Introduction au langage JAVA

Tome 1 : cours et exercices

Université de Paris 13

Cours Préparé par A. OSMANI

Sommaire

Cours n° 1 : Introduction

Java est un langage orienté objet qui a la particularité d'être indépendant de la plate-forme, selon le principe proposé par Sun Microsystems : « write once, run everywhere »[1]. Pour cela, une machine virtuelle Java (JVM) a été définie. Elle assure le fonctionnement des programmes java. La seule condition pour qu'un programme java s'exécute sur une machine physique est que celle-ci dispose d'un système d'exploitation disposant de programmes capables de compiler le code source de façon identique (interpréteur java) et savent exécuter des programmes compilés sur des processeurs 32 bits. Cette condition est vérifiée, en particulier, par les plates-formes PC sous Windows, Sun avec le système Solaris et les Macintosh.

L'objectif de ce cours est d'apprendre la programmation-objet illustrée par le langage de JAVA. Ce cours va vous permettre de comprendre l'utilité et les caractéristiques de Java. Il détaillera le rôle de Java dans le développement d'applications pour le WEB et permettra de maîtriser l'environnement de programmation.

1.1 Historique

Les langages orientés objets (OO) sont apparus au milieu des années soixante-dix (Simula, le premier langage à implanter la notion de classe, date de 1967. Smaltalk implante la plupart des principes fondateurs de l'approche objet dès 1976). Au milieu des années quatre-vingt-dix, le nombre de langages OO a dépassé une cinquantaine.

En 1990, Sun Microsystems a lancé le projet Green, dont l'objectif est la mise au point d'une télécommande universelle baptisée Star7. Ce projet comprend notamment, un système d'exploitation simple capable de gérer les appareils électroménagers. Ayant constaté les difficultés du langage C++, James Gosling commença le développement d'un langage de programmation destiné aux appareils ménagers, afin de pouvoir les rendre interactifs et aussi de permettre une communication entre les machines. Ce langage, baptisé Oak, est un langage très inspiré du langage C++. Oak n'a pas su s'imposer. Sun a alors tenté de tirer profil de cette expérience en développant un langage destiné à l'Internet, il s'agit du langage Java.

La première version du langage Java est diffusée en mai 1995. En 1996, la version 1.02 est distribuée. Cette dernière est capable de prendre en compte les bases de données et les objets distribués. La version 1.1 (1997) introduit les composants Java Beans. La version finale de Java 1.2 - appelée également Java2- est apparue en 1998 ; cette version définit en particulier les Java Foundation classes dans laquelle on retrouve les bibliothèques pour le graphisme, le glisser-déplacer et les swing. La dernière édition de Java 2 est l'édition 1.4 Bêta 2 disponible sur le site web de Sun depuis le 27 août 2001.

[1] Ecrire une fois, utiliser partout.

1.2 Propriétés du langage JAVA

Java est un langage :

- Simple et orienté objet

  Surtout pour les habitués du langage C++. En effet, les concepteurs de Java ont retiré la gestion des pointeurs, la surcharge des opérateurs, l'héritage multiple et les mécanismes de libération de la mémoire.

- Distribué

  Il est assez facile à mettre en œuvre des architectures client-serveur java pour travailler sur des ordinateurs distants. Les fonctions d'accès réseaux et les protocoles de communication Internet font partie intégrante du langage Java. Java possède une importante bibliothèque de classes et méthodes permettant la gestion des protocoles TCP/IP, http(HyperText Transfert Protocol) , FTP, etc. Une application Java est capable d'accéder à des données distantes via des URL (Uniform Resource Locator) avec la même facilité qu'elle accède à des fichiers locaux.

- Robuste et hautement sécurisé

  Le typage des données est strict (Java est fortement typé et impose aux programmeurs une certaine rigueur de programmation, ce qui permet d'éviter bon nombre de bugs). Les pointeurs sont inaccessibles aux programmeurs. De plus, l'idée principale est qu'un programme comportant des erreurs ne doit pas pouvoir être compilé.

- Portable

  Les types et plus généralement les spécificités du langage sont indépendants de la plate-forme utilisée. L'environnement de programmation est lui aussi portable.

- Indépendant des architectures matérielles

  Le compilateur java produit un bytecode universel.

- Performant

  Java est un langage interprété, ce qui implique que les programmes sont plus lents que ceux qui écrits dans des langages compilés. Des améliorations sont apportées pour générer un byte code adapté au type de machines.

- Dynamique

  Les classes java –contrairement au C++- peuvent être modifiées sans demander l'adaptation des programmes qui les utilisent.

- Multithread

Il permet une exécution simultanée de plusieurs processus (thread). Il fournit des outils de synchronisation et de rendez-vous. Avec un même programme vous pouvez, par exemple, diffuser un son, scruter le clavier en même temps.

- Extensible à l'infini

  Java est écrit en Java. Toutes les catégories d'objets existant en java peuvent être définies par extension d'autres classes en partant de la classe de base la plus générale : la classe Object.

1.3 Quelques Applications

Java peut être utilisé pour diverses applications. Parmi ces applications, nous citons :

1. les environnements de développement : beaucoup d'environnements se programment de plus en plus en Java, comme JBuider et VisualAge d'IBM.
2. les applications bureautiques : l'exemple le plus diffusé est l'application StarOffice entièrement écrite en Java.
3. les applications client-serveur. Des serveurs complets sont écrits en Java. Exemple : Java Web Server de Sun
4. les servelets. L'accélération des taux de transmission et le besoin d'une meilleure présentation des pages web augmente la demande des pages HTML dynamiques. Les langages comme les CGI, ASP et PHP3 sont utilisés pour cette fin. A l'exception des CGI, beaucoup de langages ne s'exécutent que sur des serveurs particuliers. Sun a développé le concept de servelet Java pour résoudre ce problème de compatibilité.

2 Cours n° 2 : Installation et utilisation de Java

En mode cloud, java peut s'exécuter directement à partir de vos notebook (pour les étudiants de l'IUT), ou en utilisant un IDE à partir d'un navigateur comme replit.

Localement, vous devez installer java sur vos machines en suivant le guide d'installation manuelle de java disponible sur le site d'oracle.

Les conventions d'écriture de java sont strictes. Vous devez vous référer au site de référence d'Oracle pour connaitre les détails. Ce site donne la documentation de l'édition standard de java (d'autres éditions existent comme l'édition entreprise non abordée dans ce cours). Elle donne l'essentiel des recommandations à suivre.

Le lien suivant donne aussi l'accès direct à l'API et à toutes les classes des packages standards de java, y compris le package java.lang qui est utilisé par défaut sans qu'il ne soit utile de l'évoquer.

3 Cours n° 3 : Premier programme

3.1 Les outils nécessaires

Pour écrire votre premier programme vous avez besoin :

1. D'un éditeur de texte. Vous pouvez pour cela utiliser n'importe quel éditeur de texte en prenant le soin de sauvegarder le programme en mode **TEXTE**. Des éditeurs dédiés (permettant la compilation, l'exécution, etc.) existent. Certains sont gratuits et accessibles par Internet (exemple : Jcreator sur le site www.jcreator.com ou bien l'éditeur UltraEdit qui semble être performant. Celui-ci est développé par la société IDM. C'est un shareware disponible sur le site http://www.ultraedit.com/), d'autres sont disponibles dans le commerce (exemple Jbuilder). L'environnement Eclipse (http://www.eclipse.org) offre un éditeur de code de qualité et un ensemble de fonctionnalités appréciables.

2. Une fois que le programme est écrit, il faut qu'il soit exécuté par la machine. Pour cela, nous avons besoin de l'édition standard de la plate-forme Java2. Celle-ci est accessible sur le site http://java.sun.com/javase/6/ . Vous pouvez la télécharger puis l'installer sur votre ordinateur.

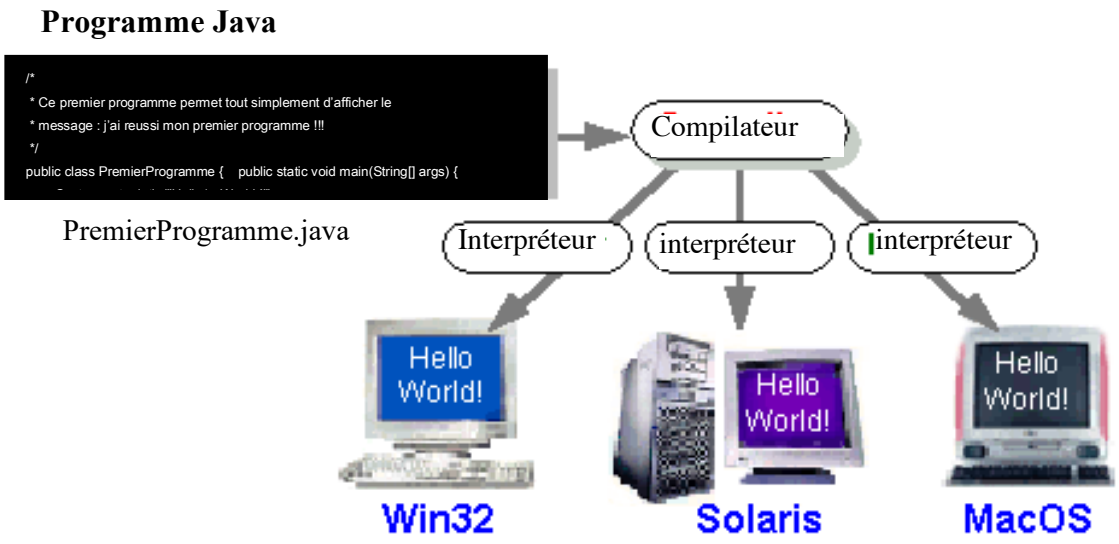
3.2 Le programme

N'essayez pas de comprendre les détails de ce petit programme. Tapez-le à l'aide de l'éditeur MicrosoftWord. Puis, sauvez-le en lui donnant impérativement le nom suivant « PremierProgramme.java ».

Vous pouvez toutefois retenir qu'un programme java comporte au moins une classe. S'il contient plusieurs classes, une et une seule doit être définie de type public (mot clé public). Le nom du fichier doit être composé du nom de la classe publique suivi de l'extension « .java ».

```
/**
 * Ce premier programme permet tout simplement d'afficher le
 * message : Hello World !
 */
public class PremierProgramme {
   public static void main(String[] args) {
      System.out.println("Hello \n World!");
   }
}
```

L'éditeur de texte vous permet de créer le code source de votre programme que tous les programmeurs java peuvent comprendre. Cependant, ce programme ne s'exécute pas tout seul. Le compilateur et l'interpréteur java permettent de transformer votre texte en un programme compréhensible et exécutable par la machine cible comme le montre la figure suivante :



La compilation d'un fichier source Java se fait en utilisant la commande javac suivi du nom de programme.

```
javac PremierProgramme.java
```

Une fois compilé, le programme java peut être exécuté en lançant l'interpréteur java suivi du nom de fichier sans l'extension.

```
java PremierProgramme
```

L'exécution de cette dernière commande permet d'afficher le message :

```
Hello
World!
```

3.3 Comment tout cela fonctionne ?

Les premières lignes du programme sont des lignes de commentaires. Elles commencent par /* et se terminent par */. Tous ce qui est entre ces deux couples de symboles est ignoré par java. Une autre façon d'ajouter une ligne de commentaires et de faire précéder cette ligne par //.

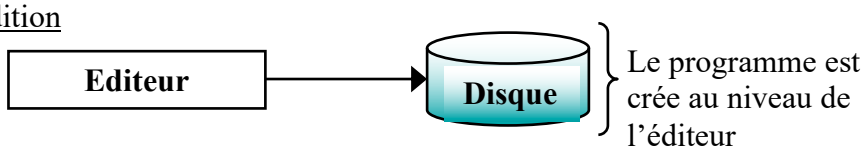
La ligne suivante commence par le mot clé public, qui signifie que ce qui va suivre est publics et accessible de partout par tout le monde. Celui-ci est suivi du mot clé class, il indique la déclaration d'une nouvelle classe dont le nom est donné juste après ce mot clé. Cette ligne se termine par une accolade, qui marque le début d'un bloc et dans ce cas le début de la définition de la classe. Un bloc se termine par une accolade fermante.

La ligne suivante définit la méthode principale main() suivi de l'appel à une méthode println qui se trouve dans la classe de l'objet out qui est contenu dans la classe System. Tout ceci paraît un peu compliqué nous y reviendrons en détail dans les prochains cours.

3.4 Analyse des étapes de traitement du programme

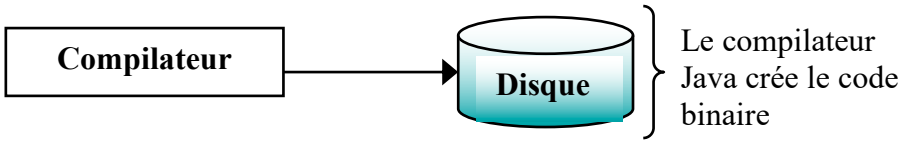
Les étapes par lesquelles passe un programme depuis l'édition, jusqu'à l'exécution peuvent être découpées en cinq, comme les montrent les figures qui suivent.

Etape 1 : édition



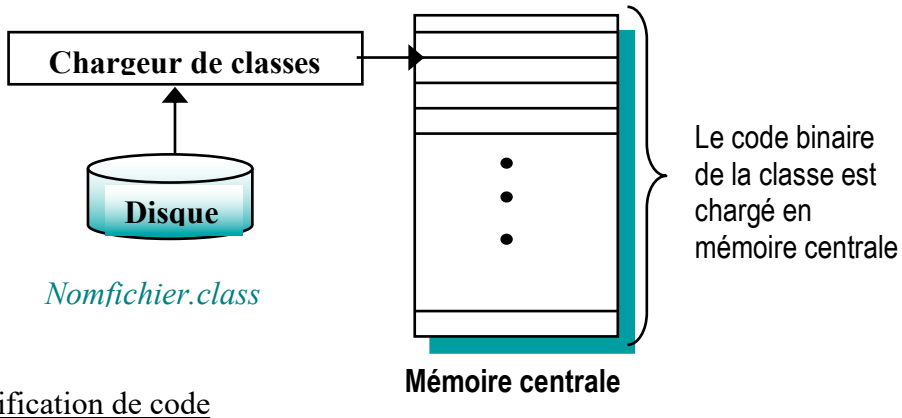
(Editeur DOS, Notepad) *Nomfichier.java*

Etape 2 : compilation

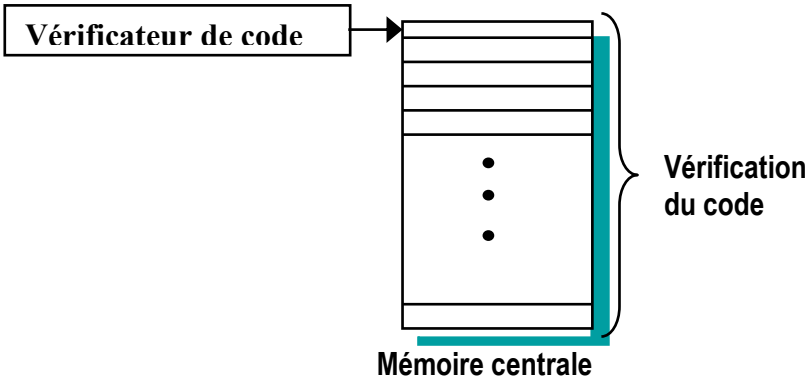


Javac *Nomfichier.java*

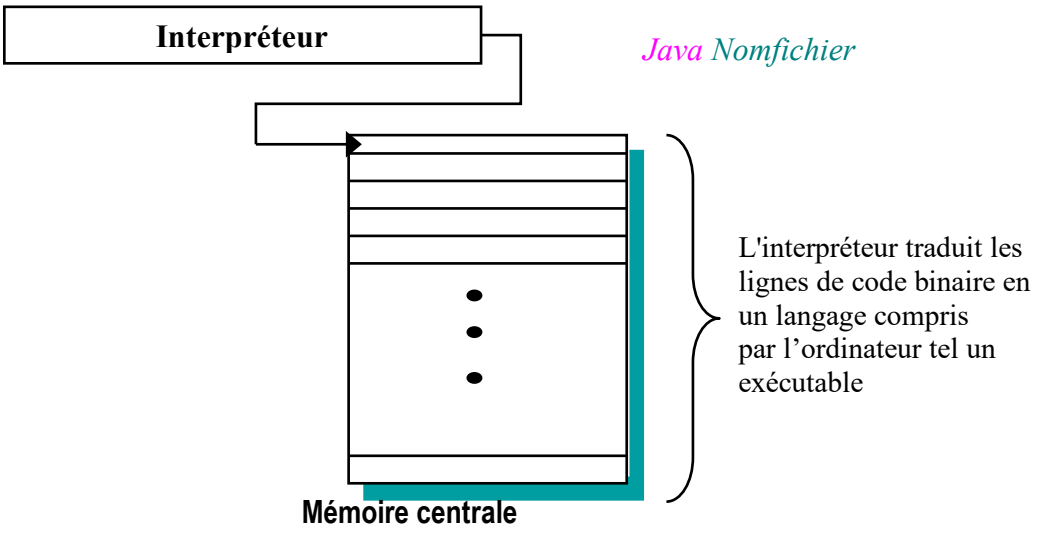
Etape 3 : Chargement



Etape 4 : vérification de code



Etape 5 : interpréteur de code (dans le cas d'une interprétation)



TP n° 1 : Environnement de développement

Exercice 1 (30 minutes)

Le site https://docs.oracle.com/javase/7/docs/api/ regroupe présente la documentation des classes de base du langage Java. Il est indispensable de passer le temps nécessaire pour découvrir l'organisation de la documentation, les packages de base (en particulier le package lang), les classes indispensables à connaitre (Object, Class et String, notamment).

Exercice 2 (1h minutes)

Rare sont encore des programmeurs qui utilisent des éditeurs (même spécialisés) pour développer des applications. Il est indispensable de connaître au moins une plateforme de développement pour bien comprendre la philosophie de développement dans un environnement assisté. Eclipse est une bonne plateforme pour tout développeur en java. Cet exercice permet de voir l'architecture d'Eclipse ainsi que quelques projets récents autour de cette plateforme.

Exercice 3 (30 minutes)

Programmer en Java nécessite une maîtrise minimale de la bibliothèque des classes.

- ✓ Navigation intuitive dans la bibliothèque des classes java.
- ✓ Visualisation de quelques programmes de démonstration
- ✓ Exécution du premier programme vu en cours.

4 Cours n° 4 : La classe, l'instance et les types de base

4.1 La programmation orientée objet

Le monde qui nous entoure est complexe. Pour domicilier cette complexité nous définissons, classifions, organisons et hiérarchisons les différents objets qui le composent. En précisant que chaque objet possède des caractéristiques et des propriétés qui décrivent sa composition et ses fonctionnalités.
Ainsi nous savons, par exemple, définir ce qu'une personne (son nom, son numéro de sécurité sociale, son emploi, etc.), sa classe sociale, son niveau dans une organisation, etc. Cette façon de faire permet de synthétiser nos connaissances à différents niveaux d'abstraction.

De cette constatation, on peut définir les objets comme des composantes (indépendantes) de l'environnement qui interagissent. Ceci permet, en cas de besoin, de négliger les détails de l'implémentation des objets et de ne se concentrer que sur ses propriétés et son interface avec l'extérieur.

Tous les objets de même nature peuvent être qualifiés par le même ensemble de caractéristiques et de propriétés. La définition des caractéristiques et des propriétés des objets de même nature s'appelle une classe. Un objet particulier d'une classe s'appelle une instance d'une classe.

Exemple :

Si on s'intéresse à l'objet toto (le chat de la grand-mère), on sait que c'est un chat qui partage beaucoup de propriétés avec d'autres chats. Et c'est aussi un animal qui a la particularité de vivre avec l'homme.
Si nous voulons considérer les chats, nous avons plusieurs possibilités :

1. Ou bien dire que c'est un animal particulier. Dans ce cas on prendrait la description de l'animal et lui ajoute ce qui caractérise un chat (héritage : spécialisation).
1. Ou bien c'est l'ensemble d'objets chat qui sont décrits par un ensemble de caractéristiques (définition d'une nouvelle classe).

1. On peut aussi dire que c'est une catégorie qui englobe les chats domestiques (héritage : généralisation)

Le langage Java est un langage tout objet ; il est exclusivement constitué de classes. Toutes les variables et toutes les fonctions doivent être définies dans des classes.

Java, le retour …

4.2 La classe et l'instance

Le JDK est composé uniquement de classes et d'interfaces. Cependant le nombre de classes est bien plus grand que le nombre d'interfaces.
Une classe peut être considérée comme une brique d'une application java. Elle contient des variables et des méthodes qui agissent sur les variables. Elle sert de schéma pour la création d'une instance.

Définition d'une classe
La forme la plus générale d'une classe est définie comme suit [2] :

```
[public|protected|private ][abstract|final] class NomDeClasse [extends ClasseDeBase]
[implements interface]
{  // déclaration des variables et définition des méthodes de la classe
}
```

public , protected et private indiquent les droits d'accès. abstract et final désignent le type de la classe. extends précise si une classe hérite d'une autre. implements précise si la classe implémente une ou plusieurs interfaces. La description de la classe est faite à l'intérieur des accolades.

- Les cours suivants vont définir plus précisément le sens et le rôle de chaque mot clé.

Exemple

```
/** commentaire destiné au générateur de documentation javacdoc
**   @author : A.Osmani
**    version : cours 4
*/
class Personne
{  String nom ;
   String prenom;
   int age;
   int renvoiAge ()
   {   return age;
   }
}
```

La classe Personne contient trois variables nom, prénom et age. Elle définit une méthode renvoiAge() qui renvoi l'age de la personne.

L'instance

[2] Les expressions mises entre deux crochets sont optionnelles. La barre verticale désigne une alternative.

Les instances de classes sont des objets utilisés lors de l'exécution. C'est un exemplaire de la classe lors de son exécution.

Définition d'une instance :

```
NomDeLaClasse   identificateurDeLInstance ;
IdentificateurDeLInstance = new NomDeLaClasse([paramètres])
```

Exemple :

```
Personne alain ;            // les deux lignes peuvent être remplacées par la ligne
alain = new Personne() ;  // suivante :  Personne alain = new Personne() ;
```

La déclaration de la variable alain ne crée pas une instance ou un objet Personne. Elle crée une variable qui référence un objet Personne. C'est uniquement l'utilisation du mot clé new qui permet de créer physiquement l'objet, en lui réservant de l'espace mémoire.

En Java, les instances d'une classe sont toujours créées dynamiquement (l'allocation de l'espace mémoire se fait en utilisant le mot clé new). Par contre la libération de l'espace occupé par une instance se fait automatiquement[3].

L'argument de new est le constructeur d'une classe.

Définition :

Le constructeur d'une classe est une méthode qui porte toujours le même nom que la classe. Il spécifie les paramètres nécessaires à la création d'une classe. Les classes définissent toujours au moins un constructeur.

Exemples :

```
class Personne
{   string nom ;
    string prenom;
    int age;

    // le constructeur sans paramètres
    Personne()
    { age = 0;
    }

     // le constructeur avec paramètres
    Personne(String leNom, String lePrenom, int lAge)
    {   nom = leNom;
        prenom = leprenom;
        age = lage;
     }
          ...
 }
```

[3] Java utilise la technique de ramasse-miettes (garbage collector) pour supprimer les objets qui ne sont plus utiles.

4.3 Accès aux variables et aux méthodes

Avant de vous montrer comment on accède aux variables et aux méthodes dans la classe ou à l'extérieur de la classe, je vais commencer par décrire leurs différents statuts.

Une variable peut être
- ✓ une variable de classe
- ✓ une variable d'instance
- ✓ une variable locale

Variable d'instances

Une variable d'instances ou d'objets est une variable qui n'est accessible qu'à partir d'un objet particulier.
Exemple : si on considère le nom d'une personne (voir classe Personne ci-dessus), on ne peut avoir accès à celui-ci que si et seulement si on dispose d'une instance donnée de la classe Personne.

En java, une variable d'instance est définie à l'intérieur d'une classe précédée d'un éventuel modificateur d'accès et de son type.
Exemple :

```
class Point
{   int abscisse ; // abscisse et ordonnée sont deux variables d'instances
    int ordonée ; // elles sont propres à chaque point
        …
}
```

Variable de classes

Une variable de classe est une variable partagée par toutes les instances d'une classe. On les appelle également variables statiques.
Syntaxe

```
static [modificateurDAccès <type> nomDelaVariable [définition ] ;
```

Exemple :

```
class Personne
{   int nom ;
    int prenom ;
    int age ;
    static int nombrePersonne = 0 ;
                …
}
```

Une variable statique est disponible en un seul et unique exemplaire pour toutes les instances. Cela signifie que la modification de sa valeur dans n'importe quelle instance impliquera sa modification pour toutes les autres instances. Une variable statique est initialisée au niveau de la classe.

Variable locales

Une variable locale est une variable déclarée à l'intérieur d'une méthode. Elle est créée et initialisée lors de l'appel de la méthode. Elle est détruite lorsque la méthode est terminée.
Exemple :

```
class Personne
{   …
   int indice ( int v)
   {  int voiciUneVariableLocale = 9 ;
      …
    }
          …
}
```

Une méthode peut être une méthode de classe ou une méthode d'instance. Elle est définie de la même façon qu'une variable de classe ou d'instance.

Accès

1. à l'intérieur d'une classe :
   Les variables et les méthodes de classes et d'instances sont accessibles directement par leur nom.
   Exemple :

```
class Personne
{  String nom ;
    String prenom;
    int age;
   Personne ( String a,String b, int c)
  {   initialisation (a, b, c);
  }
    void initialisation(String lenom,String lePrenom, int lAge)
    {   age = lAge;
        nom = leNom;
        prenom = lePrenom;
    }
                    ...
}
```

Un autre moyen est utilisé pour accéder aux méthodes et aux variables d'une classe. Il s'agit de les faire précéder du mot clé : this.

Le référence this :
this désigne l'objet qui est en train de s'exécuter.

Syntaxe : `this.<variable ou méthode>`

Généralement, il n'est pas utile d'utiliser explicitement this, car la référence à l'objet courant est implicite. Celui-ci devient utile dans certains cas. Comme, par exemple, lorsque des variables sont d'instances sont masquées.

Exemple :

```
class Personne
{ …
  Personne (String nom ,String pren)
  { this.nom = nom ;
    prenom = pren ;
  }
  Personne (Sting nom, String prenom, int a)
   {  this(nom ,prenom) ;
      age =a ;
   }
}
```

Remarque :

Les méthodes statiques d'une classe accèdent aux membres d'un objet de la même classe de la même façon que les autres classes.

Exemple :

```
class Personne
{ String nom;
   public static nombreDePersonne = 0;
   ...
   public static void main (String [] args)
   { ✗  nom = "alain"; // nom n'est pas accessible directement
                       // à partir d'une méthode statique
      Personne p =new Personne();
      p.nom = "alain" ; // ok voir accès à partir d'autres classes
            ...
   }
}
```

2. à partir d'autres classes :

Les autres classes accèdent aux membres statiques (variable ou méthode) d'une classe par l'intermédiaire d'une référence. Cette référence peut être ou bien une référence à une classe :

Syntaxe :

```
<IdentifiantDeLaClasse>.leMembre
```

ou bien une référence à un objet :

```
<IdentifiantDUnObjetquelconque delaClasse>.leMembre
```

Les autres classes accèdent aux membres d'instances d'une classe par l'intermédiaire d'une référence à une instance exclusivement.

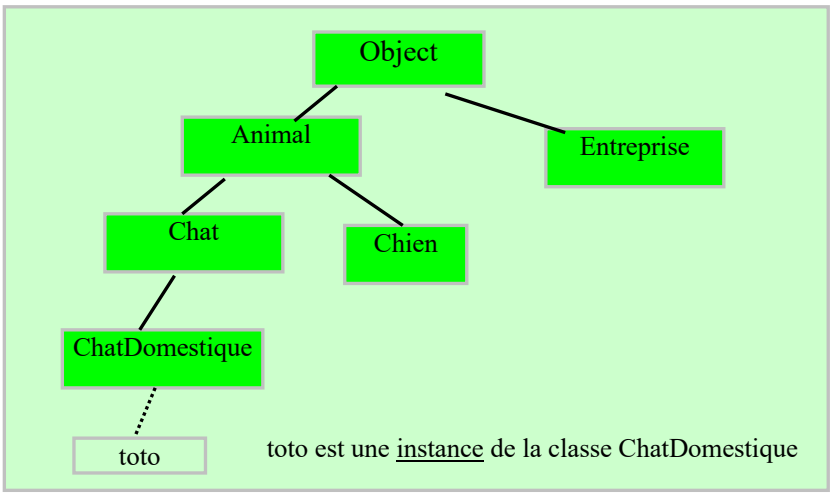
Exemple :

On crée un programme qui va contenir la classe Personne et la classe Etudiant.

```
class Etudiant
{  Sting  prenom ;
   String  nomEtudiant ;
    …
    void ajouterUnMembre (Personne p)
    {  nomEtudiant = p.nom;
       prenom = p.prenom ;
       Personne.nombreDePersonne++ ; // ou bien p.nombreDePersonne++
```

4.4 La classe superclasse

Les classes en Java sont définies sous forme d'une hiérarchie de classes. La classe la plus générale se trouve à la racine. Il s'agit de la classe Object, et est appelée classe super cosmique ou bine superclasse car c'est elle la classe la plus générale de tout ce qui peut être défini en Java. Si nous revenons à notre exemple de classe chat domestique, la représentation hiérarchique des classes est la suivante :



- ✓ La syntaxe utilisée est similaire à celle utilisée par le langage C
- ✓ Les instructions se terminent par un ;
- ✓ Les caractères sont codés dans le code UNICODE[4].

4.5 Les types de données

4.5.1 Les types de base (types primitifs)

Java possède quelques types de données primitifs qui ne sont définis par des classes. Ces types sont gérés par copie (comme en C): une affectation ou un passage de paramètre de type primitif consiste à copier sa valeur dans la variable cible. Les types primitifs ont des tailles uniques quelle que soit la plateforme.
La liste exhaustive de ces type est la suivante :

[4] L'UNICODE est un code à 2 octets qui permet de coder tous les caractères de toutes les langues. UNICODE est compatible avec le code ASCII. Si vous voulez écrire un caractère non disponible sur le clavier, il suffit d'écrire son code en hexadécimal comme suit \uxxxx.

1. Les entiers signés : byte (8bits de –128 à +127), short (16 bits de –32768 à +32767), int (32 bits de –2147483648 à 2147483647) et long (64 bits). Ils sont tous initialisés à 0 par défaut.
2. Les flottants : float (32 bits de ±1.4E-45 à 3.40282347^{E}38 avec 6 à 7 décimales significatives) et double (64 bits avec 15 décimales significatives) également initialisés à 0. Le type double est donc deux fois plus précis que le float. Les opérations de virgule flottante sous java sont standardisées afin de respecter la spécification de la norme internationale IEEE 754, ce qui signifie que les résultats des calculs flottants seront généralement[5] identiques sur toutes les plates-formes Java.
3. Les caractères : char (2 octets) initialisés à \u0000. Format de caractère Unicode.

```
char c = 'e' ;
char c2 = '\u1230' ;
char c3 = '\n' ;
```

4. Les booléens : boolean (1 bit : true ou false) initialisé à false.
   Attention : le type booléen est incompatible avec le type entier.
5. void (0 bits).

Déclaration et initialisation

Les types de base, comme les autres types en Java, sont déclarés dans les classes ou dans les méthodes.

```
int nombre1 ;
double d ;
boolean trouvé ;
```

Ils peuvent être initialisés directement lors de la déclaration :

```
int nombre1 = 25 ;
double d = 5.3;
boolean trouvé = false ;
```

Les entiers peuvent être déclarés (commence par le chiffre 0) sous la forme décimale (commence par l'un des chiffres 1 jusqu'à 9), octale ou bien hexadécimale (commence par le chiffre 0 suivi de la lettre x : 0x).

```
int nombre1 = 17 ;
int nombre2 = 021 ; //  nombre2 = 17 en décimal
int nombre3 = 0x11 ; // nombre3 = 17 en décimal
int nombre4 = 0xFABC ; // A=10, … F= 15 en décimal
```

Les entiers longs peuvent être suffixés par la lettre L ou l :

```
Long l1 = 200, l2 = 200L ;
```

Une conversion de types est implicite dans le cas ou le type de la variable cible est plus important que le type de la variable source. Cependant, l'affectation inverse doit faire apparaître de façon explicite la conversion de type.

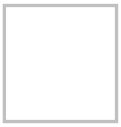
```
int a =25 ;
long b = a ;
✘ int c = b ; // Erreur à la compilation
int c = (int) b ; // conversion explicite est nécessaire
```

[5] Les dernières … supporte.

Les nombres flottants peuvent être exprimés avec des notations décimales ou scientifiques.

```
double d1= 200.0 ;
double d2 =2..0+2 ;
float f1 =200.0f ; // f ou F sont utilisés indifféremment.
float f2 = 2,0+2F;
```

Les types int, long, float, double, boolean et char peuvent être manipulés à notre guise. Cependant, ils ne possèdent aucune méthode qui leur est propre. Vu l'importance de ces types, les classes Integer, Long, Float, Double et Boolean, dites enveloppeurs, sont définies dans le package java.lang. Elles permettent de regrouper les fonctionnalités de chacun des types de bases et d'étendre les possibilités de leurs utilisations.



Lorsque vous passez des arguments à des méthodes, les paramètres sont passés toujours par valeur. Dans les cas des types de base, sa valeur est copiée dans une autre variable. Ce qui implique que la modification de la variable passée en paramètres n'affecte pas la variable initiale. Dans le cas des objets c'est la référence à l'objet qui est copiée et non pas l'objet. Il existe une seule instance de l'objet et deux références. Toute modification faite par une référence affecte automatiquement l'autre.

Le type null

null est une constante. On l'utilise, par exemple, lorsqu'il faut créer une instance d'un objet que ceci n'est pas utile à un moment donné.
null est une constante globalement utilisée comme valeur intermédiaire ou de substitution pour contourner des messages d'erreurs.

TD-TP n° 2: Ecriture des premières classes

Exercice 1 :

Ecrire le programme qui permet d'afficher les trois chaînes de caractères données en paramètre du programme.

Exercice 2 :

- Proposer une classe et définir ses données et ses méthodes.
- Définir deux ou trois constructeurs. Le premier ne doit pas avoir de paramètres.
- Déclarer puis initialiser quelques méthodes de votre classe dans la méthode main().
- Illustrer graphiquement la représentation des objets définis précédemment en mémoire centrale.

Exercice 3 :

Soit une classe Point ayant deux données privées : l'abscisse et l'ordonnée (entiers)

- Ecrire un constructeur d'un point dont les coordonnées sont i et j.
- Ecrire une méthode qui initialise un point en demandant à l'utilisateur les caractéristiques du point.
- Ecrire une méthode qui affiche le point.

Exercice 4 :

La classe Animal possède une variable booléenne : vivant et un entier : âge.

1. Définir la classe Animal
2. Le constructeur par défaut initialise l'âge à 0 et vivant à True. Il affiche à l'écran qu'il vient de créer un animal et indique son âge. Ecrire ce constructeur.
3. Ecrire le constructeur qui accepte comme paramètres l'âge de l'animal lors de la création et qui se comporte comme le constructeur par défaut.
4. L'animal peut vieillir, crier et mourir. Ecrire les corps de ces méthodes.
5. La méthode vieillir consiste à incrémenter l'âge de l'animal de 1 à chacun de ses anniversaires (appel de la méthode). Surcharger cette méthode pour pouvoir incrémenter l'âge de l'animal avec un entier positif quelconque.
6. Ecrire la méthode qui permet d'afficher les données de la classe Animal.
7. Créer une classe Jungle dans laquelle il faut définir :

- La méthode main()
- Quelques animaux
- Quelques opérations sur les animaux.

Que faut-il modifier au niveau de la classe Animal pour connaître le nombre d'animaux définit dans la méthode main() de la classe Jungle.

Exercice 5 :

La classe Complexe permet de définir des nombres complexes composés de deux variables entières : partieRéelle et partieImaginaire.

Définir cette classe ainsi que les méthodes permettant de réaliser les opérations arithmétiques entre deux nombres complexes avec affichage des résultats après chaque opération.

Exercice 6 :

Ajouter à chaque ligne du programme suivant la signification de l'instruction quelle représente. Certaines sont erronées. Trouvez-les et expliquez pourquoi.

```
class Personne
{           int age ;
            static int nombre ;
            Personne()
      { int val ;
        age =0 ;
       }
       static int renvoiNombre ()
       {  age++ ;
          return ombre ;
       }
        void modifierAge( int age)
        {  int b = age ;
           this. age =b ;
         }
         public static void main (String [] a)
         { Personne p;
           p.age = 0 ;
           P = new Personne() ;
           p.val = 0;
          Personne.age = 14 ;
           Nombre ++ ;
           Age ++ ;
        }
}
```

5 Cours n° 5 : Les opérateurs et les instructions de base

5.1 Les opérateurs

Un opérateur agit sur un ou plusieurs arguments pour produire une nouvelle valeur. Les arguments se présentent sous une forme différente de celle d'un appel de méthode standard, mais le résultat est le même.

On retrouve presque tous les opérateurs qu'en langage C. On retrouve cinq types généraux d'opérateurs et un opérateur ternaire (?) :

- ✓ opérateurs arithmétiques,
- ✓ opérateurs logiques,
- ✓ opérateurs de comparaison,
- ✓ opérateurs d'affectation,
- ✓ opérateurs au niveau bits.

Chaque catégorie peut être subdivisée en unaire et binaire. Les opérateurs unaires utilisent un seul opérande. Les opérateurs binaires utilisent deux opérandes.

Presque tous les opérateurs travaillent uniquement avec les types primitifs. Les exceptions sont « = », « == » et « **!**= », qui fonctionnent avec tous les objets. De plus, la classe **String** admet les opérateurs « + » et « += ».

5.2 Opérateurs arithmétiques

L'addition (+), la soustraction, la multiplication (*), la division (/), et l'affectation (=) fonctionnent de la même manière dans tous les langages de programmation.

Liste des opérateurs arithmétiques :

L'opérateur ++ explique le nom du langage C++ (un pas de plus au delà de c). un des créateurs de Java, Bill Joy, disait java est le C++--.

Opérateur	Rôle	associativité[6]	Définition	priorité

++/ --	Incrémentation/ décrémentation automatique	droit	Opérateurs unaires. Ils modifient la valeur de leur opérande en ajoutant ou en soustrayant 1 à leur valeur. Le résultat dépend de la position de l'opérateur.
			Exemples :
			int a =10 ; // a = 10
			int b = ++a ; // a=a+1 puis b = a donc a=b=11
			int c = b++ ; // c=b puis b =b+1 donc c =11 et b= 12
			c =++a +-b; // identique à c =(++a)+(-b)
			c =a+++-b; // identique à c =a+++(-b)
				1

+/-	Plus/moins unaires	gauche	Le compilateur les reconnaît par le contexte de l'expression Exemples : int a=12, b=-a ; b= a * -b; // instruction équivalente à b= a * (-b);	2
*	Multiplication	gauche		4
/	Division	gauche		4
%	Modulo	gauche	renvoie le reste de la division du premier opérande par le second	4
+/-	+/- Addition/ soustraction	gauche		5

[6] L'associativité indique l'ordre des opérateurs de même priorités, utilisés dans une instruction unique.

Java utilise également une notation abrégée pour effectuer en un seul temps une opération et une affectation. Ceci est compatible avec tous les opérateurs du langage (lorsque cela a un sens), on le note au moyen d'un opérateur suivi d'un signe égal. Par exemple, pour ajouter 4 à la variable **x** et affecter le résultat à **x**, on écrit : **x += 4**.

5.3 Opérateurs logiques

Les opérateurs logiques ou booléens permettent au programmeur de regrouper des expressions booléennes pour déterminer certaines conditions. Ces opérateurs exécutent les opérations booléennes standard (AND, OR, NOT et XOR).

Le tableau suivant résume ces opérateurs :

Opérateur	Rôle	associativité	Définition	priorité
!	Complément logique unaire(NOT)	droite	Exemple : boolean a = true ; if (! a) printf(“ ce message ne sera jamais affiché”); else printf("a est vraie") ;	2
&	Evaluation AND	gauche	Les opérateurs d’évaluation évaluent toujours les deux opérandes. Exemple : boolean a = (5 > 3)& (6>9) ;	9
^	XOR	gauche	a = c ^ d ; // a = true si c et d ont des valeurs de vérité différentes.	10
|	Evaluation OR	gauche		11
&&	Court-circuit AND	gauche	Les opérateurs court-circuit évaluent toujours le premier opérande. Si cette évaluation suffit, le reste de l’expression n’est plus évalué. Exemple : boolean a = (x<y) || (y<z) ;	12
||	Court-circuit OR	gauche		13

5.4 Opérateurs de comparaison ou relationnels

A la différence des opérateurs logiques, les opérateurs de comparaison n’évaluent qu’une seule expression.
Le tableau suivant résume des opérateurs de comparaison :

Opérateur	Rôle	associativité	priorité
<	inférieur	gauche	7
>	supérieur	gauche	7
<=	Inférieur ou égal à	gauche	7
>=	Supérieur ou égal à	gauche	7
= =	Egal à	gauche	8
!=	Différent de	gauche	8

Remarque :

L'opérateur d'égalité peut être utilisé pour comparer des objets de même type. Le résultat de la comparaison donne la valeur de vérité true si les deux variables référencent le même objet.

```
Etudiant e1 =new Etudiant();
Etudiant e2 =new Etudiant();
boolean  a1=m1 ==m2; //a1 va recevoir la valeur false
m1 =m2;
a1 =m1 ==m2;  // a1 va recevoir la valeur true
```

5.5 Opérateurs d'affectation

Le tableau suivant donne la liste des opérateurs d'affectation autorisés par le langage Java:

Opérateur	Rôle	associativité	Définition	priorité
=	Affectation	droite		15
+= - =	Ajout ou soustraction et affectation	Droite	Exemple : int a= 12 ; a+=3; // a=a+3 =15	15
*= /=	Multiplication ou Division et affectation	droite		15
&= |= ^=	AND ou OR ou XOR avec affectation	droite		15

5.6 Opérateurs au niveau bits

Il y a deux types d'opérateurs au niveau bits :

- Opérateurs de décalage. Ils permettent de décaler les chiffres binaires d'un nombre entier vers la droite ou la gauche.
- Opérateurs booléens.

Opérateur	Rôle	associativité	Définition	priorité
~	Complément au niveau bits	droite	short i =13;//i a la valeur 0000000000001101	2
<<	Décalage gauche signé	Gauche	Short i = 13 ; i =i <<2;//i a la valeur 0000000000110100	6
>>	Décalage droit signé	Gauche		6
>>>	Décalage droit par ajout de zéros	Gauche		6
&	AND au niveau bits	Gauche		9
|	OR au niveau bits	Gauche		10
^	XOR au niveau bits	Gauche		11
<<=	Décalage gauche avec affectation	Gauche		15
>>=	Décalage droit avec affectation	Gauche	short i = 52 ; i >>=3; //i a la valeur 0000000000000110	15
>>>=	Décalage droit par ajout de zéros avec affectation	Gauche		15

Remarque L'opération de décalage fonctionne différemment dans Java et dans C/C++, principalement en ce qui concerne les nombres entiers *signé*s. Dans un nombre entier signé, le

bit le plus à gauche précise le signe du nombre entier (il a la valeur 1 pour les nombres négatifs). Dans Java, les nombres entiers sont toujours signés, alors que dans C/C++, ils sont signés par défaut. Dans la plupart des implémentations de C/C++, une opération de décalage des bits ne conserve pas le signe du nombre entier (puisque le bit du signe serait décalé). Cependant, dans Java, les opérateurs de décalage *conservent* le bit du signe (sauf si vous utilisez l'opérateur >>>pour effectuer un décalage non signé). Cela signifie que le bit du signe est dupliqué, puis décalé (un décalage à droite de 10010011 de 1 bit donne 11001001).

5.7 Le seul opérateur ternaire : l'opérateur ? :

Le langage Java a hérité de l'opérateur ternaire ? disponible en C.

```
<expressionBooléenne>?<expression1>:<expression2>;
```

<expressionBooléenne> est évaluée en premier. Si le résultat est true alors <expression1> est évaluée sinon <expression2> sera évaluée.

Exemples

```
int a =21, b =12 ;
int c = (a<b) ?a:b; // c=12
```

Il ne faut pas confondre cet opérateur avec une instruction conditionnelle. Ce n'est pas une instruction, comme l'illustre l'erreur dans le code suivant :

```
(x >y)?max =x:max =y;  //ne peut pas être utilisé comme une instruction
```

Instanceof

Permet de vérifier si une référence à un objet donné est ou non une référence à une instance d'une classe.

Exemple :

```
Personne p = new Personne () ;
Voiture v;
if (v instanceof  Personne) v =p ;
```

5.8 Les instructions de base

5.8.1 Les instructions if et switch

```
If (condition)
{ … // peut ne pas être suivie de else
} else
{  …// ou bien suivie d'autant de else que nécessaire
} else
{ …
}
```

Si une requête contient beaucoup de valeurs entières (sauf long) ou bien de type caractère, il est plus agréable d'utiliser l'a requête switch :

```
switch ( variable)
{ case valeur1 :
```

```
        …
        break ;
    case valeur2 :
        …
       break ;
     …
    default :
        …
}
```

Exemple :

```
Int choix ;
...
Switch (choix)
{       case 1 : System.out.println("1 est sélectionné");
                 break ;
         case 2 : System.out.println("2 est sélectionné");
                 break ;
        case 3 : System.out.println("3 est sélectionné");
                 break ;
        default : System.out.println("ni 1, ni 2, ni 3 n'ont été choisis");
}
```

Expression conditionnelle

```
<condition> ? <valeur if> : <valeur sinon> ;
```

Exemple : str1= (x<=4) "valeur inférieure ou égal à 4" : "valeur supérieure à 4" ;

5.8.2 Les boucles

Les instructions d'itérations permettent de répéter une suite d'instructions jusqu'à ce que des conditions soient vérifiées.

```
while (condition)
    {  // effectue d'abord le contrôle. Tant que condition=true ce bloc d'instructions est
    exécuté
    } ;
do
    { // le contrôle de la condition est fait à la fin de l'exécution de la séquence d'instructions
    } while (condition) ;

for ( valeurInitiale ; conditionDExécution ;compteur)

    {

    }
```

5.8.3 Break, continue et return

Les instructions *break*, *continue* et *return* sont des instructions de saut.

L'instruction break permet de sortir d'une boucle.

break permet de quitter une boucle. De plus, après un contrôle, la branche switch se termine toujours par une instruction break.

NB : Break est rarement utilisé en dehors d'une requête switch.

Il est aussi possible d'utiliser des instructions break nommées (comme le goto utilisée dans d'autres langages)

```
MarqueDUnePosition :
…
break MarqueDUnePosition ;
```

Exemple :

```
class Animal
{ int age ;
  boolean vivant ;
  …
  void augmenteAge ( )
  {  Position1 : int a =Console.readInt("donner un nombre à ajouter à l'age ") ;
     if (a<0 ) break Position1 ;
  }
```

continue interrompt le passage de la boucle en cours, mais ne quitte pas la boucle. L'instruction continue termine un passage de requête, mais pas la requête elle-même.

Exemple : le programme suivant calcule la somme des ages des personnes (classe Personne) à l'exception de l'age des personnes qui ont un age supérieur à 60 ans.

```
…
public int sommeAge (Personnes [] p)
{          int resultat =0 ;
           for (int i = 0 ; i<=p.length, i++)
           {   if (p[i].age > 60) continue;
               resultat +=p[i].age;
           }
}
...
```

return est une valeur de retour. Elle permet à une méthode de renvoyer une valeur à la méthode appelante. Si une méthode ne retourne aucune valeur, la déclaration de la méthode doit être précédée du mot clé void.

Syntaxe : return (valeur) ; // ou bien return valeur ;

5.8.4 Les commentaires

Les commentaires qui portent sur une ou deux lignes peuvent être rédigés en les faisant précéder chaque ligne d'un double slache.
Les commentaires qui portent sur plusieurs lignes sont écrit à l'intérieur des symboles /* et */.

Java définit un type de commentaires utile pour la génération de code. Dans ce cas les commentaires sont compris entre les symbole /** et */. Ils sont utilisés par le générateur de documentation javadoc pour créer des pages HTML expliquant le code. Certaines informations peuvent être indiquées par des balises. C'est le cas du

- ✓ Nom de l'auteur @author nom
- ✓ La version du logiciel @version NumeroDeVersion
- ✓ Indiquée qu'une classe est remplacée par une autre : deprecated texte
- ✓ Décrire les valeurs retournées par une méthode : @return
- ✓ …

5.9 Structure de blocs

Un bloc est un ensemble d'instructions délimité par une paire d'accolades. Les blocs peuvent être imbriqués.
Il n'est pas possible de déclarer deux variables homonymes dans deux blocs imbriqués.

```
{   int x ;
    {       int x ; // impossible de redéfinir la même variable dan un bloc imbriqué
    }
}
```

5.10 Définition de constantes

En java le mot clé **final** sert à désigner une constante.

```
final float PI = 3.14 ;
```

Par convention, les noms des constantes sont tout le temps en majuscule.
Une constante de classe, se définit en la faisant précéder du met clé static :

```
static final PI=3.14
```

5.11 Conversions

5.11.1 Casting

Le casting consiste à convertir un type de donné en un autre à l'aide d'une syntaxe. On distingue deux type de casting
Casting dans le cas des types de base :
Exemple

```
float a = 12.23 ;
float b= -14.2 ;
int ia = (int)a ; // a= 12 .le nombre positif est toujours arrondi vers le bas
ia =(int)b ; // a= -14. le nombre négatif est toujours arrondi vers le haut
ia = (int) "bonjour" ; // erreur de compilation. Tous les castings ne sont pas autorisés.
```

Casting dans les cas des objets

Le casting peut être fait sur des objets appartenant à la même lignée. Ainsi, si la classe Chat dérive de la classe Animal, il sera possible d'avoir :

```
Animal x ;
…
Chat a = (Chat) x ;
```

NB : pour éviter les situations ou lace x à un chien ne soit affectée à une référence à un chat, prendre l'habitude de toujours tester le type du parent avant de l'affecter à un enfant. Utiliser pour cela l'instruction instanceof. (la notion d'héritage sera détaillée plus tard).

5.11.2 Conversion ad hoc

Il s'agit de la conversion automatique lors d'une affectation de valeur.
Exemple :

```
int a =12 ; float b = a ;
String c = "mon age est :"+a ;
```

5.11.3 Conversion à l'aide de méthodes

Cette conversion est libre. Il convient au programmeur de définir les méthodes qui lui permettent la conversion de n'importe quel type vers n'importe quel autre type.
Exemple :

```
String a ;
int val =140 ;
a=Integer.toString(a);
```

TD-TP n° 3 : Ecriture des premières classes

Exercice 1 :

La classe Personne définit les caractéristiques d'une personne. Elle contient en particulier trois attributs : taille, poids et âge.
La fonction catégorie() renvoie la catégorie de la personne en fonction de ces trois paramètres. Une personne est dite de catégorie :

- ✓ a, si l'âge de la personne est inférieur à 10 et sa taille supérieur à 1m50 ou bien son poids dépasse la moitié de sa taille ;
- ✓ b, si son âge est supérieur à 20 et son poids supérieur à 100 ou sa taille est inférieur à 1m50 et son poids supérieur à 80.

Exercice 2 :

Faire les résultats des opérations suivantes et donner les résultats.

- ✓ 4 & 7(4) 4|7(7) 4^7(3)
- ✓ 100<<2(400) -10<<2(-40) -10>>2(25)
- ✓ 100>>>2 (25) -10>>>2 (1073741821)
- ✓ a=5; a<<=3; b=9; b^=8; c=-20; c>>>3;

Rappel : les entiers négatifs sont calculés en inversant les bits et en ajoutant 1

Exercice 3 :

Ecrire une fonction prenant pour arguments deux **String**, utiliser les comparaisons booléennes pour comparer les deux chaînes et imprimer le résultat. En plus de == et !=, tester aussi equals( ). Dans la méthode main( ), appeler la fonction avec différents objets de type **String.**

Exercice 4 :

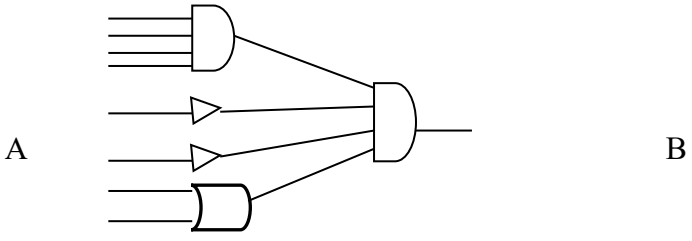
Ecrire un programme utilisant deux boucles **for** imbriquées ainsi que l'opérateur modulo (**%**) pour détecter et imprimer les nombres premiers (les nombres entiers qui ne sont divisibles que par eux-mêmes et l'unité).

Exercice 5 :

Ecrire un programme qui génère aléatoirement 25 valeurs entières. Pour chaque valeur, utiliser une instruction if-then-else pour la classer (plus grande, plus petite, ou égale) par rapport à une deuxième valeur générée aléatoirement.

Exercice 6 :

Ecrire un programme qui réalise le circuit suivant : A et B sont de type byte.



Exercice 7 :

Etudier les classes : Byte, Integer, Float, Double.

6 Cours n° 6 : Héritage

6.1 Introduction

La réutilisation est un des grands intérêts des langages objets. Supposons que nous désirons créer une classe *Etudiant* en sachant que nous disposons de la classe *Personne.* Une façon triviale de faire est de créer une classe Etudiant complètement indépendante de la classe Personne et qui contiendra les attributs de la classe *Personne* avec les méthodes associées. Cette façon de faire est à l'opposé de l'esprit de la programmation objet et du génie logiciel. Dans la mesure ou nous disposons déjà de la classe *Personne*. Les langages objets offrent un moyen simple de réutiliser le code de la classe *Personne* dans la classe *Etudiant,* il s'agit de l'héritage.

Syntaxe

```
class NomClasseFille extends NomClasseParent
{ …
}
```

Exemple

```
class Etudiant extends Personne
{  int numeroEtudiant ;
   …
}
```

NB : la classe Personne est dite superclasse, classe parent de la classe Etudiant ou encore classe de base. La classe Etudiant est dite classe dérivée, sous-classe ou classe fille de la classe Personne.

L'héritage permet de dériver de nouvelles classes à partir de classes existantes. Ce mécanisme permet la réutilisation et la spécialisation des méthodes associées aux classes existantes. Une classe obtenue par héritage possède la totalité des attributs et méthodes de la classe dérivée et ajoute des attributs et des méthodes qui lui sont spécifiques. Une classe dérivée peut également redéfinir le comportement des méthodes définies dans la classe parent.

```
class Personne
{ String nom, prenom ;
   int age ;
   …
   void afficheAge()
   { System.out.print("age de la
         personne :" + nom + " est :" + age) ;
   }
   …
}
```

```
class  Etudiant extends Personne
{  int numeroEtudiant ;
   void affiche()
   { System.out.print("l'age de
         l'étudiant"+nom+
          " est : "+age) ;
   }
   …
}
```

L'héritage est omniprésent en java. L'ensemble des classes du langage est organisé en hiérarchie permettant de structurer l'ensemble des classes du langage. La classe qui se trouve à la racine de la hiérarchie -à partir de laquelle dérivent toutes les classes en java – est la classe *Object*.

6.2 Les constructeurs de la sous-classe

6.2.1 Le constructeur par défaut

Si le constructeur sans paramètres n'est pas défini explicitement par le programmeur alors Java le crée automatiquement. Ce constructeur est dit constructeur par défaut.

Le constructeur par défaut d'une sous-classe n'invoque pas explicitement le constructeur par défaut de la classe parent ; cet appel est fait automatiquement par Java.

```
class  Etudiant extends Personne
{  …
   Etudiant()
   { numeroEtudiant = -1 ;
   }
  …
 }
```

```
class Personne
{ …
   Personne(String n, String p)
   {   nom= n;
       prenom = p ;
      age = -1 ;
   }
   …
}
```

La création d'un objet de type Etudiant en utilisant le constructeur par défaut va fait appel d'abord au constructeur par défaut de la classe parent (Personne), puis au constructeur par défaut de la classe Etudiant.

6.2.2 Invocation du constructeur de la classe parent

Le constructeur de la classe parent peut être invoqué par la sous-classe en utilisant le mot clé super (…).

```
Etudiant()
{ super() ;
  numeroEtudiant = -1 ;
}
```

```
class  Etudiant extends Personne
{  …
   Etudiant(String nom, String p)
   {   super (nom,p) ;
       numeroEtudiant = -1 ;
   }
  …
}
```

Le constructeur par défaut de la classe Etudiant, présenté dans la section précédente, sera transformé automatiquement par Java comme suit :

```
class Personne
{ …
   Personne()
   {   nom ="";
       prenom = "";
```

Il est facile de constater que dans le cas ou la classe parent dérive d'une autre classe, son propre constructeur par défaut commence par l'appel du constructeur par défaut de sa superclasse. En java, la racine de la hiérarchie d'héritage étant la classe Object, l'enchaînement des appels de constructeurs se poursuit jusqu'au constructeur de cette classe. Le premier constructeur qui sera exécuté sera donc le constructeur de la classe Object suivi de tous les autres constructeurs. Le dernier qui sera exécuté est le constructeur de la classe appelante.

NB : l'invocation de super(..) doit être la première instruction du constructeur de la classe dérivée.

6.3 Redéfinition des membres (des attributs)

En général, les attributs définis dans les classes filles sont des attributs complémentaires des classes parents ayant des noms différents de ceux apparaissant dans les superclasses.
Si on définit dans une sous-classe un attribut ayant le même nom que celui de la superclasse alors l'invocation du nom de l'attribut dans la sous-classe désignera toujours le membre de la sous-classe.

Pour avoir accès à l'attribut de la superclasse, il faut utiliser le mot clé super.

```
class Personne
{ int num ;
   void initNenum(int b)
   {    num = b;
    }
   …
 }
```

```
class  Etudiant extends Personne
{  int num ;
   void initNumero(int a)
   {  num = a; // initialise num de Etudiant
      this.num = a ; //  idem
      super.num= a;// initialise num de Personne
   }
  …
 }
```

6.4 Redéfinition des méthodes

La redéfinition des méthodes consiste à définir une implémentation différente des méthodes de la classe de base ayant les mêmes signatures dans les classes filles.
Dans l'exemple suivant la méthode affiche() est redéfinie dans la classe Etudiant pour mieux préciser le type de l'objet. Si la fonction affiche() n'est pas redéfinie alors l'affichage sera le même que l'objet appelant soit un étudiant ou une personne qui n'est pas de la classe Etudiant.

```
…
Etudiant e = new Etudiant() ;
Personne p = new Personne();
e.affiche() ;  //  va afficher : le nom
de l'étudiant est :
p.affiche() ; // va afficher : le nom
de la personne est :
p = (Personne) e ;
```

```
class  Etudiant extends Personne
{  …
   void affiche()
   { System.out.print("le nom de
         l'étudiant est : "+nom ;
   }
  …
 }
```

Lorsqu'une méthode est redéfinie, l'appel de la méthode de la classe de base ne peut se faire par un simple changement de type de référence. Dans l'exemple suivant, le changement de référence de l'objet de type Etudiant ne permet pas de changer la méthode appelée.

```
class Personne
{ …
   void affiche()
   {    System.out.print("le nom de
           la personne est :"+nom) ;
   }
   …
 }
```

Pour avoir accès à la méthode initiale définie dans la classe de base, on précède de la même façon que pour les attributs : on utilise le mot clé super.

```
class Personne
{ …
   void afficheTout()
   {   System.out.print(" nom :"+nom
       +"\ n   prenom :" + prenom
        + "\n    age : " + age) ;
   }
   …
}
```

```
class  Etudiant extends Personne
{  …
   void afficheTout()
   {   super.afficheTout() ;
       System.out.print("\n num : "+numero ;
   }
  …
}
```

Une méthode static ne peut être redéfinie que par une autre méthode static.

6.5 Les destructeurs

Nous n'avons pas encore introduit les destructeurs en Java. Quand les seules ressources utilisées par la classe se résument à des emplacements mémoires, il n'est pas utile de définir des destructeurs. Le ramasse-miettes Java s'occupe de récupérer automatiquement toute la mémoire non utilisée. Cependant, toute autre ressource réservée par les méthodes de la classe doit être libérée explicitement par des méthodes standards ou par le destructeur.
Le destructeur est appelé par java lorsqu'un objet n'est plus accessible.

Syntaxe :

```
class NomClasse
{ public finalize () {  …}
   …
}
```

Contrairement aux constructeurs, les destructeurs ne sont pas invoqués en chaîne lors de la destruction d'un objet. Il appartient au programmeur de faire appel au destructeur de la classe de base en utilisant le préfixe super. L'instruction super.finalize() définie dans une classe invoque le destructeur de la classe de base.

6.6 Les classes et méthodes abstraites

Lorsqu'on analyse la chaîne de hiérarchie d'héritage on constate qu'à mesure qu'on remonte vers la racine de l'arbre d'héritage les classes deviennent plus générales, plus abstraites. Parfois, la classe devient tellement générique qu'il devient impossible de l'instancier. Ces classes sont utilisées pour définir des concepts et non des objets.

Exemple : Construction d'une classe *FigureGéométrique* pour le traitement d'un ensemble de figures géométriques.

Parmi les méthodes de base qu'on s'attend retrouver dans cette classe, on peut citer le dessin des figures. On appellera cette méthode *dessiner()*. Etant donné que nous ne savons pas de quelle figure il s'agit, la méthode *dessiner()* ne peut donc avoir un corps. Cette impossibilité de définir le corps d'une seule méthode implique qu'il est impossible d'avoir une instance de la classe *FigureGéométrique* car si cette instance existe alors il sera impossible de lui appliquer la méthode dessiner(). La classe *FigureGéométrique* est dite *classe abstraite*. Elle sera définie comme suit :

```
public abstract class FigureGéométrique
{        … // autres méthodes et champs concrets
         public abstract void affiche() ; //permet de dessiner la figure
         … // autres méthodes abstraites ou concrètes
}
```

La classe *FigureGéométrique* va regrouper toutes les méthodes et tous les membres communs à toutes les figures. Elle va définir, en particulier toutes les méthodes pour lesquelles une implémentation est possible. Les méthodes qui doivent absolument apparaître dans les classes dérivées et dans le contenu n'est pas connu sont définies de type *abstract.*

Une méthode abstraite est une méthode dont la définition ne peut être donnée dans la classe de base mais dont la définition est supposée connue dans toutes les classes dérivées.

Ce qu'il faut retenir :

- Il n'est pas possible d'avoir un objet d'une classe abstraite ;
- Toute classe qui contient au moins une méthode abstraite est, elle aussi, abstraite ;
- Toute classe qui dérive d'une classe abstraite doit absolument définir toutes les méthodes abstraites des classes parent sinon elle sera, elle aussi, abstraite ;
- Le mot-clé *abstract* apparaît à la fois au début de la classe abstraite et comme préfixe de toutes les méthodes abstraites ;

Les figures géométriques peuvent être soit des cercles, des carrés ou toute autre figure non encore définie :

```
class Cercle extends FigureGéométrique
{        // attributs
         public Cercle(centre,rayon) { … }
         public void affiche() {…}; //dessin d'un cercle
                  …
}
```

6.7 Autres propriétés

1. Les classes dérivées n'ont pas accès aux champs privés de la classe de base ;
2. Java ne permet pas d'avoir directement de l'héritage multiple. Néanmoins, le mécanisme d'interfaces que nous verrons plus loin permet de contourner cette limitation du langage Java ;
3. Un objet de la sous-classe peut être passé en argument pour toute méthode qui attend un objet de la classe de base ;
4. Un objet de la classe de base ne peut –en principe- pas être affecté à un objet de la classe dérivée : Personne p ; Etudiant e= p ; // générera une erreur

   Il est possible de faire une conversion entre classes et sous classes sous certaines conditions. Si la référence à la classe référence un objet de la sous classe, il suffit de

faire un *cast* pour que l'affectation d'une référence de la classe à la sous classe soit possible.
Par mesure de précaution, il convient de tester si la référence de la classe de base référence bien un objet de la sous-classe en utilisant le mot clé *instanceof.*

```
Personne p ;
Etudiant e ;
…
e = p ;  // erreur de compilation
if ( p instanceof Etudiant) e = (Etudiant) p ; // ok
```

5. Le polymorphisme est la capacité de l'objet à décider quelle méthode il doit appliquer sur lui-même, selon sa position dans la hiérarchie

   Lorsqu'une méthode d'une classe est appelée dans un programme, la classe vérifie si elle dispose d'une méthode disposant des mêmes paramètres. Si tel n'est pas la cas la classe parent reprend la responsabilité de la méthode. Ce processus continue dans la hiérarchie jusqu'à ce que la méthode soit trouvée ou qu'il n'y est plus de parent.

6. Il est possible d'empêcher l'héritage de classes. Pour cela on ajoute le modificateur d'accès final qui indique que la classe ne peut être dérivée.
   final class Véhicule
   { …
   }
   Cette déclaration empêche la création de sous-classes de cette classe.
   De la même manière, il est possible d'interdire aux sous-classes de surcharger une méthode en la faisant procéder du modificateur final dans la classe parent.

   La déclaration d'une classe ou d'une méthode de type final permet une gestion statique[7], d'ou un gain en temps d'exécution (quand la déclaration n'est pas statique la gestion des liaisons se fait de façon dynamique). D'un autre coté, une définition statique d'une méthode permet un contrôle plus sûre de son exécution.

7. Le transtypage : conversion d'un type de base à un autre.
   Exemple : int x=(int) 3.14 ; // x=3
   Le transtypage est généralisé en java à tous les types. Il ne peut, cependant, s'appliquer qu'à des objets de la même hiérarchie de classes.
   Exemple :

   ```
   Personne p ;
   …
   Etudiant e =(Etudiant) p  ; // bon transtypage si p référence un objet Etudiant !
   ```

[7] Le compilateur ne peut pas remplacer une méthode simple par du code en ligne (inline), car il est toujours possible qu'une classe dérivée surcharge le code de la méthode. Par contre, le compilateur peut remplacer une méthode final par son code car celle-ci ne pourra pas être surchargée.

```
                                        // sinon erreur à l'exécution
                if (p instanceof Etudiant) e = p ; // bon transtypage même à l'exécution
```

8. Les modificateurs
   (1) private : accessible uniquement par la classe ;
   (2) public : accessible par toutes les classes ;
   (3) protected : accessible par la classe de base et les classes dérivées ;
   (4) par défaut visible uniquement dans le package.

6.8 La classe Object

En java la classe Object est dite superclasse cosmique. Toutes les classes java sont descendante de cette classe sans qu'on soit obliger de spécifier cet héritage dans la définition des classes.

Il est, par exemple possible d'écrire :

```
…
Object o=new Véhicule("Renault","laguna",2000) ;
if (o instanceof Vhicule) Vehicule laguna2=(Vehicule) o ;
…
```

On peut classer les méthodes de la classe *Object* en deux catégories :

- Les méthodes utilitaires parmi lesquelles :
  - public String toString() : cette méthode renvoie une chaîne représentant la valeur de l'objet. Cette méthode est souvent surchargée pour afficher une représentation de l'état de l'objet ;
  - public boolean equals(Object o) : dans la classe Object cette méthode vérifie si les deux instances pointent la même zone mémoire (cette définition par défaut est redondante avec le test d'égalité réalisé par l'opérateur ==). L'intérêt de cette méthode est de pouvoir la redéfinir pour comparait les champs des objet et non les références ;
  - public final native Class getClass
  - protected native[8] Object clone() throws CloneNotSupportedException : retourne une copie de l'objet sur lequel la méthode est invoquée.
- Les méthodes pour le support des threads que nous verrons prochainement.

6.9 La classe Class

Java gère pour tous les objets l'identification de type à exécution. Cette information mémorise la classe à laquelle appartient chaque objet. Cette information est accessible via une classe particulière de java baptisée : Class

[8] Native permet d'incorporer du code source écrit dans un autre langage de programmation.

La méthode *getClass()* de la classe Object permet de renvoyer une instance de type Class.

```
Véhicule a ;
Class essai =a.getClass() ;
```

Une des méthodes de la classe Class est la méthode *getName()* qui renvoie le nom de la class :

```
System.out.println(a.getClass().getName()) ;  //affiche le nom de la classe
```

La réflexion

La réflexion est caractéristique d'un programme qui analyse les caractéristiques d'une classe. La méthode Class permet de fournir un ensemble d'outils permettant d'écrire des programmes qui manipulent dynamiquement des programmes java. (Cette caractéristique est employée fréquemment dans l'architecture des composants Java : Javabeans).

Le programme qui peut analyser les caractéristiques d'une classe est dit programme réflecteur. Le package qui offre cette fonctionnalité s'appelle java.lang.reflect.

Vous pouvez étudier ce package sur lequel nous reviendrons plus tard.

6.10 Quelques conseils pour l'utilisation de l'héritage

- Placer les méthodes et les champs communs dans la classe de base ;
- Utiliser l'héritage pour déterminer la relation d'appartenance ;
- N'employer l'héritage que si toutes les méthodes héritées restent valables ;
- Utiliser le polymorphisme plutôt que l'information de type :
  Exemple : si (x est de type1) instruction1(x)
  Sinon (si x de type2) instruction2(x)
  Si instruction1 et instruction2 représentent un concept commun, il est plus intéressant d'en faire un concept d'une classe parent des deux classes et d'exécuter x.instruction()

6.11 Les interfaces

Une interface est une sorte de classe abstraite spéciale. Toutes les méthodes d'une interface sont implicitement *public* et *abstract*. Toutes les données (attributs) sont implicitement des constantes *static* et *final*. Une interface est donc totalement abstraite.

Une interface permet de définir un contrat fonctionnel avec toutes les classes qui décident de l'implémenter : une classe qui implémente une interface doit impérativement concrétiser (définir) toutes les méthodes de l'interfaces en respectant impérativement les signatures des méthodes. Si une classe implémente une interface sans définir au moins une méthode, cette classe devient abstraite.

Définition d'une interface

```
interface NomDELInterface extends AutreInterface1, AutreInterface2
{       // déclaration de constantes
        // déclaration des signatures des méthodes
}
```

Exemple : Cette interface est définie dans la bibliothèque standard Java (package java.lang). Elle indique que les éléments d'une classe peuvent être comparés.

```
interface Comparable
    {   public int compareTo(Object o) ;
            //  Ceci indique que toute classe qui implantera cette
            //  interface doit définir une méthode compareto
    }
```

.

Il est important de noter que l'implémentation des méthodes des interfaces dans la classe privée sera toujours public. Java ne dispose pas de moyen permet de restreindre l'accès aux méthodes héritées.

Les interfaces peuvent être utiliser pour gérer les constantes. Elles sont surtout utilisées pour implanter l'héritage multiple qui normalement n'est pas possible en Java.

Héritage multiple :

Prenons l'exemple suivant : les animaux peuvent se subdiviser en mammifères, en reptiles, en insectes, etc. Ils peuvent aussi être subdiviser en carnivores, herbivores, etc. Si nous considérons l'exemple du chien : il est à la fois carnivore et mammifère. Une chèvre est, par contre, mammifère et herbivore.

La chèvre et le chien partagent les caractéristiques des mammifères. La possibilité donnée à une classe de dériver de plusieurs parents est dite *héritage multiple*.

En Java, une classe ne peut pas être dérivée directement de plusieurs classes. En revanche, la notion d'interface peut être utilisée pour ce type d'héritage.

Une interface annonce qu'une classe implantera certaines méthodes ayant une signature particulière. Le mot clé utilisé pour signaler qu'une classe tiendra la promesse d'implanter les méthodes est *implements*. Une classe peut implémenter plusieurs interfaces.

```
class NomClasse [extends NomClasse] implements NomInterface1,
                                                NomInterface2, NomInterface3
{        …
}
```

Exemple : la classe Double implémente l'interface Comparable.

```
class Double extends Number  implements Comparable
{     int compareTo(Object o)
      { …
      }
      …
}
```

6.11.1 Quelques propriétés des interfaces

- Les interfaces peuvent être déclarées mais ne peuvent pas être instanciées par new ;
- Une fois qu'une interface est définie, on peut déclarer qu'une variable objet appartient à cette interface

```
Comparable x =new Double() ;
Double y= new tableau() ;
 Comparable c =y ;
If (x.compareto(y)<0) {…}
```

- Comme pour les classes, on peut vérifier si un objet appartient ou pas à une interface en utilisant le mot clé *instanceof*

```
if (a instanceof Comparable) …) { …}
```

- Les interfaces ne contiennent pas des variables d'instances ou des méthodes statiques. Mais, elles peuvent avoir des constantes.
- Les interfaces peuvent implémenter d'autres interfaces (*extends*). Ceci permet une spécialisation des activités des interfaces.
- Une classe peut implanter plusieurs interfaces

```
public NomClasse extends NomClasseDeBase implements NomInterface1, NomInterface2
```

Exemple d'interface :

```
public interface shape
{       public double area();
        public String getname();
}

public class Point extends Object implements shape
{               // déclaration des données membres
        public double area() {return 0}
        public double volume () {return 0}
        public String getname() {return "point";}
}

interface Affichable
{       void afficheMoi(int x,int y);
}
class Alien implements Affichable
{       public void afficheMoi(int x, int y)
        {
                // code servant á afficher un Alien
        }
}
class Chat extends Felins implements Affichable
 {      public void afficheMoi(int x, int y)
        {
                // code servant á afficher un chat
        }
}
```

Utilisation des interfaces:

```
Chat lechat=new Chat();
Alien  alien1=new Alien();
Affichable ObjetAffichable;
ObjetAffichable=lechat ;
ObjetAffichable.afficheMoi(10,10);

ObjetAffichable=alien1;
ObjetAffichable.afficheMoi(50,50);
```

TD-TP n° 4 : Héritage

Exercice 1 :

Pour calculer le prix de vente des véhicules, une taxe de 5% est ajoutée au prix indiqué par la variable prix. Si le modèle est un modèle français, une taxe supplémentaire de 1% est ajoutée après la surtaxe de 5%.

- Ecrire la classe Véhicule qui définie quelques méthodes et membres relatifs aux véhicules. Inclure en particulier la méthode calculPrixVente() ;
- Ecrire les deux classes VéhiculeFrancais et VéhiculeNonFrancais. Puis, redéfinir la méthode calculPrixVente().

Exercice 2 :

Ecrire le programme qui permet de compter le nombre d'objets de la classe personne disponible à tout moment en mémoire centrale.

Exercice 3 :

1. Ecrire une classe PileDEntiers de n entiers avec les méthodes estPleine(), estVide(), void empiler(int), int depiler(), et un constructeur PileDEntiers(int taille).
   Empiler dans une pile pleine est erreur. Dépiler une pile vide aussi.
2. Ecrire une class PileDEntiers2 qui hérite de PileDEntiers et dispose en plus d'une méthode getTaille() qui donne le nombre de valeurs encore dans la pile

Exercice 4 :

Etudier les classes : java.lang.Object, java.lang.Class
Ecrire le programme qui permet d'afficher les information concernant une personne en utilisant tout simplement l'instruction System.out.print(objetPersonne).
(penser à la fonction toString de la classe Object)

Exercice 5 :

Ecrire une classe Calculette qui utilise les paramètres de la ligne de commande pour faire les 4 opérations (+, -, x et /) sur des entiers, plus afficher le résultat avec =, le tout en notation postfixée (c'est-à-dire 1 2 + donne 3).

Exemples d'utilisation :

```
java Calculette 1 2 + =
3
```

```
java Calculette 1 2 + 5 1 - = x =
4
12
```

Indications :

- ne surtout pas utiliser le * pour indiquer une multiplication car * est un méta-caractère du shell !
- utiliser la classe PileDEntiers ou PileDEntiers2 pour stocker les résultats intermédiaires.
- l'algorithme est un très simple analyseur syntaxique : pour chaque paramètre de la ligne de commande :

```
Si c'est une opération +, -, x ou /   alors
        depiler 2 valeurs
        exécuter le calcul
        empiler le résultat
sinon   Si c'est un  =  alors
                dépiler 1 valeur
                afficher la valeur
                empiler la valeur
        sinon    // c'est un nombre
                empiler la valeur
```

- utiliser la méthode parseInt de la classe Integer pour obtenir un int à partir d'une String

Exercice 6 :

(1) Écrire une classe Point représentant un point dans le plan, disposant d'un constructeur qui accepte deux entiers x et y pour coordonnées. Ajouter un constructeur par défaut. Définir une méthode de translation translate() et une méthode de comparaison du point avec un autre de signature boolean same(Point p).
Créer un objet de la classe Point et le désigner par deux références p1 et p2. Comparer ces variables avec == et avec same(). Effectuer une translation sur p1 et refaire les comparaisons. Écrire une méthode toString() permettant de représenter le point sous la forme d'une String de la forme (x,y).

(2) Essayer le code suivant:

```
Point p1 = new Point(1,1);
Point p2 = new Point(1,1);
System.out.println(p1.equals(p2));
```

Qu'affiche ce code? Pourquoi le programme compile? Que changer dans la classe Point pour que ce code affiche ce qu'on attend? Faire cette modification et vérifier que cela fonctionne.

(3) Essayer ensuite le code suivant:

```
Object o1 = new Point(1,1);
Object o2 = new Point(1,1);
System.out.println(o1.equals(o2));
```

Expliquer le comportement obtenu. Faire les modifications adéquates dans la méthode equals() de la classe Point de sorte qu'elle *redéfinisse* celle appelée dans l'exemple ci-dessus.

(4) Comment définir la méthode equals() de la classe ColoredPoint? En particulier, tester et commenter la portion de code suivante:

```
Point p = new Point(1,1);
ColoredPoint cp = new ColoredPoint(1,1,Color.red);
System.out.println("p.equals(cp) vaut "+p.equals(cp));
System.out.println("cp.equals(p) vaut "+cp.equals(p));
```

On désire maintenant disposer d'un *type*, représentant des points, sur lequel les méthodes suivantes soient disponibles:

```
float getX()
float getY()
void setX(float x)
void setY(float y)
boolean equals(Object o)
String toString()
void translate(float dx, float dy)
```

Néanmoins, on veut avoir le choix de stocker les coordonnées x et y d'un point **soit** sous la forme d'un int, **soit** sous la forme d'un float.

(5) Réaliser le type attendu en utilisant une interface (mot-clé interface), avec deux classes PointImplInt et PointImplFloat réalisant cette interface (mot-clé implements).
Pouvez vous déceler des parties de codes dupliquées ou tout du moins *factorisables*, entre ces deux implémentations de l'interface?

(6) Factoriser le maximum de code dans une *classe abstraite* APoint (mot-clé abstract class), dont deux classes *concrètes* PointInt et PointFloat hériteront.

On veut maintenant pouvoir ordonner les points, en comparant leurs abscisses et, en cas d'égalité, leurs ordonnées.

(7) Ecrire une méthode sortArray() qui accepte en argument un tableau de points et qui le trie suivant cet ordre. On pourra pour cela créer un comparateur de points implémentant l'ionterface l'interface java.util.Comparator et se servir de méthodes de la classe java.util.Arrays. (Question à traiter plus tard)

Exercice 7 :

Ecrire une classe Superficie qui contient une méthode abstraite calculSuperficie(), qui calcule la superficie d'une forme géométrique et une méthode concrète additionSuperficie(double[] a).

- Compléter ce programme en écrivant deux classes Carré et Cercle qui héritent de la classe Superficie et qui implémentent la méthode calculSuperficie() ;
- Définir dans le programme principale un tableau qui peut contenir des carrés et des cercles et le bloc d'instructions qui permet d'additionner les superficies.

Exercice 8 :

Ecrire une interface FormeGéométrique qui définit une méthode Dessiner() et une méthode Définition().

La méthode Dessiner() affiche le type de la figure géométrique, la méthode Définition() renvoie une chaîne de caractères qui définit la forme géométrique.

Compléter les classes Carre et Cercle pour qu'elles implémentent l'interface FormeGéométrique.

Exercice 9 :

Ecrire une classe Véhicule qui définit quatre méthodes :

- demarrer() : méthode abstraite qui ne renvoie aucun paramètre
- rouler() : méthode concrète avec un corps vide, elle ne renvoie aucun paramètre
- arrêter() : méthode concrète avec un corps vide, elle ne renvoie aucun paramètre
- toString() : cette méthode renvoie la chaîne de caractères « je suis un véhicule »

2. Ecrire une classe concrète Voiture qui hérite de la classe Véhicule. La classe Voiture surcharge la méthode toString.

Exercice 10 :

- Ecrire une interface PeutSeBattre() qui déclare une seule méthode : void seBattre()
- Ecrire une classe HommeDAction qui définit une méthode seBattre(). Cette méthode se contente d'afficher le message "peut se battre".
- Ecrire deux interfaces : PeutVoler() et PeutNager(). Elles implémentent respectivement les méthodes vole() et nage().
- Ecrire une classe Héros qui hérite de la classe HommeDAction et qui implémente les trois interfaces PeutVoler(), PeutSeBattre() et PeutNager(). (n'implémenter que les méthodes qui génèrent des erreurs de compilation si elles ne sont pas implémentées)

7 Cours n° 7 : Packages

7.1 Les packages

Les packages permettent de regrouper les classes dans un ensemble. La bibliothèque java est organisée en packages : java.lang, java.net, etc. Tous les packages java sont organisés en hiérarchie dont la tête est le package java.

Plus concrètement, un package désigne un ensemble de classes compilées (*.class), se trouvant dans le même répertoire. La position ou se trouve le fichier compilé défini le nom du package en remplaçant les séparateurs de répertoires par des points[9]. Exemple :
Le package : *javaCore/monpackage/lesEntiers* aura pour nom *javaCore.monPackage.lesEntiers*

Lorsque les utilisateurs ajoutent des packages, il est préférable de les mettre dans un package ayant le nom corejava.uti. il est possible de spécifier autant d'imbrications que nécessaire. Pour éviter les conflits entre les noms de package, Sun propose de faire procéder les noms des packages par l'adresse Internet. Exemple, si votre adresse est aaa.com, alors votre corejava sera noté com.aaa.corejave.

7.2 Utilisation des différents packages du kit java:jdk1.3

Par défaut lorsque le compilateur cherche une classe, il va chercher dans le package java.lang. Mais si on utilise une classe qui n'est pas dans ce package, il faut indiquer au compilateur où se trouve le package contenant la classe.

Pour utiliser une classe qui se trouve dans un package fourni par le kit java, on peut :

1. Spécifier le nom complet du package : int i=corejava.Console.readInt() ;
2. Importer une classe, puis utiliser ses champs et méthodes (directive import)

Exemple: utilisation de la classe Date

[9] Pour des raisons d'économies de place, une arborescence de packages peut être compressé dans un fichier « .zip ».

```
java.util.Date laDate = new java.util.Date();

Ou bien

Import java.util.Date ;
Date = new Date() ;
```

La localisation des package n'est pas normalisée. Sous windows et unix, le chemin de la classe est donné par la variable d'environnement CLASSSPATH (CLASSPATH=c:\jdk).

Sous windows la variable d'environnement classpath peut être définie à l'aide de la commande : `set classpath = .c:\java ;c:\java\util`

Sous unix, la syntaxe sera : `setenv classpath .:/ ~/java ; ~/java/util`

- Le nom d'une classe est identifié en la faisant précéder du nom du package dans lequel elle est située ;
- Deux classes peuvent avoir le même nom si elles ne sont pas dans le même package ;
- Un nom de package doit obligatoirement correspondre à un nom de répertoire.

Exemples

- (A l'IUT de villetaneuse), la classe Concole se trouve dans le package corejava. Pour pouvoir utiliser cette classe, il faut importer le package corejava se trouvant dans le répertoire t:\ens\javacore ;
- La classe Integer se trouve dans le package lang qui lui se trouve dans le package java d'où le nom de la classe : java.lang.Integer
- Le package util qui se trouve dans le package java contient une classe Date
  Le nom de la classe est: java.util.Date
  ➔ il existe un répertoire java dans lequel il y a un répertoire util dans lequel se trouve la classe Date.

7.3 Création de packages

Lorsqu'on utilise un package non fourni par le jdk, il faut indiquer au compilateur où se trouve le répertoire contenant l'ensemble des classes définies dans ce package.

Lorsqu'on crée une classe sans indiquer de package, le compilateur considère que la classe est dans un package par défaut. Le répertoire correspondant est le répertoire où on met la classe. On peut ainsi avoir plusieurs classes définies dans le même répertoire. Toute classe définie dans le package par défaut est donc visible et utilisable par les autres classes définies dans le package par défaut.

Exemple de création de package : Soit à créer le package mon*corejava* qui va contenir l'ensemble de mes classes réutilisables et qui sera placé dans le répertoire c:\tpsJava.

- On peut commencer par la création des fichiers contenant des classes du package. Ces fichiers doivent commencer (première ligne) par l'instruction : `package moncorejava` [10] . Soit MaClasse un exemple de classes du package moncorejava ;
- Les classes sont ensuite compilées. Ceci peut se faire de deux manières différentes
    - En utilisant la commande `javac –d c:\tpsJava MaClasse`. Dans ce cas, le fichier MaClasse.class générée par le compilateur java est mis dans le répertoire c:\tpsJava\moncorejava. Si le répertoire moncorejava[11] n'existe pas celui-ci est créer automatiquement.
    - En utilisant la commande : javac MaClasse.java puis en créant puis décalant manuellement le fichier MaClasse.class dans le répertoire moncorejava.
- Il faut préciser au compilateur où se trouve le répertoire mon*corejava* en utilisant la variable d'environnement *CLASSPATH* :

    u:\algo :> set CLASSPATH = . ; c:\tpsJava\moncorejava

7.4 Utilisation des classes d'un package

Si on considère l'exemple précédent, ou on dispose d'un package moncorejava qui contient une classe MaClasse alors pour utiliser la classe MaClasse dans une autre classe, utilisera :

- Le nom moncorejava.Maclasse (voir section 1) ou bien
- En importe cette classe ou toutes les classes situées dans le packages en utilisant l'instruction : *import moncorejava ;*

Puis, on accède aux méthodes ou données des classes du package moncorejava sans les faire précéder du nom du package (on référence les classes importées par des noms abrégés et non par des noms complets).

Remarques :

1 - L'instruction import, ne provoque pas le chargement des classes, elle permet simplement et uniquement d'alléger la syntaxe d'accès aux classes.

2 - Le lien entre une classe et une classe importée se fait à l'exécution. On appelle cela une édition de liens dynamiques ;

[10] Si cette instruction est omise, les classes feront implicitement parties du package par défaut et le fichier compilé doit résider dans le répertoire courant.

[11] Le nom du répertoire doit être impérativement celui du package.

8 Tableaux, chaînes de caractères et structures de données

8.1 Les Tableaux

Les tableaux, en java, sont des structures pouvant contenir un nombre fixe d'éléments de même nature.

8.1.1 Déclaration d'un tableau

Les deux syntaxes suivantes sont équivalentes :

```
TypeElementDuTableau [] nomDutableau ;
TypeElementDuTableau nomDuTableau [] ;
```

Elles permettent de définir une référence vers un tableau. La tableau lui-même n'est pas encore défini.

Exemples :

```
int  pileEntiers[] ;
Etudiant [] listeDEtudiants ;
```

8.1.2 Initialisation d'un tableau

Initialiser un tableau consiste à le créer. L'initialisation indique toujours la taille du tableau qui reste fixe.

```
nomDuTableau = new TypeElementDuTableau[taille]
```

Exemple :

```
int pileEntiers[] ;
pileEntiers = new int[100] ;
```

Un tableau peut être déclaré et initialisé en même temps :

```
TypeElementDuTableau [] nomDutableau = new TypeElementDuTableau[taille] ;
```

L'initialisation et donc la réservation d'espace mémoire pour les objets du tableau peut se faire au moment de son utilisation c-à-d. à l'exécution.

Les indices du tableau commencent à partir de 0 jusqu'à la valeur taille-1. Pour accéder à une position se trouvant à une position *indice* d'un tableau *nomDuTableau* en commençant à partir de zéro, on écrit *nomDutableau[indice]*. Les éléments *nomDutableau[0],…,nomDutableau[taille-1]* sont des objets ou des types de base de type *typeElementDuTableau.*

8.1.3 Initialisation des éléments du tableau

Lors de l'initialisation du tableau ses éléments sont automatiquement initialisés.
Le programme suivant :

```
class Test1
{       public static void main (String[]a)
        int [] entiers = new int[10];
        String [] chaine = new String [2];
        System.out.println("("+chaine[0]+","+chaine[1]+","+x[4]+")");
}
```

va produire le résultat suivant : (null,null,0)

Java initialise automatiquement les éléments du tableau à 0 pour les types de base byte, short, int et long. Il initialise à 0.0 les réels (float et double), false les boolean et à null les objets.

8.1.4 Les tableaux de types de base

Les tableaux de types de base peuvent être initialisés au moment de la déclaration comme suit : `int [] entiers ={1,2,5,68} ; // tableau de quatre éléments`
ou bien :

```
int a =1,  b= 68 ;
int [] entiers ={a,2,5,b} ;
```

Il peut aussi contenir des variables de types différents :

Exemple :

```
byte a =1 ;
short b =2 ;
long c = 5 ;
float d =5 ;
char e = 3 ;
long [] reels ={a,b,c,d,e} ; // a,b,c,d,e sont sur-castés vers le type supérieur.
```

Il n'est, cependant, pas possible de faire un casting lors de l'affectation.
Le programme suivant :

```
int [] a = {3,4,5} ;
short [] b = {4,5} ;
a =b ;
a[1] = 100 ;
```

provoque une erreur à la compilation (impossible de convertir un short en un int).

8.1.5 Les tableaux d'objets

Les tableaux d'objets peuvent aussi être initialisés au moment de la déclaration par des objets anonymes :

```
Etudiant [] etudiants ={new Etudiant(),new Etudiant(), new Etuidiant()} ;
```

De la même façon que pour les autres objets, les tableaux peuvent être "surcastés" :

```
Etudiant []etudiants  ={new Etudiant(),new Etudiant(), new Etudiant()} ;
Personne [] p ={new Personne(),new Personne()} ;
p=etudiants;
```

Il est, aussi, possible de faire les initialisations suivantes :

```
Personne p [] = new Etudiant[]{new Etudiant(),new Etudiant()} ;
        // application du polymorphisme qui consiste à initialiser les élément d'un tableau d'une
        // classe et les affectés à une classe parent
Personne p [] = new Personne[]{new Etudiant(),new Etudiant()} ;
        // le sur-casting est fait au niveau des éléments et non au niveau du tableau lui-même
```

Par contre, il n'est pas possible d'avoir :

```
Etudiant p [] = new Personne[]{new Personne (),new Personne ()} ;
```

De la même façon qu'avoir des objets anonyms, on peut faire l'initialisation d'un tableau comme suit :

```
Personne p1 = new Personne() ;
Personne p2 = new Personne() ;
Personne p3 = new Personne() ;
Personne p [] = {p1,p2,p3} ;
```

8.1.6 Taille d'un tableau et la variable length

La taille d'un tableau est fixe. Une fois initialisée, elle ne peut plus être modifiée. Cette taille n'est pas nécessairement connue à la compilation.

Pour connaître la taille du n'importe quel tableau, il suffit de consulter la variable length.
La consultation de sa valeur se fait comme suit :

```
Personne p [] = new Personne[]{new Etudiant(),new Etudiant()} ;
System.out.println("la taille du tableau est" + p.length) ;
```

8.1.7 Tableaux multidimensionnels

Comme les tableaux sont des objets, les tableaux de tableaux multidimensionnels sont implémentés comme des tableaux de tableaux.

Exemples :

```
int tableau[][][] = new int [10][10][10] ;
int  [][]a  = {{1,2,3},{4,5,6}};
```

Comme il s'agit de tableaux de tableaux, il est possible de créer des tableaux triangulaires ou en losanges.

```
int [][] a = {{1},{2,3},{4,5,6}} ;
int [][]b ={{1},{2,3},{4,5,6},{7,8},{9}} ;
```

La syntaxe int [][]a = new int [10][20] ; créée une matrice. Si vous voulez créer un tableau à deux dimensions différent d'une matrice, il faut faire une initialisation tableau par tableau.

Que fait le programme suivant ?

```
int [][] a =new int[3][] ;
for (int I=0;i<3;i++) {a[i]=new int[i+1];}
```

8.1.8 Les passages d'arguments

Un tableau comme n'importe quel autre objet peut être passé comme argument lors des appels de méthodes. Et comme les autres objets seul la référence vers les données du tableau est passée en argument.

8.2 Les chaînes de caractères

8.2.1 la classe String

Le langage Java ne contient pas de classe préfini pour les chaînes de caractères. La bibliothèque standard (java.lang) contient une classe String dédiée au traitement des chaînes de caractères. Elle dispose d'environ 50 méthodes.

Exemple :

- Concaténation d'une chaîne avec un autre type donne une chaîne

```
String a=  "moi" ;
String b=  "toi" ;
String c = a+  "et "+b ; //  c va recevoir  la chaîne "moi et toi"
String d =  "ABC"+24 ; // d aura comme valeur  "ABC24"
```

- Sous-chaîne : d=c.substring(0,3) ; // d= "moi"
- Longueur d'une chaine : a.length() ;
- Accès à un caractère de la chaîne : a.charAt(3)='i' ;
- Comparer deux chaînes :

```
a.equals(b) ; // test d'égalité
a.equalsIgnoreCase(b); test d'égalité sans prise en compte de la casse
```

Important : le test (a==b) vérifie uniquement si les deux chaînes a et b référence une chaîne stockée dans le même emplacement.

Exemple d'utilisation des chaînes de caractères : paramètre de la méthode main

Le point d'entrée dans une classe est la méthode main qui contient en paramètre un tableau d'arguments: public static void main(String args[])

Ces arguments sont des paramètres qui peuvent être passés à la méthode main au moment de l'exécution du programme (args[0] contient le premier paramètre : param1, args[1] contient le deuxième paramètre : param2, etc.).

Pour connaître le nombre d'arguments passés, il suffit de faire : args.length

Lors de l'exécution d'un programme, on écrit

```
java nom-classe param1 param2 …
```

Remarque il n'est pas possible de modifier individuellement les caractères d'une chaîne.
Pour cela java utilise une autre classe : StringBuffer

8.2.2 La classe StringBuffer

La classe StringBuffer manipule des chaînes de caractères dans la taille est amenée à variée. La classe String manipule des chaînes de taille fixe.

Par exemple dans le cas des String, l'utilisation de l'opérateur « + » permet de changer les valeurs des chaînes de caractères ainsi que leur taille en référençant de nouvelles zones mémoires.

Exemple : String a = « bon » ;
a= a + « jour » ;
a= « un » +a ;

Le a référence à chaque fois une zone mémoire différente. (la vitesse d'exécution est lente).

Avec Stringbuffer :

```
StringBuffer a= new «un » ;
        a=a .append( « bonjour » );
```

8.2.3 La classe StringTokenizer

Cette classe est définie dans le package util. Lors de son utilisation il faut ajouter au début du programme la ligne : import java.util.StringTokenizer ;

Cette classe permet de récupérer les mots ou les termes contenus dans une chaîne
Public StringTokenizer (String a, String b) ;

a est la chaîne à stocker dans l'objet, b désigne le délimiteur

Exemple :

```
…
String s1= « un, deux, trois, quatre, cinq » ;
StringTokenizer st1=new StringTokenizer(s1, « , « ) ; // « , « est un séparateur
While (st1.hasMoreTokens())
{ …}
```

TP : étude de la classe StringBuffer et la classe StringTokenizer.

8.3 Quelques structures de données

8.3.1 BitSet

Cette classe permet de gérer un ensemble de bits implémenté sous forme d'un vecteur de bits. La taille du vecteur croît en fonction des besoins. La taille minimum d'un BitSet est celle d'un long, soit 64 bits.

Un BitSet est utile si on veut stocker efficacement un grand nombre d'informations 0-1. Cette classe n'est efficace toutefois que sur le plan de la taille ; si le but recherché est un accès performant, mieux vaut se tourner vers un tableau de quelque type natif.
La signature de la classe est :

public class BitSet implements Cloneable, Serialisable

Parmi les methods de cette classe, on y retrouve :

pubic void and (BitSet a)
public void or(BitSet a)
public int size()
public void xor(BitSet a)

Exemple :

```
short st = (short)rand.nextInt();
BitSet bs = new BitSet();
for(int i = 15; i >=0; i--)
  if(((1 << i) &  st) != 0)
    bs.set(i);
  else
    bs.clear(i);
System.out.println("short value: " + st);
printBitSet(bs);

BitSet b127 = new BitSet();
b127.set(127);
System.out.println("set bit 127: " + b127);
```

8.3.2 Vector

Cette classe permet une gestion dynamique de la taille du tableau. La classe Vector contient des éléments de type Object (n'importe quel type). Cette caractéristique nécessite un transtypage lorsqu'on désire récupérer un élément du vecteur dans un type plus spécifique que Object.

Lors de la création du vecteur, une taille peut être spécifier (l'espace n'est cependant utilisé qu'en cas de besoin). Si la capacité alloue préalablement est dépassée, le vecteur se cherche une place plus grande ou il fera une copie des éléments déjà stockés. Par défaut, la taille est doublée à chaque nouvelle allocation. Pour éviter une croissante exponentielle de l'allocation, un autre paramètre régulant l'augmentation peut êtres spécifier lors de la construction du vecteur. Vector(x,n) indique que la taille maximale initiale de x, et qu'en cas de dépassement l'augmentation se fait par des pas de n éléments.

Quelques méthodes :

- size() : donne la taille du vecteur
- trimTosize() réajuste la taille du bloc au strict nécessaire. Utile à la fin de mise à jour de la taille du vecteur
- add (Object) : ajoute un objet au vecteur

- setSize(int n) : spécifie la taille du vecteur à exactement n éléments. Ceux qui dépassent sont éliminés.
- ElementAt(i) ou get(i) permet de récupérer l'élément situé à la position i
- SetElementAt(x,i) ou set(i,x) permet de mettre l'objet x à l'emplacement i

8.3.3 Stack

Elle permet de gérer des structures de données de type pile. La classe Stack dérive de la classe Vector (?).
Signature de la classe Stack :

```
public class Stack extends Vector
```

Parmi les méthodes de cette classe – en plus des méthodes de la classe Vector- on y retrouve :

```
public boolean empty()
public synchronized Object pop()
public synchronized Object push(Object a)
public synchronized int search(Object o)
```

8.3.4 Dictionary

8.3.5 Hashtable

TD-TP n° 5 : Les tableaux et les structures de données

Exercice 1 :

Ecrire la classe Tableau.moncorejava qui définit un tableau d'objets de type Integer et qui implémente les méthodes suivantes :

a. Le constructeur par défaut qui initialise le tableau à 10 éléments et le constructeur avec un paramètre qui donne la taille du tableau ;
b. La méthode qui affiche les objets référencés par le tableau ;
c. Les méthodes minTableau et moyenneTableau qui calculent, respectivement, le minimum et la moyenne ;
d. La méthode renverse(int[] t) qui renverse l'ordre des éléments du tableau t (sans créer de tableau auxiliaire).
e. La méthode fréquenceMax qui renvoie le premier indice de la valeur la plus fréquente du tableau et qui permet également de l'afficher à l'écran (on utilisera la méthode Integer.parseInt(String s));
f. Ecrire la méthode qui réalise le tri par sélection, par bulle ou par insertion en fonction du paramètre de la fonction.

Rappel :

- Tri par sélection
  On recherche le plus petit élément t[m] de la suite t[0] ... t[N-1] et on l'échange avec t[0]
  On recherche le plus petit élément t[m] de la suite t[1] ... t[N-1] et on l'échange avec t[1],etc.
- Tri Bulle
  On parcourt la suite t[0] ... t[N-1] en échangeant les éléments consécutifs tels que t[j-1] > t[j]
  On parcourt la suite t[0] ... t[N-2] en échangeant les éléments consécutifs tels que t[j-1] > t[j], etc.
- Tri par insertion
  On suppose les i-1 premiers éléments triés et on insère le i-ème à sa place parmi les i premiers.

Exercice 2 :

Un concessionnaire qui vend des véhicules de toute marque désire avoir le tableau des prix de vente de son stock de véhicule. Supposant que son stock est construit à partir du programme suivant :

```
VehiculeFrancais twingo=new VéhiculeFrancais(« Renault, »Twingo2»,1999) ;
        VéhiculeFrancais p206=new VéhiculeFrancais(« Peugeot, »206cc »,2000) ;
        Véhicule a3=new Véhicule(« Audi », »A3 ambition », 1999) ;
        Véhicule [] stock= new Véhicule[3] ;
        stock[0]=twingo ; stock[1]=p206 ; stock[2]=a3 ;
```

L'exécution de l'instruction : for (int i=0 ;I<3 ;i++) stock[i].PrixVente() ; permet de calculer le prix de vente de chaque voiture (les voiture françaises seront automatiquement surtaxées de 1%).

Exercice 3 : Gestion d'entrepôts

On se propose de réaliser une simulation de gestion de stocks. Pour cela, on a besoin de manipuler des objets Entrepots. Un Entrepôt est caractérisé (notamment) par un nom, et possède un stock. Pour simplifier, on considèrera que les objets stockés sont tous de la même taille, et sont identifiés simplement par une chaîne de caractères. Le stock est donc un tableau tridimensionnel de chaînes de caractères.

La chaîne "X" signifie que l'objet est à mettre à la casse.

On désire en fait étendre le concept d'entrepôt. On souhaite disposer également d'entrepôts pouvant stocker des données périssables, c'est-à-dire où les produits sont caractérisés cette fois par une chaîne de caractères et une date (qu'on modélisera ici avec une variable entière). On souhaite encore étendre le concept d'entrepôt afin qu'il permette de gérer des entrepôts frigorifiques, caractérisés par une température maximale.

1. Déduire de ces considérations la manière dont vous allez modéliser les différents types de produits ;
2. Ecrire les spécifications des classes Entrepot, EntrepotPer et EntrepotFrig sans implémenter les méthodes.

Doter les classes de constructeurs. On supposera que le constructeur de la classe Entrepot fait appel `a une méthode saisieStock(). Cette méthode permet de saisir le contenu du stock ; son implémentation n'est pas demandée.

3. Implémenter une méthode affiche() qui permette d'afficher le contenu du stock.

On souhaite maintenant doter les classes d'une méthode casse() qui signale tous les objets à mettre à la casse, et d'une méthode controle(), pour détecter si l'entrepôt est vide. La méthode casse(int date) doit en outre, pour les produits périssables, vérifier que la date limite de consommation n'est pas inférieure `a la date courante.

En ce qui concerne la méthode controle(), dans le cas des entrepôts frigorifiques,
elle doit vérifier également que la température courante n'est pas supérieure ou égale
à celle tolérée.

4. Donner les spécifications des méthodes casse() et controle() dans les classes où c'est nécessaire. Justifier les choix, en particulier l'utilisation des modificateurs de visibilité.
5. Implémenter les méthodes.

Exercice 4 :

La classe Technique suivante dispose d'un champ chaîne de type String et d'un champ tab de type tableau d'entiers.

```
class Technique
{  String chaine;
   int [] tab;
   public Technique (String chaine, int [] tab)
   {  chaine=chaine;
      tab=tab;
   }

    void affiche() f
    {   System.out.println("chaine: " + chaine);
```

```
            System.out.println("tableau: ");
            for (int i=0;i! tab.length;i++)
            System.out.print( tab[i] + ", ");
            System.out.println();
        }
        public static void main (String args[])
                                {   String c="test";
            int [] tab=f3, 4, 5, 6g;
            Technique t=new Technique(c,tab);
             t.affiche();
             c="lkds";
            for (int i=0;i!tab.length;i++) tab[i]=0;
             t.affiche();
         }
}
```

Déterminer ce que produit l'exécution de ce programme et si cela vous paraît correct, en particulier en ce qui concerne l'encapsulation des données.

Exercice 5 : Tours de Hanoi

On désire créer un jeu de tours de Hanoi. Pour cela, on veut utiliser une classe pile d'entiers (supposée développée au préalable).

1. Créer les spécifications de la classe Pile, dotée des méthodes permettant de l'utiliser. Implémenter les méthodes et constructeurs. En particulier, la pile doit pouvoir s'agrandir dynamiquement en cas de débordement de capacité.
2. Définir une classe Tour, qui permette d'empiler deux étages uniquement si le nouvel étage est plus petit que le haut de la tour.
3. Définir une classe Hanoi, qui initialise un plateau constitué de trois tours, et dotée d'une méthode pour déplacer les étages. Utiliser les modificateurs de visibilité, et justifier leur emploi.

9 Cours n° 9 : Les exceptions

9.1 Introduction

Java se trouve aussi bien sur les ordinateurs que sur les systèmes embarqués (téléphones cellulaires, ordinateurs de poche, etc.). Pour limiter la possibilité d'erreurs dans les programmes, il est utile d'identifier et de traiter les éventuelles erreurs d'une manière rigoureuse et systématique.

Dans certains langages, comme le C, le traitement des erreurs incombe aux programmeurs. Il n'y a aucune aide provenant du langage lui-même (en général, une routine manifeste une erreur en retournant une valeur –1 ou null). En tant que développeur, vous devez savoir quelle est la valeur qui indique une erreur et quelle est sa signification.

Java utilise un mécanisme d'interception des erreurs élégant grâce au traitement des exceptions. Ce traitement ressemble à ce qui ce fait en C++ et en Delphi.

Une exception indique un état inhabituel ou une erreur. La gestion des erreurs est séparée du code. Lorsqu'une exception est détectée, le contrôle du programme est transféré à un programme particulier où elle est capturée et traitée.

Une méthode java doit spécifier les exceptions qu'elle doit lever, c'est-à-dire celles qu'elle ne capture pas elle-même. Cela signifie que le compilateur peut s'assurer qu'elles sont bien gérées par le programme. Vous pouvez décider de ne pas traiter les erreurs importantes, mais vous devez le faire de façon explicite.

9.2 Lever d'exceptions

La syntaxe suivante indique que les exceptions levées dans la zone try sont capturées dans la zone catch.

```
try
{       // Code pouvant lever des exceptions
} catch (Exception e)
{       // capture et gère les exceptions et des sous-classes de Exception levées
        // dans la partie try
}
```

Interprétation : le bloc d'instructions encadré par le mot clé try s'exécute normalement. Lorsqu'une exception est détectée, un objet de la classe exception est crée. Cet objet est transmis à la partie du code encadré par le mot clé catch. Le mot clé catch est suivi du type de l'exception (ici Exception). Le nom de l'objet Exception « e » est donné à l'objet transmis par le bloc try, ainsi si on désire afficher le nom de l'exception dans le bloc catch il suffit d'exécuter l'instruction : System.out.println(e) ;

Exemple :

```
// Essai.java
 class Essai
 {   public static void main (String [ ] args)
     {   System.out.println("levée de l'exception division par zéro") ;
         try
         {  int i = 1, j= 0 ;
            System.out.println("dès que la division par zéro est détectée, l'exception
                  correspondante sera levée et le traitement sera transféré à la partie
                  catch du programme") ;
            i = i/ j ;
         } catch (Exception e)
         {  System.out.println("Attention vous venez de faire une division par zéro !!") ;
            System.out.println("Votre premier traitement des exceptions est réussi") ;
            System.out.println("on peut aussi afficher l'exception e : " + e);
         }
     }
  }
```

Une instruction try peut comporter plusieurs catch correspondant à des exceptions différentes :

```
try {
        // Code pouvant lever des exceptions
}
catch (IOException d) {
        // capture et gestion des IOException et des sous-classes de IOException
        // levées dans la partie try
}
catch (Exception d){
        // Gestion de toutes les autres exceptions
}
```

Dans cet exemple lorsqu'une exception est levée dans la partie try, les catch sont traités dans l'ordre. Le premier qui capture l'exception levée dans le catch exécute son bloc : si une erreur d'entrées/sorties est levée dans la partie try du programme le premier catch capture cette exception puis l'exécute sinon s'il s'agit d'une autre exception celle-ci est capturée par le deuxième catch.

Exemple :

```
// Essai.java
class Essai
{    public static void main (String [ ] args)
     {   try
         {   int i = 1, j= 0 ;
             i = i/ j ;
             System.out.println("Ce message ne sera jamais affiché");
         }
          catch (NullPointerException e)
          {  System.out.println("l'application à besoin d'un objet non pas de null !!") ;
          }
          catch (Exception e)
          { System.out.println("Division par zero");
         }
     }
 }
```

9.3 Remontée des exceptions

Un objet exception est crée par le code à l'endroit ou l'erreur se produit. Il peut contenir toute l'information nécessaire à la description de la condition de l'exception, celui-ci est passé en argument pour le bloc de code qui doit traiter l'exception. L'objet Exception est envoyé d'un endroit du code throws et capturé à un autre endroit catch où l'exécution reprend.

L'appel à une méthode pouvant lever une exception doit :

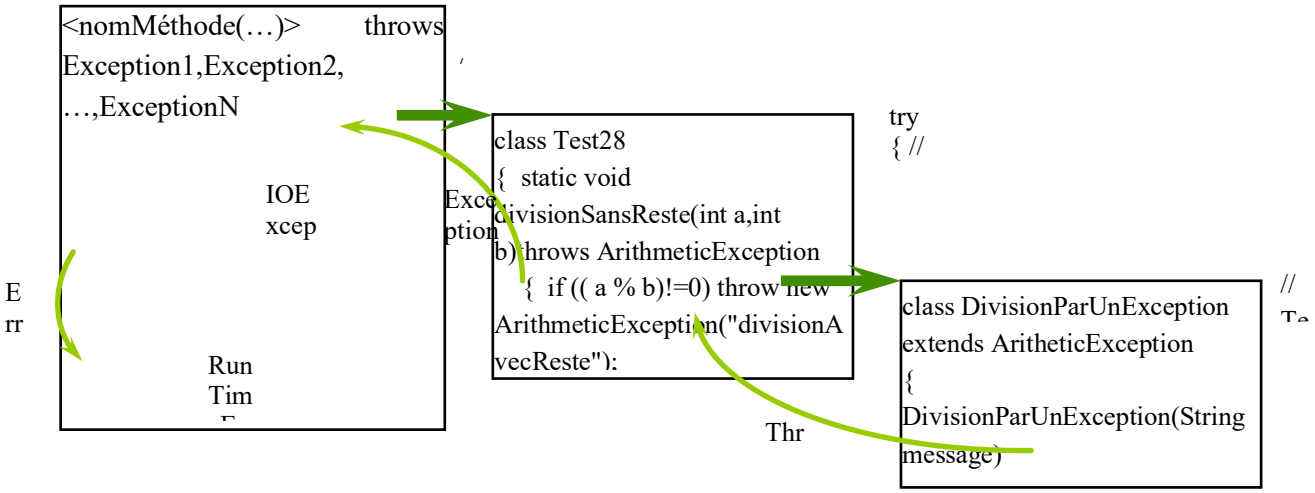
- ✓ être contenu dans un bloc try/catch

ou bien

- ✓ être situé dans une méthode propageant (throws)

L'exception levée est propagée à la méthode appelante. Lorsqu'il n'y a pas de bloc try/catch, l'exception remonte jusqu'au niveau le plus haut, ce qui provoque une erreur d'exécution.

Voyons ce processus sur un exemple :



Les deux méthodes openConnection() et sendRequest() lèvent les exceptions d'E/S (IOException). La méthode openConnection() se situant à l'intérieur d'un bloc try appelle la méthode sendRequest(). Cette dernière exécute la méthode write() qui génère une erreur d'E/S. A ce moment un objet de type IOException est crée (on dit que la méthode write() lève une exception IOException). L'exécution de write() est suspendue et l'exception est envoyée au niveau supérieur(méthode openConnection()). Cette méthode ne peut pas capturer l'exception, celle-ci est de nouveau renvoyé au niveau supérieur. Finalement, elle est capturée par le bloc try de la méthode getContent(), puis traitée dans le bloc catch.

Au niveau d'un bloc catch, il est possible de savoir l'endroit exact où l'exception s'est produite. Pour cela il suffit d'appeler la méthode printStackTrace() définit dans la classe Throwable qui indique toutes les méthodes qu'à traverser l'exception avant qu'elle ne soit captée.

Exemple :

```
// AffichageSourceException.java
class  AffichageSourceException
{  static int division(int a, int b)    throws Exception
   { return (a/b);
   }
   static float etapeSup(int a, int b) throws Exception
   { return (division(a,b)) ;
   }
   public static void main(String [ ] m)
   {  try
      {  int a=5,b=0;
         AffichageSourceException.etapeSup(a,b)) ;
      }   catch (Exception notreObjet)
      {   notreObjet.printStackTrace(System.err) ;
    }
}
```

9.4 Contrôle des exceptions

Lorsqu'on écrit les méthodes d'une classe, on peut contrôler les exceptions qui doivent être levées. Le mot clé throws indique au compilateur que cette méthode peut être la source d'une exception contrôlée.

Syntaxe :

```
Object getContent()
{
```

Tout utilisateur qui doit utiliser la méthode <nomMéthode()> doit impérativement traiter les exceptions qui suivent le throws.

Exemple :

Dans l'exemple AffichageSourceException.java, si vous enlevez le bloc try/catch alors le compilateur générera une erreur pour le traitement d'une exception non contrôlée. En effet, le bloc try/catch utilise la méthode etapeSup() qui impose le contrôle de l'exception Exception.

Remarque : Toutes les exceptions ne sont pas contrôlable. Les exceptions qui héritent des classes java.lang.RuntimeException et java.lang.Error ne peuvent pas être contrôlées.

9.5 Comment lancer une exception

Prenons un exemple :

Supposons que l'on décide d'écrire une méthode divisionSansReste(int a, int b) qui doit déclencher une exception de type ArithmeticException si le reste de la division de a par b est différent de zéro.
Pour lancer l'exception on ajoute l'instruction : throw new ArithmeticException ou bien les deux instructions suivantes : ArithmeticException a=new ArithmeticException ; throw a ;

Ce qui donne le résultat suivant :

```
Void openConnection()
   throws IOException
{
  openSocket();
  sendRequest();
  receiveResponse();
}
```

9.6 Création de classes exceptions

Il est également possible de créer des classes exceptions par dérivation de la classe Exception ou de l'une de ses filles.

Exemple : Interdire la division par 1

```
void sendRequest()
 throws IOException
{  write(header);
  write(body);
      // ERROR
}
```

Un bloc (optionnel) encadré par le mot clé finally peut-être posé à la suite des catch. Son contenu est exécuté après un catch ou après un return dans le bloc try qu'une exception ait été détectée ou pas. Ceci permet en particulier d'interrompre correctement le code, en libérant les ressources locales par exemple.

Exemple

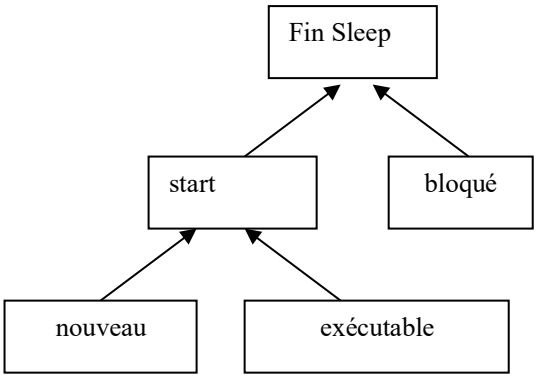
```
1
```

Le bloc finally peut être posé à la suite du bloc try si un bloc catch n'est pas prévu après le try. Qu'il est une exception ou pas, les instructions de la clause finally sont exécutées.

```
2
```

9.7 Compléments

Une exception est un objet d'une sous-classe de java.lang.Throwable. Celle-ci possède deux sous-classes standards Error et Exception. La classe Exception possède une sous-classe RunTimeException.



Les exceptions de la classe Error correspondent aux erreurs fatales. Elles ne sont pas capturées par les programmeurs, se sont des erreurs non récupérables. Elles provoquent l'arrêt de l'interpréteur java et permettent l'affichage d'un message d'erreurs.

Les exceptions de la classe Exception indiquent des erreurs non fatales pouvant être traitées par le programmeur. Une méthode successible de générer une exception doit nécessairement la capturer et la traiter ou bien prévenir dans sa signature avec le mot clé throws qu'elle ne la traite pas.

Les exceptions de la classe RunTimeException indiquent les erreurs non fatales que le programmeur peut capturer et traiter. Ce sont des erreurs trop courantes et ne nécessite pas de déclaration dans l'entête des méthodes qui ne les traitent pas. Ce sont des exceptions non contrôlées.

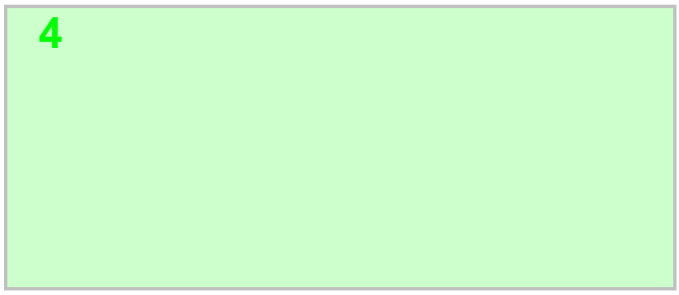
TD-TP n° : Les exceptions

Exercice 1 :

1. La classe Throwable est la superclasse des Exceptions et des erreurs en java. Elle dispose de 7 méthodes. Utilisez la documentation JDK pour découvrir ces méthodes.
2. La classe Exception hérite de la classe Throwable. Celle-ci est définie dans le package java.lang. la documentation de JDK 1.3 définit 23 classes filles de la classe Exception. Je vous propose de les lire au moins une fois.

Exercice 2 :

Soit le programme suivant :



1. L'exécution de ce petit programme déclenche une exception que je vous laisse le soin de découvrir.
2. proposer une solution try/catch qui permet d'afficher le message "Exception détectée" et de sortir correctement du programme.
3. (pour le tp) vous avez devant vous l'ensemble des classes dérivées de la classe Exception. Choisissez la classe adéquat pour notre exemple.

Exercice 3 :

considérez le programme suivant :

```
class Test26
{    public static void main(String [] arguments)
     {  float a=Float.parseFloat(arguments[0]);
        for (int i=1;i<=arguments.length; i++)
            a/=Float.parseFloat(arguments[i]);
        System.out.println("Le resultat est :" +a);
    }
}
```

La syntaxe de ce programme est correcte (la compilation ne décèle pas d'erreurs). Son exécution provoque à coup sûr au moins un type d'exception

1. Trouvez cette exception et tenter de la contrôler en afficher le message "exception !".

Le type d'arguments introduits au programme peut provoquer un autre type d'exception

2. Trouvez cette exception

Etant en présence de deux exceptions différentes, on peut vouloir les traiter différemment.

3. Trouvez l'ordre d'exécution du traitement de ces deux exceptions et afficher un message qui indique dans chaque cas le type d'exception.

Dans tous les cas le programme doit afficher le résultat qu'il a trouvé avant le déclenchement de l'exception.

La première version (td) peut contenir des noms d'exceptions qui ne correspondent pas aux noms donnés par java (exemple vous pouvez utiliser le nom ReferenceNonInitilisée pour indiquer qu'une référence à une classe n'est pas initialisée. Java appelle cette exception NullPointerException, elle hérite de la classe RunTimeException.

La deuxième version (tp) doit comporter les classes java appropriées.

Indications :

L'exécution de ce programme se fait comme suit :

```
java Test26
java Test26 12
java Test26 4 1 2
java Test26 4 0 8
```

Exercice 4 :

Ecrire un programme qui peut être invoqué de la façon suivante :

Java Exo4 taille int1 int2 int3 int4 … intN

taille, int1, … intN sont des entiers (int)

Le programme additionne les nombres int1 jusqu'à inttaille, divise le résultat par intN. Le paramètre taille indique le nombre de paramètres à additionner.

Utilisez les exceptions pour vous protéger des erreurs suivantes :

1. division par zéro
2. introduction d'un paramètre qui n'est pas un entier

Perfectionner votre programme en écrivant

1. une classe TailleIncompatibleException qui se déclenche si taille est différent de N-1
2. Une classe argsVide qui doit hériter de la classe Exception (indic. args est un tableau) et qui se déclenche si aucun argument n'est introduit lors de l'exécution
3. Une exception qui se déclenche si le résultat final n'est pas un entier.

10 Les Threads

10.1 Introduction

Les threads donnent la possibilité aux programmeurs de pouvoir manipuler plusieurs taches à la fois. Ceci vous permet, par exemple, d'afficher plusieurs plateaux de jeux d'échecs sur votre page web et de pouvoir jouer simultanément sur tous les plateaux, chaque plateau pouvant être géré par un thread.
Un thread peut être considéré comme un processus. Les threads d'une même application partagent le même espace d'adressage : ils peuvent utiliser leurs propres variables locales mais partagent les mêmes variables d'instances).

Prenons un exemple pour mieux illustrer ce concept : vous décidez d'aller avec des ami(e)s au restaurant, chacun d'entre vous a envie d'un menu particulier mais vous partagez le même espace : le restaurant, la carafe d'eau, le pain, la bouteille, etc. mais chacun possède un couteau et une fourchette. Chacun d'entre vous peut être vu comme un thread.
Tant que vous mangez dans votre coin, tout ce passe bien. Mais si vous décidez de vous servir un verre de vin un problème se pose : les autres peuvent vouloir la même chose que vous. Pour résoudre le problème d'accès aux ressources communes vous devez vous synchroniser. Les threads procèdent de la même façon : ils utilisent la synchronisation.

Revenons à notre exemple, si vous êtes galant vous commencerez par servir les dames ; vous venez d'établir des priorités. Un thread dispose d'une priorité pour accéder aux ressources communes. Il peut par exemple se réserver le droit d'utiliser un objet jusqu'à ce qu'il ait terminé sa tâche.

10.2 La classe Thread

La classe java.lang.Thread permet la gestion de plusieurs tâches parallèles en Java. Un objet de type thread est crée par java pour gérer et contrôler tout processus indépendant.

Conceptuellement, un thread a besoin d'une adresse (un pointeur) lui permettant de localiser les fonctions que doit exécuter le processus qu'il gère. Comme les pointeurs n'existent pas en java, une interface est créée pour permettre de résoudre ce problème, il s'agit de l'interface *Runnable*. Runnable est une interface qui contient une seule fonction (la méthode *run()*) qui doit être surchargée par toute classe désireuse d'utiliser un thread.

```
L'interface Runnable
Public interface Runnable
{   public void run();
}
```

10.2.1 Comment se déroule l'exécution d'un thread ?

La création d'un objet Thread se en invoquant un des constructeurs de la classe Thread. La création de l'objet n'implique pas l'exécution du Thread. Un Thread ne s'exécute que lorsque sa méthode *start()* soit explicitement exécutée (la méthode start() ne s'exécute qu'une seule fois durant toute la vie d'un thread). L'exécution de la méthode start() entraîne le déclenchement de la méthode run() de l'objet cible.

Un objet cible est une instance d'une classe qui implémente l'interface Runnable et qui implémente la méthode *public void run()*.

De manière générale, toute classe contenant une méthode public void run() peut implémenter l'interface Runnable et devenir cible de la classe Thread.

NB : s'il n'est pas souhaité de créer une méthode run() dans une classe, il est toujours possible de créer un adapteur (voir la déclaration plus loin) qui servira de Runnable.

10.2.2 Quelques méthodes

Thread () : construit un nouvel thread.

void run(). Elle doit être surchargée et il faut y ajouter le code qui doit être exécuté dans le thread.

void start() lance le thread et appel la méthode run()

static void sleep (long millis) : place le thread en cours d'exécution en veille.

Exemple 1:

```
import java.lang.*;
public class Test32 implements Runnable
{   boolean dejaok;
    int f;
    public Test32()
    {    dejaok = false;
         f=0;
    }
    void affiche()
    {   if (!dejaok) {
        System.out.println("message1");
        dejaok=true;
     }
 }
 public void run()
 {  int x;
    while (true)
     affiche();
  }
   public static void main(String [] args)
   {   Test32 test = new Test32();
       Thread leThread = new Thread(test);
        leThread.start();
    }
}
```

Exemple 2 :

```
import java.lang.*;
import java.io.*;
public class Test32 implements Runnable
{  boolean dejaok;
   int f;
   public Test32()
   {   dejaok = false;
       f=0;
    }
    public void run()
    {   if (f==0)
        {  f++;
           while (true)
             System.out.println("la valeur de f est = " +0);
        }
        else
        while (true)
            System.out.println("la valeur de f est = " +f);
      }
      public static void main(String [] args)
      {  Test32 test = new Test32();
         Thread leThread = new Thread(test);
         leThread.start();
         Thread leThread2 = new Thread(test);
         leThread2.start();
      }
}
```

Il existe plusieurs façon d'utiliser les threads. La première est celle exhibée dans les deux exemples précédents, mais cette approche ne respecte pas parfaitement le paradigme de la programmation orientée objet. Pour respecter cette condition on peut utiliser les deux méthodes suivantes :

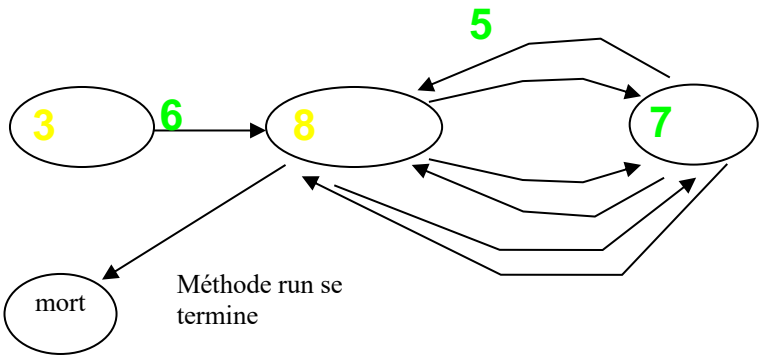
1. Définir un Thread comme une variable de la classe

```
class Test implements Runnable
{     Thread x ;
      Test ()
      {   x = new Thread(this);
          x.start();
       }
 }
            …
```

1. utilisation d'un adaptateur :

```
class Test
 {  maMéthode ()
    {   Thread x = new Thread  (new Runnable()
           { public void run(){UneMéthodeAExecuter();});
             x.start();
           }
         private void uneMéthodeAExecuter(){ …}
            …
    }
}
```

10.3 Le diagramme d'états d'un thread



10.4 Synchronisation

Quand plusieurs Threads accèdent aux mêmes objets, il est indispensable de contrôler l'accès à ces objets afin de maîtriser les mises à jours qui sont faites dessus.

Le langage java utilise un mécanisme simple de contrôle d'accès. Celui-ci s'inspire des moniteurs. Ceci se fait en marquant l'opération qu'on veut ne pas interrompre par le mot clé *synchronized*.

Exemple : public synchronized void test () { …}
L'exécution de cette opération ne peut être exécutée que par un thread à la fois. Tous ceux qui la demande pendant qu'elle s'exécute, seront mis en attente.

TD-TP n° 5 : Les Threads

Exercice 1 :

Etudier la classe Thread

Exercice 2 :

Trois individus (LeChat, LaPoule et LeChien) se mettent autour d'une table pour déjeuner. Ils peuvent faire trois activités : manger, discuter et se servir de l'eau. Les individus mangent en silence durant des phases qui excèdent 5 secondes après avoir déclarer qu'ils sont en train de manger. Un individu qui veut se servir de l'eau indique qu'il veut boire et monopolise la bouteille pendant 5 secondes (si l'eau est fraîche, il informe les autres). Ecrire un programme qui permet de reproduire cette situation en utilisant les threads.

Exercice 3 :

Votre employeur verse votre salaire sur votre compte régulièrement (toutes les secondes pour notre programme). Vous utilisez à tout moment ce même compte pour faire vos achats (une valeur prise au hasard entre 100 et 1000 milis).

a. Sans vous préoccupez de la synchronisation, écrire un programme qui permet de simuler cette situation en respectant les indications suivantes :
    i. initialement le contenu de votre compte est 0
    ii. à la fin de l'opération vous devez afficher le total de vos achats, le solde de votre compte. Un message est affiché indiquant si une erreur calcul s'est produite pendant l'opération.
b. Utiliser la synchronisation et vérifier si des erreurs de calcul se produisent toujours.

Solution Exercice 2

```
import java.math.*;
import java.lang.*;
import java.io.*;
public class Restaurant implements Runnable {
//  static boolean platLibre ;
  static boolean bouteilleLibre;
 String nom;

  public Restaurant(String n) {
 // platLibre = true ;
  nom = n;
  bouteilleLibre = true ;

  Thread a = new Thread (this);
    a.start();
  }
  public void run(){
  int h= 0;
  while (h <20)
  { h++;
  double x = Math.random();
  if (x < 0.3)
  { System.out.println(nom+":  Je mange");
    try
    { Thread.sleep(2000);
    }catch (InterruptedException e){}
  } else
  if (x <0.6)
  { System.out.println(nom+ ":  Je discute ...");
   } else
   if (x<0.9) { System.out.println (nom+":  je veux
de l'eau : ");
     if (bouteilleLibre)
     { try
     { bouteilleLibre =false;
       Thread.sleep(5000);
       bouteilleLibre=true;
     }catch (InterruptedException e){}
     if (Math.random()<0.5)
System.out.println(nom+": Cette eau est fraiche
") ;
     }
    } else
    {  System.out.println (nom+":  je suis fatigué, je
vais me reposer ");
       try{Thread.sleep(10000);}catch
(InterruptedException e){}
       System.out.println (nom+":  je suis de retour
!!! ");
    }
    }
   }
   public static void main(String a[])
   { Restaurant a1= new Restaurant ("Le chat");
     Restaurant a2= new Restaurant ("La poule");
     Restaurant a3= new Restaurant ("Le chien");
   }
}
```