String, Boolean, Integer, Float, Math, System (fonctions système), Runtime (mémoire, processus), etc.)
- java.util: structures de données et utilitaires (Dates, Collections, etc.)
- java.io: bibliothèque des entrées / sorties, fichiers.
- java.net: bibliothèque réseau
- java.awt: librairie graphique
- java.applet: librairie des applets.
- java.security: sécurité (contrôle d'accès, etc.)
- java.rmi: applications distribuées.
- java.sql: accèes aux BD (JDBC)

Etc….

SYNTAXE DE BASE

Les commentaires

- Les commentaires tiennent sur une ligne :

// tout le reste de la ligne est un commentaire

- Les commentaires sont multilignes :

/* ceci est un commentaire

tenant sur deux lignes

***/**

- Les commentaires sont destinés au système javadoc pour la génération d'une documentation API à partir du code :

/ ceci est un commentaire spécial destiné au**

système JavaDoc */

SYNTAXE DE BASE

Les identificateurs

- Recommandations de SUN : Java Code Conventions
  - http://www.oracle.com/technetwork/java/codeconv-138413.html

- Le nom d'une classe ou d'une interface est fait d'un mot ou de la concaténation de plusieurs mots. Chacun commence par une majuscule : Etudiant, EtudiantBoursier, etc.
- Les noms des méthodes et des variables commencent par une minuscule. Lorsqu'ils sont formés d'une concaténation de mots, chacun, sauf le premier, commence par une majuscule : afficher, toString(), nom, numeroInscription, etc.
- Les constantes de classe (variables static final) sont écrites entièrement en majuscules : Integer.MAX_VALUE
- Java ⇒ sensible à la casse (Myvar est différent de myvar)

SYNTAXE DE BASE

Les types de données primitifs

- Entiers : signé
  - byte (8 bits) -128 à 127 n bits, signé ==> $[-2^{n-1}, 2^{n-1}-1]$
  - short (16 bits)
  - int (32 bits)
  - long (64 bits)
- Réels/flottants : norme IEEE 754
  - float (32 bits)
  - double (64 bits)
- Caractère : Unicode
  - char (16 bits)
- Booléen : (32bits)
  - boolean : true | false

SYNTAXE DE BASE

Les types de données primitifs

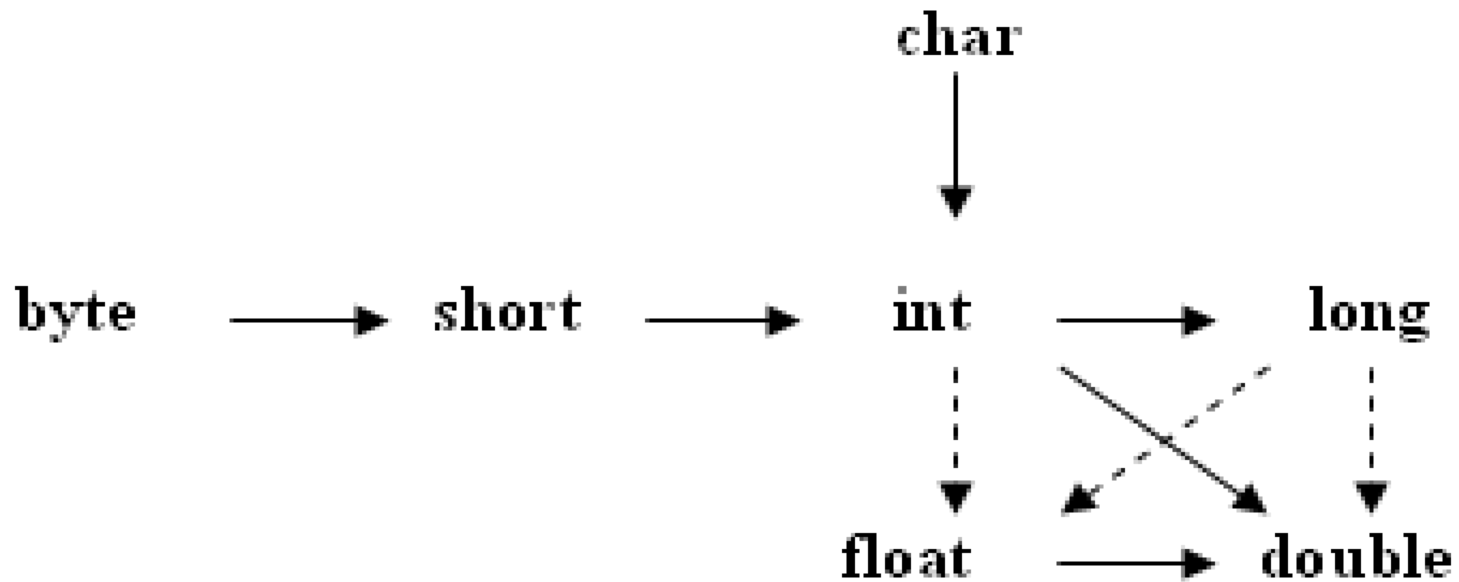
- Possibilité de faire du casting comme en C :

```
double d; int i;
i = (int) d; // troncation de d en un entier
```

SYNTAXE DE BASE

http://imss-www.upmf-grenoble.fr/prevert/Prog/Java/CoursJava/TypeDeDonneePrimitifs.html

- Fleche pleine : conversion automatique sans perte d'information
- Pointillée : conversion automatique avec possible perte d'information



SYNTAXE DE BASE

Les classes enveloppes

- Les données de types primitifs (byte, short, int, long, float, double, char et boolean), ne sont pas des objets en Java.
- A chaque type primitif, correspond une classe dite «classe enveloppe» : Byte, Short, Integer, Long, Float, Double, Character et Boolean ⇒ classe d'où nom commence par une majuscule
- Chaque instance d’une de ces classes se compose d’une unique donnée membre qui est une valeur du type primitif que l’instance enveloppe.
- Une instance dispose d'un ensemble de méthodes comme ==>

SYNTAXE DE BASE
Les classes enveloppes

- Exemple de méthodes pour Integer (voir doc pour les autres type enveloppes)

  Integer v1 = new Integer (4) ; Integer v2=new Integer(5) ;

  byte k= v1.byteValue() ; // retourne la valeur enveloppée par v sous forme de byte

  int p= v1.intValue() ;   //retourne la valeur enveloppée par v sous forme de int

- Autres méthodes : v1.longValue(), v1.floatValue(), ...
- Méthode : v1.compareTo(v2) ; // 0 si égalité, négatif si v1 <V2, positif sinon
- Méthode v1.toString() pour afficher (appelér par println)

SYNTAXE DE BASE

Les classes enveloppes

- Exemple de méthodes pour Integer ⇒ methodes statiques
  - public static int parseInt(String s) throws NumberFormatException
  - String p= "123 " ; int a=Integer.parseInt (p) ; ⇒ a=123
  - public static int max(int a, int b) ⇒ retourne le max entre a et b
    - comme Math.max
  - public static int min(int a, int b) ⇒ retourne le min entre a et b
    - comme Math.min
- Constantes statiques
  - Integer.MIN_VALUE ⇒ valeur max qu'un int peut contenir
  - Integer.MAX_VALUE ⇒ valeur min qu'un int peut contenir
  - Integer.SIZE ⇒ nombre de bits pour représenter un int
  - Etc..

SYNTAXE DE BASE

Les classes enveloppes

- Double :
  - Constantes statiques :
    - Double.MAX_VALUE,Double.MIN_VALUE,…
    - Double.NEGATIVE_INFINITY ⇒ - l'infini
    - Double.POSITIVE_INFINITY ⇒ + l'infini ⇒ valeur obtenue lorsque /0
    - Double.NaN ⇒ constante signifiant Not a Number
    - Etc...
  - Méthodes statiques : voir doc pour signature précise
    - Double.parseDouble (), Double.max(),Double.sum(),…
- Float, Byte,Character,….
  - Voir doc

SYNTAXE DE BASE

Les classes enveloppes

- Les opérations sur les réels ne lèvent pas d'exceptions mais produisent des valeurs comme NaN, Infinity,…

```
float p = 5000f;

System.out.println(p/0); // affichage de Infinity

p = p*p*p*p;

System.out.println(p);   // affichage de 6.2500002E14

p = p*p*p*p;

System.out.println(p);   // affichage de Infinity

System.out.println(p/p); // affichage de NaN  ⇒ infini/infini
```

SYNTAXE DE BASE

Emballage et Deballage pour les classes enveloppes

- Emballage : opération consistant à construire un objet enveloppant une valeur v d’un type primitif

```
int v;
...
Integer k = new Integer(v);      // emballage de v
```

- Déballage : opération inverse.

```
...
v = k.intValue();                // déballage de v
```

SYNTAXE DE BASE

Les types enveloppe

- Java (depuis version 5) permet de confondre le type primitif et un objet du type enveloppe ⇒ les types sont déduits selon le contexte
- ⇒ Integer n =5 <==> Integer n = new Integer(5) ;
- ⇒ n = n + 5; ⇒ n est considéré comme un int
- ⇒ String s = n.toString(); ⇒ n est considéré comme un objet . System.out.println(s);
- if (n == 10) … <==> if (n.intValue() == 10) ...

SYNTAXE DE BASE

- Initialisation avec des valeurs fixes (littéraux) :

  exemple : pour initialiser des variables locales

```
boolean test = true;

byte b = 20;

short s = 45;

int i = 100;

char c1 = 't', c2='K';
```

SYNTAXE DE BASE

- Initialisation avec des valeurs fixes (littéraux) :
  - Entier
    - Un litteral entier qui n'est pas de type long est de type int

      Exemple : byte a=12 ; short b=23 ; int k=15 ;
      - Les variables ci-dessus sont tous des int
      - Pour déclarer un long ⇒ faire suivre par L ou l
        - Ex : long k=24L ; long c=75l ;
        - Préférer L au l   car parfois l ⇒ confusion avec i

SYNTAXE DE BASE

- Initialisation avec des valeurs fixes (littéraux) :
  - Entier
  - On peut écrire dans une base décimale, hexadécimale, binaire

```
// Le nombre 26, en decimal
int decVal = 26;
//  Le nombre 26, en hexadecimal  ⇒ préfix (zeroX)    0x
int hexVal = 0x1a;
// Le nombre 26, en binaire  ⇒ prefix              0b
int binVal = 0b11010;
```

SYNTAXE DE BASE

- Initialisation avec des valeurs fixes (littéraux) : les réels

  un réel est float (suffixe F ou f) ou bien c'est un double (et peux optionnellement avoir le suffix d ou D)

  - Type float :
    - Ex : float f=1.45f ;
    -
  - Type double (par défaut les réels sont des double)
    - double d=1.34d ; double k=1.67 ;
    - double d1 = 123.4;
    - double d2 = 1.234e2; // même valeur que d1 en écriture scientifique

SYNTAXE DE BASE

- Initialisation avec des valeurs fixes (littéraux) : caractères et chaines

  - Type char :
    - Ex : char k='c' ;

  - Type chaine
    - String p="langage java" ;

SYNTAXE DE BASE

Les types enveloppe

- Exercice

Écrire une classe avec main dans lequel vous créez deux objets de chaque type enveloppe, une variable du type primitif avec une valeur initiale, afficher les valeurs des objets , comparer les valeurs des deux objets, comparer un objet avec la variable du type primitif, utiliser deux méthodes statiques de chaque classe enveloppe.

SYNTAXE DE BASE

Les types enveloppe

- Test classe enveloppe

```
public static void main(String[] args) {
    Integer v1=new Integer(4), v2=4;
      int v3=5;
      Double d1=new Double(2.5),d2=4.2;
// par default les constantes sont Double. Pour en faire un float==> float k=2.7F;
       int k=v1.byteValue();
       //Double b; b.
      if (v1.compareTo(v2)==0)
       System.out.println("egal");
      if (v1.compareTo(v3)<0)
       System.out.println("v1 inferieur v3");
       if (d1.compareTo(d2)>0)
       System.out.println("v1 superieur v3");
       int pp=v1+v3;
       System.out.println(pp);
       char m1='b';
      Character c=new Character('a');
      System.out.println(c);
       System.out.println(m1);
       System.out.println(c.charValue());
   }
```

SYNTAXE DE BASE : Les opérateurs : priorité décroissante

- Postfixé : expr++ expr--
- Préfixé/Unaire : ++expr --expr +expr -expr ~ !
  - ! ⇒ non logique ⇒ inverse une valeur logique (! false ⇒ true)
  - ~ ⇒ complément à 1
- Multiplicatif : * / %
  -
- Additif : + -
  - + ⇒ sert aussi à concaténer des chaînes

SYNTAXE DE BASE : Les opérateurs : priorité décroissante

- Décalage : << >> >>>
  - >>> : Ajoute un zéro à gauche pour un bit décalé (non signé)
  - >> : prise ne compte signe
- relationnel : < > <= >= instanceof
- Égalité : == !=
- Et binaire (bit à bit): &
- Ou exclusif binaire : ^
- OU inclusif binaire : |

SYNTAXE DE BASE : Les opérateurs : priorité décroissante

- ET Logique (cour-circuit) : &&
  - Court-circuit ⇒ évaluation 2ieme membre seulement si nécessaire
- OU Logique (cour-circuit) : ||
- Opérateur ternaire : ? :

```
int result, s=3;
result = (s>2 ) ? 5: 8;
System.out.println(result);
```

- Affectation : = += -= *= /= %= &= ^= |= <<= >>= >>>=

SYNTAXE DE BASE : Les opérateurs : priorité décroissante

- Priorité descendante
- Même ligne = même priorité ⇒
  - Opérateurs binaires autres que affectation ⇒ associativité gauche ⇒ droite
  - Opérateurs affectation ⇒ associativité droite ⇒ gauche

SYNTAXE DE BASE : Les fonctions mathématiques

- Méthodes statiques de la Classe Math ⇒ voir doc
  - java.lang.Math.abs () ⇒ valeur absolue
  - java.lang.Math.exp() ⇒ exponentiel
  - java.lang.Math.max() ⇒ max entre deux valeurs
  - java.lang.Math.sqrt() ⇒ racine carrée
    - public static double sqrt(double a)
  - Etc..
- Constantes :
  - Approx de Pi ⇒ java.lang.Math.PI ⇒ double
  - Approx de E (base logarithme népérien) ⇒ java.lang.Math.E ⇒ double

SYNTAXE DE BASE

La classe String

- String ⇒
  - Chaîne de caractères . Index début =0
  - Méthodes :

    String ch1="abcd",ch2="bcd";

    char c1=ch1.charAt(1); // retourne le caractère à l'index en argument ; 1 dans cet exemple

    int test=ch1.compareTo(ch2); //comparaison ordre lexicographique . Retourne 0 si égal, négatif si ch1< argument, positif sinon

    int taille=ch1.length(); // taille de la chaîne

  - Etc...

Entrées/sorties

- Methode de la classe javax.swing.JoptionPane
- JoptionPane.showInputDialog(Object message) ⇒ String
  - Plusieurs déclinaisons ⇒ voir doc
  - Exemple
    - String s = JOptionPane.showInputDialog("Entrez une chaîne: ");
- JOptionPane.showMessageDialog(Component parentComponent, Object message) ⇒ void
  - Voir doc + autres déclinaisons
  - Exemple
    - JOptionPane.showMessageDialog(null, "test") ;

Entrées/sorties

- Methode de la classe   javax.swing.JoptionPane
- JoptionPane. showConfirmDialog(Component parentComponent, Object message) ⇒ int
  - Plusieurs déclinaisons ⇒ voir doc
  - Exemple
    - int reponse = JOptionPane.showConfirmDialog(null, "Voulez-vous quitter ?");
    - Valeur de reponse (0,1,2) ⇒ dépend du bouton sur lequel on a cliqué (OUI,NON,ANNULER)
      - Ces valeurs correspondent aussi à des constantes :
      - JOptionPane.YES_OPTION ⇒ 0
      - JOptionPane.NO_OPTION ⇒ 1
      - JOptionPane.CANCEL_OPTION ⇒ 2

Entrées/sorties

- Exemple

```
public static void main(String arg[]){
String ch = null;
int sommeLongueurs = 0;  int reponse ;
do {
ch = JOptionPane.showInputDialog("Entrez une chaîne: ");
JOptionPane.showMessageDialog(null, ch + ": "+ ch.length());
sommeLongueurs = sommeLongueurs + ch.length();
reponse = JOptionPane.showConfirmDialog(null, "Voulez-vous
quitter ?");
} while (reponse == JOptionPane.NO_OPTION);
JOptionPane.showMessageDialog(null, " La somme des longueurs
est : "+ sommeLongueurs);
}
```

Entrées/sorties

- La classe java.util.Scanner
  - permet de lire un flux de données formatées.
  - Offre des méthodes pour lire un flux de données formatées :
    - int nextInt() : lecture de la prochaine valeur de type int.
    - long nextLong() : lecture de la prochaine valeur de type long.
    - float nextFloat() : lecture de la prochaine valeur de type float.
    - double nextDouble() : lecture de la prochaine valeur de type double.
    - String nextLine() : lecture de tous les caractères se présentant dans le flux d’entrée jusqu’à une marque de fin de ligne et renvoi de la chaîne ainsi construite. La marque de fin de ligne est consommée, mais n’est pas incorporée à la chaîne produite.
    - String next() : sans paramètre, retourne le prochain motif sous forme de chaîne
      - La forme avec paramètre (une expression régulière) permet de retourner le prochain motif qui correspond à l'exp régulière

Entrées/sorties

- La classe java.util.Scanner
  - Offre aussi toute une série de méthodes booléennes pour tester le type de la prochaine donnée disponible, sans lire cette dernière :
  - boolean hasNext() : Y a-t-il une donnée disponible pour la lecture
  - boolean hasNextLine() : Y a-t-il une ligne disponible pour la lecture
  - boolean hasNextInt() : La prochaine donnée à lire est-elle de type int
  - boolean hasNextLong() : La prochaine donnée à lire est-elle de type long
  - boolean hasNextFloat() : La prochaine donnée à lire est-elle de type float
  - boolean hasNextDouble() : La prochaine donnée à lire est-elle de type double

Entrées/sorties

- La classe java.util.Scanner

```
java.util.Scanner entree = new java.util.Scanner(System.in);
System.out.println("Donner votre prenom et votre nom:");
String prenom = entree.next(); String nom = entree.next();
System.out.println("Donner votre age:");
int age = entree.nextInt();
entree.nextLine();
System.out.println("Donner une phrase:");
String phrase = entree.nextLine();
System.out.printf("%s %s, %d ans, dit %s\n", prenom, nom, age,
phrase);
```

Entrées/sorties

- La classe java.io.PrintStream contient une méthode printf qui fonctionne presque de la même façon qu'en C.

- Exemple :

```
int degre = 28; float k=22.8F ;

System.out.printf("Il fait %d degres  et k=%f \n", degre,k);

// %d indicateur de format pour des entiers

System.out.printf("Le %2$s %1$s", "Java", "langage");

// le 2$ du %2$s indique qu'il s'agit d'ecrire le 2nd paramètre de susbstitution

// le s indique qu'il s'agit d'une chaîne de caractères.
```

- Comme en C on utilise des indicateurs de format commençant par

le symbole %. (Consulter doc de la classe java.util.Formatter)

Structures de contrôle

- If
  - If (condition) { ...instructions..}
  - If (condition) { ...instructionsA..} else { ...instructionsB..}
    - Généralisation ; suite de If …..else

- switch

Structures de contrôle

- If (condition) { ...instructions..}
  - Le block d' instructions est exécuté si condition est évaluée à true
  - Exemple :

```
--------------
int a =10;
   if (a==15){
      System.out.println("la valeur de a egale 15");
   }
 System.out.println("la valeur de a="+a) ;
--------------
```

Structures de contrôle

- If (condition) { ...instructionsA..} else { ...instructionsB..}
  - Le block A est exécuté si condition est vraie, le block B est exécuté si condition est fausse. Un seul des deux blocks est exécuté.
  - Exemple

```
int a =10;
  if (a==15){
    System.out.println("la valeur de a egale 15");
  }
  else {
    System.out.println("la valeur de a est différente de 15");
  }
 System.out.println("la valeur de a="+a);
```

Structures de contrôle

Généralisation : Suite de If .....else

```
int a =10;
   if (a==15){
      System.out.println("la valeur de a egale 15");
   }
   else if (a==20){
      System.out.println("la valeur de a=20");
   }
   else if (a==11){
       System.out.println("la valeur de a=11");
   }
   else
   {
       System.out.println("la valeur de a!= de 15, de 20 et de 11");
   }
 System.out.println("la valeur de a="+a);
```

Structures de contrôle

- Switch

```
int chiffre = 8;
String enLettre;
switch (chiffre) {
    case 1:  enLettre= "un";
             break;
    case 2:  enLettre = "deux";
             break;
    case 3:  enLettre = "trois";
             break;

    default: enLettre = "différent de un, deux, trois";
             break;
}
 System.out.println(enLettre);
```

Structures de contrôle

- Switch
  - L' Expression évaluée par switch peut être de type :
    - byte, short, char, int
    - énuméré
    - String
    - Classe enveloppe : Character, Byte, Short, and Integer

Exercices Application structures de contrôle

Exercice 1 : Écrire un programme réalisant la facturation d'un article livré en un ou plusieurs exemplaires. On fournira en données le nombre d'articles et leur prix unitaire hors-taxe. Le taux TVA sera toujours de 18.6 %. Si le montant TTC dépasse 1000 FCFA, on établira une remise de 5 %.

Exercice 2 :

a) Écrire un programme qui permet de lire la moyenne d'un candidat (faire un contrôle de saisie pour que la moyenne soit>= 0.0 et <=20.0) et de le déclarer "ajourné" si la moyenne est strictement inférieure à 8, "convoqué à l'oral" si la moyenne est dans l'intervalle 8 (compris) et 10 (non compris), ou "admis" si la moyenne est supérieure ou égale à 10.

b) Même question que a) mais cette fois les appréciations sont les suivantes : « ajourné » si la moyenne est <9.5 ; « admis » si le moyenne est >=9.5 ; « avec mention » si la moyenne >=12 ; « avec félicitation du jury » si la moyenne >=17 ;

Exercice 3 :

Écrire un programme qui permet de lire trois entiers et de déterminer s'ils sont tous différents (aucun n'est égal à un autre), ou bien si deux sont égaux et le troisième différent, ou s'ils sont tous égaux.

Exercices Application structures de contrôle

Exercice 4 : switch

Écrire un programme qui permet de saisir un nombre entre 1 et 12 (correspond à un mois, il faut faire un contrôle de saisie) et une année ; et qui affiche le nombre de jours que compte ce mois. Pour le mois de Février , c'est 29 si l’année est bissextile et 28 sinon. Une année est bissextile si elle est divisible par 4 et non divisible par 100 ou bien si elle est divisible par 400.

Exercice 5 :

Écrire un programme qui permet de saisir deux réels et un opérateur (‘+’, ‘-’, ‘*’, ‘ /’) et de calculer la valeur correspondante à l’opération (afficher ‘erreur’ lorsque l’opérateur saisi est inconnu).

Structures répétitives/boucles

- For

```
for (initialisation; terminaison;incrémentation) {
        instruction(s)
    }
```

- Exemple :

```
for(int i=1; i<5; i++){
        System.out.println("i= " + i);
    }
```

Structures répétitives/boucles

- For : forme pour parcours de Tableaux et collections :

```
public static void main(String[] args){
        int[] numbers = {1,2,3,4,5,6,7,8,9,10};
        for (int item : numbers) {
            System.out.println("valeur= " + item);
        }
    }
```

Structures répétitives/boucles

- While

```
while (expression) { ..instructions... }
```

- Répétition tant que expression est vraie
- Exemple

```
int a = 1;
while (a < 5) {
    System.out.println("a= " + a);
    a++;
}
```

Structures répétitives/boucles

- do ..while

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

- Test en fin (block exécuté au moins une fois).
- Répétition tant que condition vraie==> Arrêt si condition est fausse
- Exemple

```
int a = 1;
     do {
         System.out.println("a= " + a);
         a++;
     } while (a < 5);
```

Structures répétitives/boucles

Break, continue, return

- break
  - Dans une boucle for, while, do...while , switch⇒ sortie de la boucle interne qui le contient ⇒ une forme avec label ⇒ passe au label qui suit le mot clé
- rontinue ⇒ deux formes (avec label, sans label)
  - Dans une boucle passe à l'itération suivante (forme sans label) ou au label qui le suit
    - Dans une boucle for ⇒ passe directement à la partie mise à jour/incrémentation
    - Dans une boucle while/do...while ⇒ passe directement à la partie évaluation de l'expression de contrôle de la boucle.
- return
  - Sortie de la fonction courante (qui le contient) et donne la main à la fonction qui avait appelée la méthode courante
  - Sert aussi à retourner une valeur

Structures répétitives/boucles

- Break | continue

```
int i=0 ;
for (;i<13;i++)
{
  if(i==4) continue ;
  System.out.println("valeur= "+i) ;
 if (i==6) break ;
}
```

- ⇒ affiche 0 1 2 3 5 6 (n'affiche pas 4 , ni les i >6)

Structures répétitives/boucles

- Break | continue

```
int i=0 ;
while (i<13)
{
  if(i==4) {i++ ; continue ;}
  System.out.println("valeur= "+i) ;
 if (i==6) break ;
 i++ ;
}
```

- ⇒ affiche 0 1 2 3 5 6 (n'affiche pas 4 , ni les i >6)

Exercices applications structures répétitives

- Exercice 1 : afficher les entiers pairs entre 1 et N (saisi au clavier) en utilisant une boucle for, une boucle while, une boucle do...while
- Exercice 2 : boucles imbriquées :

  saisir un entier M > 2 (faire un contrôle de saisie) et saisir et afficher M notes correctes (une note N est correcte si N ≥ 0.0 et N ≤ 20.0). Écrire ce programme de plusieurs manières en utilisant des imbrications de boucles différentes (for/do..while ; do..while/do..while ; for/while,...
- Exercice 3 : break , continue

  Écrire un programme qui permet de saisir autant d'entiers qu'on souhaite, qui compte le nombre d'entiers strictement positifs saisi (à afficher à la sortie de la boucle) ; qui affiche les entiers strictement négatifs ; la saisie prend fin (on sort de la boucle) lorsque zéro est saisi ou bien .

Exercices applications structures répétitives

Exercice 1 : Écrire un programme qui permet d'afficher la lettre H avec un caractère '+' . La longueur (impaire et >=3) de la lettre et sa largeur sont saisies au clavier. Exemple avec Longueur = 7 (lignes) et largeur = 8 (caractères)

```
+      +
+      +
+      +
++++++++
+      +
+      +
+      +
```

Exercice 2 : Affichez la lettre E

Exercices applications structures répétitives

- Exercice : jeu du nombre caché

On considère le jeu du nombre caché, jeu dans lequel l'ordinateur choisit un nombre (H) au hasard entre 1 et 10 ; il faut ensuite, pour le joueur, le deviner en N essais au plus (N une constante que vous pouvez fixer à 3). A chaque fois que le joueur donne une réponse (R), l'ordinateur répond par :

« trop petit » si le nombre donné (R) est plus petit que celui à deviner (H)

ou « trop grand » si le nombre donné (R) est plus grand que celui à deviner (H)

ou « Vous avez trouvé le nombre exact » si c'est le nombre exact (c.-à-d. H = R).

Ecrire un programme Java qui permet de simuler le jeu du nombre caché.

On utilisera la fonction statique *random* de la classe *java.lang.Math* pour générer le nombre H de l'ordinateur.

Les échanges (entrées/sorties) avec le joueur se feront en utilisant les méthodes de la classe *javax.swing.JOptionPane*.

Les tableaux

- Les tableaux se comportent un peu comme des objets.

- Deux syntaxes pour la déclaration:

  - TypeDesElements[ ] NomDuTableau;
    - Forme encouragée par convention

  - TypeDesElements NomDuTableau[ ];

Les tableaux

- Exemple :

  int[ ] tab_entiers; ou int tab_entiers[ ];

  String[ ] tab_chaines;

- NB : Le type des éléments du tableau peut être une classe existante.

Les tableaux

- Instanciation

TypeDesElements[ ] NomDuTableau = new TypeDesElements[TailleDuTableau];

- Exemple

```
int[ ] tab_entiers = new int[15];
String[ ] chaines = new String[3];
```

Les tableaux

- Matrices

  TypeDesElements[ ][ ] NomDuTableau;

  TypeDesElements NomDuTableau[ ][ ];

- Exemple

  int [ ][ ] matrice

  ou int matrice [ ][ ];

- Instanciation

  Matrice = new int[3][4] ; ⇒ 3 lignes , 4colonnes

Les tableaux

- Exemple de saisie d'un tableau:

```
public static void main(String[] args) {
int[] t = new int[5];
Scanner entree = new Scanner(System.in);
for (int i=0; i<5; i++){
t[i] = entree.nextInt();
entree.nextLine();}
}
```

Les tableaux

- Taille tableau

```
int[] t = new int[5];
int taille=t.length ;   // différent de la méthode length() de la classe String
```

- Création + Initialisation

```
int[] t = new int[3];   t[0]=1 ; t[1]=14 ;  t[2]=21 ;
```

- Autre forme ⇒ créer et initialiser en même temps

Int[] t={1,14,21} ; ⇒ taille déduite du nombre de valeur séparées par des virgules entre les accolades

Les tableaux

- Création + Initialisation matrice

```
int[][] t = new int[2][2];
T[0][0]=1 ;
 t[0][1]=14 ;  etc..
```

- Autre forme : créer et initialiser en même temps

```
int[][] t={{1,2,3},{5,6}} ;
```

  - Cette forme permet un nombre de colonnes différent pour chaque ligne.

Les tableaux

- Exemple : saisie des valeurs d'une matrice

```
int k=2, h=2;
int[][] t2=new int[k][h];

----lecture------
Java.util.Scanner entree = new Scanner(System.in);
for (i=0; i<k; i++){
    for (j=0; j<h; j++){
        t2[i][j] = entree.nextInt();
        entree.nextLine();
}
```

Les tableaux

- Exemple : affichage contenu d'une matrice

```
int[][] t1={{1,2,3},{2,5}};
int i,j;
        ---------
        for (i=0;i<t1.length;i++)                // nombre de ligne
            for (j=0;j<t1[i].length;j++)          // taille de la ligne i
                System.out.println(t1[i][j]);
        ------
```

Les tableaux

- Exercice 1:

  déclarer un tableau de 10 caractères, saisir 10 caractères dans le tableau, compter et afficher le nombre de voyelle (a,e,i,o,y) qu'il y a dans le tableau.

- Exercice 2: Créer, de manière dynamique (avec l'opérateur new) un tableau d'entiers (la taille sera saisi au clavier) ; ensuite saisir taille entiers ; ensuite afficher tous les entiers pairs dans le tableau, calculer et afficher aussi la somme des entiers impairs du tableau.

- Exercice 3: créer dynamiquement une matrice d'entiers de 3 lignes 4 colonnes, saisir ses éléments , et afficher le contenu.

Les tableaux : recherche séquentielle dans un tableau: 1/3

```
package tp_ucao;
import java.util.Scanner;

public class TestTableau {
    public static void main(String[] args) {
      int[] t;
      int taille;

      Scanner entree = new Scanner(System.in);
        //-----saisie taille tableau-----
        System.out.println("Donnez la taille du tableau");
       taille=entree.nextInt();
       //-----création dynamique du tableau----
         t=new int[taille];
     //-----saisie des éélements du tableau----
      int i=0;
      for (; i<taille; i++)
     {
           System.out.print("Donnez t["+i+"]=");
            t[i] = entree.nextInt();
              entree.nextLine();
      }
```

Les tableaux : recherche séquentielle dans un tableau: 2/3

```
// recherche séquentielle
  // saisie de l'entier à rechecher dans le tableau
   System.out.println("Donnez la valeur à rechercher");
        int valRecherche = entree.nextInt();

  // début recherche avec une boucle for  :  dés qu'on trouve on arrête la recherche
      boolean trouve=false;
     for (i=0; (i<taille) && (! trouve); i++)
     {
       if(t[i]==valRecherche){trouve=true;}
    }
   if (trouve)  // equivalent à if(trouve==true)
   {
       System.out.println(valRecherche+" est dans le tableau");
  }
  else
  {
      System.out.println(valRecherche+" n'est pas dans le tableau");
 }

}
}
```

Les tableaux : recherche séquentielle dans un tableau: 3/3

```
// recherche séquentielle  ⇒ autre forme de la boucle for dans booleen
  // saisie de l'entier à rechecher dans le tableau
   System.out.println("Donnez la valeur à rechercher");
        int valRecherche = entree.nextInt();

      // début recherche avec une boucle for : autre forme avec break sans boolean

 for (i=0; ; i++)
 {
     if(t[i]==valRecherche){ break; }
 }
 if (i<taille)
  {
      System.out.println(valRecherche+" est dans le tableau");
  }
 else
 {
     System.out.println(valRecherche+" n'est pas dans le tableau");
 }

}
}
```

Les tableaux : recherche séquentielle dans un tableau

Exercice 1:
refaire la partie recherche séquentielle avec une boucle while en utilisant dans un premier un booléen pour arrêter la recherche dés qu'on trouve ce qui est recherché ; et dans un deuxième temps utiliser une boucle while , sans un booléen mais en utilisant l'instruction break pour sortir de la boucle dés qu'on trouve.

Exercice 2: reprendre l'exercice précédent en utilisant une boucle do...while

Exercice 3 :
créer et saisir un tableau de chaînes de caractères (des prénoms), saisir une chaîne au clavier et rechercher si elle se trouve dans le tableau.

Les tableaux

- Exercice : carré magique

  saisir la valeur de la taille n d'une matrice , créer une matrice d'entiers de n lignes et n colonnes et saisir la matrice, afficher la diagonale, calculer la somme des éléments de la ligne i, la somme des éléments de la colonne j, tester si la matrice est un carré magique (⇒ la somme des entiers de chaque ligne, chaque colonne et des deux diagonales est identique). Exemple de carré magique de dimension 3

8 1 6

3 5 7

4 9 2

La classe java.lang.String

- Déclaration/création d'un objet de la classe String

  String ch ; ch4=null ; ⇒ déclaration sans initialisation ou avec null

  String ch1="Bonjour" ; ⇒ initialisation avec un littéral ⇒ un objet de la classe String est crée et sa référence est affectée à ch1

- Création d'un objet ⇒ utilisation d'un constructeur de la classe String avec l'opérateur new
  - La classe String dispose de différents constructeurs permettant de fournir la source des caractères qui vont former la valeur initiale de la chaîne (source : tableaux de caractères,….) ⇒ exemple

La classe java.lang.String

```
Char[] tabCar={'B','o','n','j','o','u','r'} ;
String ch3= new String(tabCar) ;
System.out.println(ch3) ; ⇒ Bonjour
```

- Autres constructeurs :

public String() ⇒ initialise une chaine vide ⇒ String p= new String() ;

public String(String original) ; ⇒ String p= new String("test") ; String k=new String(p) ;

public String(char[] value) ⇒ char[] tab={'a','b'} ; String p=new String(tab) ;

etc...

La classe java.lang.String

- Les chaînes de caractères de type String sont constantes/non modifiables (une fois crée la valeur d'une chaîne ne peut pas être modifiée)
- Exemple : String ch1= "abc" ;   ch1="test" ;

  ⇒ après la deuxième instruction, la chaîne "abc" existe toujours, mais on a perdu sa référence car ch1 porte maintenant une référence sur la chaîne "test" ⇒ on a donc ici deux chaînes qui occupent chacun un espace mémoire même si on a perdu la référence sur la chaîne "abc"

La classe java.lang.String

- Exemple :

```
String ch1="abc", ch2=new String("bb");
String ch3=ch1, ch4=ch2;        // ch3 et ch1 porte la même référence et pointe sur la même chaine ; pareille pour ch4 et ch2
System.out.println(ch1);          //  abc
System.out.println(ch2);          // bb
System.out.println(ch3);         // puisque pointe sur la même chaîne que ch1 ⇒ abc
System.out.println(ch4);         // puisque pointe sur la même chaîne que ch2  ⇒ bb

System.out.println("+++++Si on modifie les références de ch1 et ch2++++++++");

//maintenant ch1 et ch2 portent des références vers d'autres chaînes,  mais les chaînes sur lesquelles pointent ch3 et
//ch4  existent  toujours dans la mémoire. Donc il y  a création de nouveaux objets,  autant d'objets que de chaînes Pas
//d'écrasement du contenu des  chaines par les nouvelles valeurs

ch1="test"; ch2="tt";
System.out.println(ch1);        // test
System.out.println(ch2);        // tt
System.out.println(ch3);        // abc
System.out.println(ch4);        //bb
```

La classe java.lang.String

- Exemple :

```
 String ch= "abc" ;
String t =ch.toUpperCase(); // met les caractères en majuscules
System.out.println(ch);        // abc
System.out.println(t);         // ABC
```

⇒ la chaîne ch n'est pas modifiée, la méthode qui est appelée créée un autre objet avec les caractères majuscules et la référence de ce nouvel objet est affectée à t

La classe java.lang.String

- Quelques méthodes de la classe String

```
String p = "abcde" ;
String t =p.toUpperCase();   // met les caractères en majuscules=> ABCDE
String t =p.tolowerCase() ;   //retourne une chaine en minuscules
int k=p.length() ;                   // retourne la taille de la chaine ⇒ ici 5
boolean b=p.isEmpty() ;    // retourne true ssi p.length() ==0, sinon false
int c=p.compareTo("abcd") ; // compare deux chaines ⇒ 0 si égalité | < 0 | >0
int c1=p.compareToIgnoreCase(String str) ; //comparaison qui ignore la casse
```

etc..   voir documentation

La classe java.lang.String

- Les chaînes dont on a perdu les références occupent inutilement des espaces jusqu'à ⇒ garbage collector
  - Appel direct du garbage collector au lieu d'attendre les appels automatiques ⇒ java.lang.System.gc(); //public static void gc()
- S'il y a nécessité de modifier souvent des chaînes de caractères ⇒ utiliser d'autres types qui permettent de créer des chaînes modifiables
  - StringBuilder
  - StringBuffer
- Les deux sont semblables mais avec comme différences :
  - StringBuilder ⇒ pas adaptée à la programmation concurrente (not thread-safe)
  - StringBuffer ⇒ adaptée aussi à la progr. concurrente (thread-safe) ⇒ méthodes synchronisées

La classe java.lang.StringBuilder

- Exemple

```
StringBuilder sb = new StringBuilder("abc") ;
sb.append("de") ;
System.out.println(sb) ; // affiche   abcde
sb.insert(2, "le");
System.out.println(sb);  // affiche ablecde
```

La classe java.lang.StringBuffer

- Exemple

```
StringBuffer sb = new StringBuffer("abc") ;
sb.append("de") ;
System.out.println(sb) ; // affiche   abcde
sb.insert(2, "le");
System.out.println(sb);  // affiche ablecde
```