Brève introduction au langage de programmation Java

Java est un langage de programmation orienté objet développé par SUN Microsystems.
Bien que similaire à C++, il existe des différences fondamentales entre les deux langages.

Principales distinctions avec le C/C++ ...

- Il existe une classe Object dont dérivent implicitement toutes les autres classes.
- Seul l'héritage simple est implémenté (il n'y a pas d'héritage multiple). Par contre, il est possible d'indiquer qu'une classe implémente plusieurs **interfaces**.
- Plus de struct et d'union. La structure de base c'est tout simplement la classe.
- La notion de *package* existe, un package peut contenir des classes ou d'autres packages.
- Les pointeurs ne sont plus manipulés directement.
- Le *respect des types* est systématiquement vérifié à la compilation. En règle générale, il n'y a pas de transtypage implicite.
- Les fichiers .h n'existent plus, remplacés par le mécanisme d'importation de classes, intégrées dans un package.
- La documentation d'un programme peut être générée automatiquement à partir des commentaires inclus dans le source, avec la commande javadoc.
- Tout le code source correspondant à une classe se trouve dans un seul et même fichier NomClasse.java.
- Une classe peut être publique ou protégée (accessible uniquement à partir de son package).
- Chaque méthode **doit** déclarer la liste des exceptions qu'elle peut "soulever" et le compilateur vérifie leurs présences.
- Il existe une bibliothèque très complète et standardisée disponible chez SUN, fournissant une série de paquetages de classes qu'il faut connaître (pour gérer notamment les entrées-sorties, les interfaces graphiques awt et swing, les connexions réseaux, les applets ...).
- Java impose la gestion des exceptions

Syntaxe Java en bref

Visibilité en java

	private	***pas de spec***	**protected**	**public**
même classe	**OUI**	**OUI**	**OUI**	**OUI**
sous-classe même package	**NON**	**OUI**	**OUI**	**OUI**
pas une sous-classe, même package	**NON**	**OUI**	**OUI**	**OUI**
sous-classe, packages différents	**NON**	**NON**	**OUI**	**OUI**
pas sous-classe, packages différents	**NON**	**NON**	**NON**	**OUI**

Source
http://users.polytech.unice.fr/~dedale/cours/langages-programmation/java/generalites/visibilite.html

Les identificateurs

L'identificateur en JAVA se compose d'une suite sans espace de caractère alphanumériques, _ et $. Le premier caractère doit être soit une lettre, soit _, soit le $. Java est sensible à la casse.
NB: Un identificateur ne peut appartenir à la liste des mots réservés du langage java.

Déclaration d'une classe

```
[visibilité] class NomDeLaClasse {
        [declaration d'attributs]
        [declaration de méthodes]


  [ Public static void main(String args[])
        {
        bloc d'instructions
        } ]
}
```

L'instruction ***main*** déclarée comme une fonction ou une procédure correspond au programme principal, c'est à dire celui qui contient les instructions correspondant aux actions que l'on doit effectuer. Il est exécuté au travers de la classe dans laquelle il est déclaré.

Déclaration d'un attribut

```
[visibilité] typeAttribut nomAttribut [=valeur initiale] ;
private static int counter=0 ;
private Person p ;
```

Déclaration d'un constructeur

Constructeurs

```
[visibilité] NomClasse (paramètres){
        // instructions
}
```

Déclaration d'une méthode

```
 [visibilité] typeRetourné nomMethode(paramètres) {
        // instructions
}
```

Toutes les méthodes en java ne peuvent être définies qu'en tant que membres d'une classe

Déclaration d'accesseurs

Il est bon en java pour respecter le principe d'encapsulation des attributs de déclarer les attributs privés et d'utiliser des méthodes appelées accesseurs pour lire ou modifier ses attributs.

Méthode pour lire un attribut :

```
public typeAttribut getNomAttribut(){
        return this.attribut ;
}
```

Méthode pour modifier un attribut :

```
public void setNomAttribut(typeAttribut var){
        this.attribut=var ;
}
```

Surcharge des méthodes.

La surcharge des méthodes est possible en java et permet de définir des méthodes possédant le même nom mais des paramètres différents en termes de types ou nombre de paramètres.

Instanciation (création d'objets)

Déclaration d'un objet : Nomclasse nomObjet;
Création d'un objet : nomObjet = new NomClasse(paramètres);// NomClasse (paramètres) correspond à un constructeur de la classe

Déclaration et création en une seule opération : NomClasse nomObjet = new NomClasse(arguments);
Exemples : Point p1 ; Point p2 = new Point (); Point p3 = new Point ('A', 7, (float)-0.5);

Affectation et comparaison d'objets

Affecter un objet à un autre (objet2 = objet1) rend les 2 objets identiques; ils ont la même référence. Toute modification de objet1 est visible sur objet2 et vice et versa.
Deux objets peuvent avoir les mêmes valeurs d'attributs sans être identiques: ils ont des références différentes.
On peut recopier les attributs d'un objet dans un autre en utilisant la méthode clone(). Cette méthode définie dans la classe Object peut être redéfinie dans toute autre classe pour fournir cette fonction.
Dans ce cas les objets sont distincts et une modification de l'un n'impacte pas l'autre.
Exemple : Point p4 = p2.clone().

Comparer deux objets

Si o1 et o2 étant deux objets, o1==o2 est vrai si a et b sont identiques. Cette opération ne

compare pas les valeurs des attributs mais les références des objets.
Pour comparer les valeurs des objets, on utilise la méthode equals(objet) définie dans la classe Object et qui doit être re-implémentée toute classe qui veut fournir cette fonction.

Types primitifs
short, int, long, float, double, boolean

Déclaration de variables:
NomType nomvar; ou nomType nomvar1, nomvar2,...;

Instructions conditionnelles
Si alors sinon
```
if (condition ) {
        [instructions]
}else {
        [instructions]
}
```

selon que … faire
```
switch ident {
        case val1 : {[instructions]}

        case val2 : {[instructions]}
.
.
        default : {[instructions]}
}
```
NB : Typiquement, l'instruction break sera appelée à la fin de chaque bloc d'instructions pour un case.

Instructions itératives
Pour
```
for (initialisation; condition; modificateur) {
        [instructions]
}
```
exemple:
```
int c;
for ( c=0; c<=10; c++) {
        System.out.println ( c );
}
```

Tant que
```
while ( condition ) {
```

```
        [instructions]
}
```

Faire jusqu'à
```
do {
        [instructions]
}while (condition)
```

Les débranchements

break est une instruction qui permet de quitter immédiatement une boucle ou un branchement.
continue s'utilise dans une boucle pour passer directement à l'itération suivante.

Chaines de caractères et tableaux

Initialisation d'une chaines de caractère : String nomchaine = " valeur ";
Voir la classe java.lang.String pour les méthodes de manipulation de chaines de caractères.

En Java, les tableaux sont considérés comme des objets. Ils sont dérivés de la classe Object.
Déclaration: TypeElt nomTab[] ou TypeElt [] nomTab ou TypeElt nomTAb[] = new TypeElt [20];
Initialisation: un tableau peut être initialisé lors de sa création
int tab1[5]= {1,2,5,6,-3} ou int tab1[]= {1,2,,5,6,-3};

Accès à un element du'un tableau: nomTab [indiceElt]
NB.: comme en C et C++, les indices des tableaux en java commencent par 0.

Tableau à plusieurs dimensions
TypeElements tab [][]; ou TypeElements tab [][] = new TypeElements [][]; ou
TypeElements tab [][]=TypeElements tab [nb1][nb2];

Les commentaires

Les commentaires en java sont de trois types:
1. Commentaires abrégés ou sur une ligne // commentaire
2. Commentaires sur plusieurs lignes

```
/* commentaire .....
.....*/
```

3. Commentaire de documentation automatique: ce type de commentaire va permettre de générer automatiquement une documentation pour votre programme. Il respecte une certaine syntaxe.

```
/**
        @ paramètres
        @ return
*/
```

Principales conventions d'écriture en java

- Une classe définie *public* doit être placée dans un fichier de même nom (obligatoire).
- L'identificateur d'une classe ou d'interface doit commencer par une majuscule. Il est fortement recommandé de recapitaliser à chaque nouveau mot. (ex : MaClasse plutôt que Maclasse)
- L'identificateur d'un attribut ou d'une méthode doit commencer par une minuscule on recapitalise à chaque nouveau mot. (ex : nomEtudiant, execute)
- L'identificateur d'une constante s'écrit en majuscule et on sépare chaque mot du suivant par un underscore (ex : RED_ALERT_CODE)

Héritage en java

Pour qu'une classe hérite d'une autre, on la déclare à l'aide de la syntaxe suivante:

```
[visibilité] ClasseFille extends ClasseMere{
        [declaration d'attributs]
        [declaration de méthodes]

        [ Public static void main(String args[]){
                        bloc d'instructions
                } ]
}
```

Exemple: Etudiant extends Personne

L'héritage multiple n'est pas permis en Java. Une classe ne peut hériter que d'une seule classe.
Les héritages successifs permettent de créer une hiérarchie de classes. En java la classe Object est la racine de la hiérarchie des classes. Toute classe hérite par défaut (de manière implicite) de la classe **Object** sauf si un lien d'héritage explicite est indiqué lors de sa déclaration. (cf. la doc de java au niveau de la classe Object pour connaitre les méthodes dont hérite toute classe de java.)
Pour simuler l'héritage multiple, on utilise les interfaces.

Interfaces

Une interface est une collection de déclarations de méthodes dépourvues d'implémentation. Une interface se déclare à l'aide la syntaxe :

```
[visibilité] interface NomInterface [extends InterfaceMere]{
        [declaration d'attributs]
        [declaration de méthodes sans corps]
        //redefinir toutes les méthodes de toutes les interfaces implémentées
}
```

Lorsqu'une classe implémente une interface elle doit alors redéfinir toutes les méthodes de l'interface.

Une classe peut implémenter plusieurs interfaces à l'aide la syntaxe :

[visibilité] **class** NomDeLaClasse [extends NomClasseMere] implements interface 1[,interface2, …, interface n]{
    [declaration d'attributs]
    [declaration de méthodes]
//redefinir toutes les méthodes de toutes les interfaces implémentées
}

Exemple d'interface : java.util.List
Exemple de classe implémentant cette interface : java.util.ArrayList

Classes et méthodes abstraites

Une classe abstraite est une classe qui ne peut être instanciée. Elle doit donc avoir obligatoirement des classes filles non abstraites pour être utilisées. Une classe abstraite se déclare ainsi :
[visibilité] abstract **class** NomDeLaClasse {
    [declaration d'attributs]
    [declaration de méthodes]
}

Une méthode abstraite est une méthode dont l'entête seulement est fourni :
[visibilité] typeRetourné nomMethode(paramètres) ;

Une classe est abstraite dès qu'elle contient une méthode abstraite, elle doit alors être déclarée **abstract**.
Toutes les méthodes abstraites d'une classe abstraite doivent être redéfinies dans toutes les sous classes non abstraites de cette classe.
Les classes abstraites sont très utiles pour définir des méthodes dépendant d'autres méthodes qui ne sont pas précisées.

Gestion des exceptions

Une *exception* est une interruption de l'exécution d'un programme suite à une erreur. Par exemple, une division par zéro provoque une exception de type ArithmeticException. Java permet non seulement la gestion des exceptions, mais aussi la création d'exceptions spécifiques.
La gestion des exceptions se fait à l'aide des blocs d'instructions try et catch. La syntaxe est la suivante :
try {

```
/       / zone contenant des instructions pouvant lever des exceptions
} catch (NomException e) {
        // Traitement de l'exception capturée. Une exception ets un objet Java. Cf la classe
java.lang.Exception
}
```
On peut donc définir autant de bloc **catch()** qu'il y a d'exceptions susceptibles d'être levées.

Lever une exception

Pour qu'une exception puisse être capturée il faut qu'elle soit levée. Pour lever une exception, on utilise le mot clé **throw.**

Plusieurs classes d'exception existent déjà en java mais on peut définir nos propres classes d'exception à l'aide de la syntaxe ci-dessous.

```
[visibilité] class NouvelleException extends Exception{
NouvelleException () {}
        NouvelleException (String msg) {
        super(msg);
}
```

Pour pouvoir mettre le mécanisme de gestion d'exception en place, il faut alors lever l'exception dans toute fonction qui entraine cette exception et ensuite dans les méthodes faisant référence à cette méthode, la traiter en utilisant les blocs try et catch.

Exercice 1 (tiré du site http://www.siteduzero.com/)
Créer un nouveau Projet Inf2430TP1 dans Eclipse (File→New→JavaProject)

Créer dans ce projet un nouveau package, ne pas utiliser le package par défaut ; regarder les conventions de nommage des packages (lien= http://www.siteduzero.com/forum/sujet/conventions-de-codage-en-java-64171)

Créer la classe MyMain qui va contenir un main où vous pourrez déclarer les variables et faire vos traitements.

Créer la classe Ville
Quels peuvent être les attributs et les méthodes de cette classe? Implémentez les dans la classe. Combien de constructeurs prévoyez vous ?

Créez dans le main des objets de type ville et exécutez le programme.

Exercice 2 :
Créer dans le projet la classe Capitale qui hérite de la classe Ville
Quels sont les attributs et méthodes propres à cette classe ?

Testez la nouvelle classe dans votre main.

Exercice 3
Écrire la classe Arithmetic avec la méthode double divide(a,b) qui divise a/b et appeler cette méthode depuis le main avec b=0. Observez ce qui se produit.
Rajoutez les instructions try et catch afin de traiter cette exception. Ré-exécuter le programme.
Utilisez la méthode **printStackTrace()** de l'objet exception pour afficher le message par défaut de l'exception.
Il existe un autre mot clé dans la gestion des exceptions qui définit un bloc dont les instructions sont systématiquement exécutées, qu'une exception soit levée ou non. C'est le mot clé finally.
Rajouter donc à votre programme le bloc finally après le bloc catch et affichez juste la phrase « fin du programme ».

```
finally {
        System.out.println("Fin du programme");
}
```

Exécutez le programme avec b =0 et b=5. Observez ce qui se passe.

Exercice 4
Créer la classe NombreHabitantException qui lèvera une exception si le nombre d'habitants passé en paramètre est négatif.

Modifiez le constructeur de la classe Ville pour lever l'exception

On est maintenant obligé de rajouter un bloc try…catch dans le main afin de gérer la capture éventuelle de cette exception.
Créer un objet ville avec un nombre négatif d'habitants ; exécutez le programme et décrivez ce qui se passe.

Références et liens

Cours java Isig
http://www.siteduzero.com/informatique/tutoriels/apprenez-a-programmer-en-java
http://www.eclipsetotale.com/articles/premierPas.html (Premiers pas avec Eclipse pour voir comment créer un projet et se familiariser avec l'environnement)
http://www.jmdoudoux.fr/accueil_java.htm
http://docs.oracle.com/javase/7/docs/api/overview-summary.html