ECOLE NATIONALE SUPERIEURE POLYTECHNIQUE

Département des Génies Électrique et des Télécommunications

Support de cours

BASES DE DONNEES ET PROGRAMMATION Java/C++

Réalisé par

Dr Narcisse Talla

ntalla@gmail.com

2012-2013

SOMMAIRE

CHAPITRE 1 LE SYSTEME D'INFORMATION

1.1 Quelques définitions

- Une information est le résultat d'un processus de mise en forme et de matérialisation visant à communiquer un fait ou un ensemble de faits à un public donné.

- Un système est un ensemble d'éléments en interaction dynamique organisés en fonction d'un but.

- Un système **ouvert** est en relation permanente avec son environnement : c'est un **réservoir** qui se remplit (énergie, matière, informations) et se vide (entropie, énergie usée) à la même vitesse pour se maintenir dans un état donné. Il est en constante interaction avec son environnement, le modifiant et en retour, se trouvant modifié.

- Un système **fermé** n'échange ni matière, ni informations, ni énergie : il vit sur son énergie interne et au fur et à mesure des réactions, accroît son entropie qui devient maximale.

- Un système **complexe** est constitué d'une grande variété de composants ou d'éléments possédant des fonctions spécialisées :
  - éléments organisés en **niveaux hiérarchiques**,
  - éléments et niveaux reliés par multiples **liaisons** ,
  - interactions **non linéaires**,
  - comportement difficilement **imprévisible** et **grande résistance** au changement.

Tout système présente deux aspects :

- L'aspect structural (description structurelle ou spatiale).
- L'aspect fonctionnel (description temporelle).

1.2 La notion de "système d'information".

Le concept de système d'information est un produit de la théorie générale des systèmes, mise sur pied dans les années 1940 par Van Bertalanffy et développée dans les années 1950, notamment, en France, par Jean-Louis Le Moigne. Ce qui caractérise l'approche systémique peut être considéré comme une réaction à une approche analytique typique de la logique cartésienne et on notera que la systémique insiste sur la notion de totalité.

Par exemple pour Saussure, un système est une *"totalité organisée, faite d'éléments solidaires ne pouvant être définis que les uns par rapport aux autres en fonction de leur place dans cette totalité."* Pour Von Bertalanffy il s'agit d'un *"ensemble d'unités en interrelations mutuelles" ou encore pour Lesourne d'un "ensemble d'éléments liés par un ensemble de relations"*.

La systémique distingue des systèmes et des sous-systèmes. Par exemple dans un véhicule automobile, le moteur sera un sous-système. Dans le corps humain, l'appareil digestif sera un sous-système.

1.2.1 La systémique

Pour scruter l'infiniment petit et l'infiniment grand, les scientifiques disposent d'outils tels le microscope et le télescope qui ont permis des bonds importants de la connaissance. Aujourd'hui, on est confronté à **l'infiniment complexe** ; on ne dispose pas d'outils pour étudier cette formidable complexité des systèmes dont nous sommes les éléments, les particules.

L'approche systémique a pour but de développer un **instrument symbolique** que Joël de Rosnay appelle le **macroscope** pour étudier cette complexité. Toutefois, l'approche systémique n'a pas la prétention de tout expliquer, de tout résoudre et n'a pas pour but de présenter des modèles du monde qui prétendent tout englober.

La tendance naturelle de l'esprit humain est unifiant, réducteur, rapprochant ; cela donne des modèles apparemment satisfaisants mais dangereux : on filtre, on élimine tout ce qui est différent du modèle unifié et on arrive aux pires intransigeances. Les modèles systémiques sont des points de départ de la réflexion ; ils doivent être confrontés à la réalité, agressés, détruits pour être mieux reconstruits car ils ne **peuvent évoluer** que dans la **confrontation**.

1.2.2 Quelques méthodes d'analyse

En analyse informatique, on distingue plusieurs méthodes parmi lesquelles :

1.2.2.1 La méthode FAST (Function Analysis Systems Techniques)

Principe général : Instrument graphique de communication entre les intervenants, FAST porte sur les relations entre produits et fonctions. Cet instrument permet de représenter la logique des relations entre les fonctions par répétition de « *pourquoi/comment/quand* » posés à chaque étape de l'analyse. FAST présente les relations existant entre produits et fonctions à l'intérieur d'un domaine strictement délimité.

Originalités : FAST produit un diagramme qui permet au concepteur d'expliquer et de justifier les solutions techniques.

1.2.2.2 La méthode APTE (APplication des Techniques d'Entreprise)

Principe général : Ensemble d'outils méthodologiques mis à disposition des équipes de projets pour maîtriser la cohérence entre les différents éléments d'un projet, faire les choix technologiques performants et fiables pour satisfaire les besoins au moindre coût. Cette méthode propose des logiques de formalisation communes pour les phases conceptuelles de projets complexes.

Originalités : Très systémique dans le sens où le produit est conçu à partir de son milieu environnant en privilégiant le point de vue du concepteur et de l'utilisateur, la méthode APTE comporte ses propres outils internes de **validation** (besoin, fonctions, contraintes, ...). APTE propose une méthode d'animation de groupe efficace.

1.2.2.3 La méthode GRAFCET

Principe général : GRAFCET propose une représentation graphique des comportements attendus d'un système logique. « Faire un GRAFCET » consiste à décrire un fonctionnement séquentiel à partir des spécifications fonctionnelles d'un équipement automatisé. Un GRAFCET permet de décrire la partie « commandes » d'un système automatisé.

Originalités : GRAFCET s'adresse uniquement aux automatismes. Le langage GRAFCET est employé dans la programmation d'automates programmables. GRAFCET, est normalisée par norme internationale (CEI 848 de Décembre 1988).

1.2.2.4 La Méthode RELIASEP

Principe général : A partir du besoin exprimé en terme de fonctions, rechercher les sous-fonctions nécessaires à la satisfaction de ce besoin (arbre fonctionnel). Cette recherche est progressive et adaptée aux phases de développement.

Originalités : Très orientée Sûreté de Fonctionnement, son arbre fonctionnel sert de base à l'Analyse des Modes de Défaillance, de leurs Effets et leur Criticité (AMDEC). Cette méthode permet de visualiser le cheminement fonctionnel des dégradations.

RELIASEP se révèle efficace pour l'étude de systèmes complexes plutôt « matériels » que « logiciels ».

1.2.2.5 Méthode SA/RT

Principe général : A partir d'une analyse fondée sur des interrogations pertinentes et de la représentation graphique de cette analyse, cette méthode permet l'élaboration :

- d'un modèle du système analysé,
- du modèle du processus statique qui lui est attaché,
- du modèle de contrôle dynamique qui en permettra l'utilisation.

Originalités : Cette méthode prend en compte l'aspect dynamique du système analysé. Elle est fondée sur une méthode plus ancienne (Structured Analysis -SA) qui a été largement utilisée.

- SA/RT est bien adaptée aux applications à fort comportement dynamique.
- SA/RT est plutôt orientée partage du travail et interface homme/machine.

1.2.2.6 La méthode SADT

Définition : L'acronyme S.A.D.T signifie : **S**ystem **A**nalysis and **D**esign **T**echnic. Cette méthode a été mise au point par la société Softech aux Etats Unis. La méthode SADT est une méthode d'analyse par niveaux successifs d'approche descriptive d'un ensemble quel qu'il soit. On peut appliquer le SADT à la gestion d'une entreprise tout comme à un système automatisé.

Principe général : Modélise des systèmes existants ou futurs pour en comprendre le fonctionnement et envisager des solutions. Cette modélisation porte sur les actions du système analysé (actigrammes), et sur les données que ce système doit traiter (datagrammes) dans une structure arborescente de ce système.

Originalités : Plutôt orientée systèmes d'information, logiciels et automatismes, SADT met en jeu deux types d'acteurs : un *« auteur »* qui conçoit le système et un *« lecteur »* qui le « critique ». Ces travaux permettent de préciser et d'optimiser le système par approche itérative.

SADT permet une bonne traçabilité et des contrôles internes de cohérence. Il peut se résumer en les points suivants :

- C'est un outil graphique de représentation.
- Il oblige à consigner par écrit les décisions d'une équipe de travail. Ceci permet progressivement de créer une documentation complète.
- C'est un travail d'équipe qui demande discipline et coordination. En effet, Le SADT est un produit pour communiquer et pour être diffusé.
- Son formalisme conduit à une représentation structurée ascendante ou descendante.

- Si le SADT est utilisé complètement (Actigrammes et Datagrammes) il permet de programmer directement un système automatisé. Une MOCN (Machine Outil à Commande Numérique) peut être programmée directement en SADT.

1.2.2.7 La méthode MERISE

Principe général : Représentation d'un système existant par les éléments invariants les plus simples (sous-systèmes de rang n) qui le composent, optimisation des liens et relations entre ces éléments invariants puis reconstruction d'un nouveau système sur les seuls éléments, liens et relations nécessaires au fonctionnement du système.

Originalités : MERISE est très orientée systèmes d'information et logiciels. Plutôt ancienne, elle s'est progressivement enrichie de particularismes au fur et à mesure de son utilisation, jusqu'à donner naissance à des « clones de MERISE » propres aux entreprises utilisatrices.

CHAPITRE 2 MODELISATION D'UN SYSTEME D'INFORMATION PAR MERISE

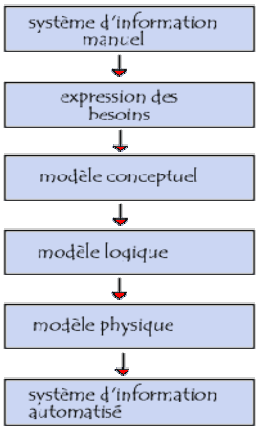
2.1 Introduction

La conception d'un système d'information n'est pas évidente car il faut réfléchir à l'ensemble de l'organisation que l'on doit mettre en place. La phase de conception nécessite des méthodes permettant de mettre en place un modèle sur lequel on va s'appuyer. La modélisation consiste à créer une représentation virtuelle d'une réalité de telle façon à faire ressortir les points auxquels on s'intéresse.

2.2 Cycle d'abstraction de conception des systèmes d'information

La conception du système d'information se fait par étapes, afin d'aboutir à un système d'information fonctionnel reflétant une réalité physique. Il s'agit donc de valider une à une chacune des étapes en prenant en compte les résultats de la phase précédente. D'autre part, les données étant séparées des traitements, il faut vérifier la concordance entre données et traitements afin de vérifier que toutes les données nécessaires aux traitements sont présentes et qu'il n'y a pas de données superflues.

Cette succession d'étapes est appelée *cycle d'abstraction* pour la conception des systèmes d'information :



2.2.1 Système d'Information manuel

Le système d'information manuel est un ensemble d'informations manuscrites et orales utilisées pour la gestion d'une structure. Ces informations se présentent généralement sous forme de fiches

à remplir ou sous forme de rapports à interpréter. Dans le processus d'analyse, cette étape consiste à réaliser des interviews auprès des différents responsables de service ou gestionnaires du système et à collecter les échantillons des différents types d'information manipulée dans le système.

2.2.2 Expression des besoins

L'expression des besoins est une étape consistant à définir ce que l'on attend du système d'information automatisé, il faut pour cela :

- faire l'inventaire des éléments nécessaires au système d'information,
- s'en quérir des limites ou des difficultés de gestion du système,
- délimiter le système en s'informant auprès des futurs utilisateurs.

2.2.3 Modèles conceptuels

Les modèles conceptuels ici concernent l'aspect organisationnel et statique du système. Il est gouverné par un modèle sur lequel repose la base de données du système : le modèle conceptuel de données (MCD). Ce modèle décrit l'architecture de la base de données et présente les règles et les contraintes à prendre en compte.

2.2.4 Modèle logique

Le modèle logique représente un choix logiciel pour le système d'information. Il est déduit du MCD et permet d'écrire des requêtes d'interrogation de la base de données. Il présente la base de données sous forme relationnelle et exprime les dépendances fonctionnelles associées.

2.2.5 Modèle physique

Le modèle physique reflète un choix matériel (par exemple Ms Access, Oracle, Paradox, Mysql, etc.) pour le système d'information. Il présente l'organisation des tables crées et les différentes relations entre elles.

2.3 Modèle conceptuel de données (MCD)

2.3.1 Le but

Le modèle conceptuel des données (MCD) a pour but d'écrire de façon formelle les données qui seront utilisées par le système d'information. Il s'agit donc d'une représentation des données, facilement compréhensible, permettant de décrire le système d'information à l'aide d'entités.

2.3.2 Entités et classe d'entités

Une *entité* est la représentation d'un élément matériel ou immatériel ayant un rôle dans le système que l'on désire décrire.

On appelle *classe d'entité* un ensemble composé d'entités de même type, c'est-à-dire dont la définition est la même. Le classement des entités au sein d'une classe s'appelle *classification* (ou *abstraction*). Une entité est une *instanciation* de la classe. Chaque entité est composée de propriétés, données élémentaires permettant de la décrire.

Prenons par exemple une *Toyota KE70*, une *Renault 12* et une *Peugeot 504*. Il s'agit de 3 entités faisant partie d'une classe d'entité que l'on pourrait appeler *voiture*. La Toyota KE70 est donc une instanciation de la classe voiture. Chaque entité peut posséder les propriétés *couleur*, *année* et *modèle*.

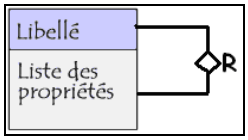
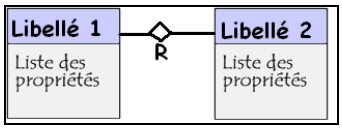
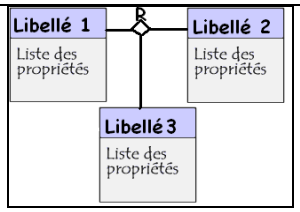
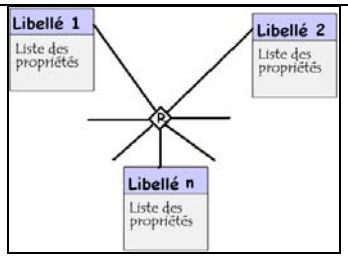
Les classes d'entités sont représentées par un rectangle. Ce rectangle est séparé en deux champs:

- le champ du haut contient le libellé. Ce libellé est généralement une abréviation pour une raison de simplification de l'écriture. Il s'agit par contre de vérifier qu'à chaque classe d'entité correspond un et un seul libellé, et réciproquement.
- le champ du bas contient la liste des propriétés de la classe d'entité

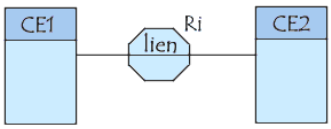
2.3.3 Relations et classes de relation

Une *relation* (appelée aussi parfois *association*) représente les liens sémantiques qui peuvent exister entre plusieurs entités. Une *classe de relation* contient donc toutes les relations de même

type (qui relient donc des entités appartenant à des mêmes classes d'entité). Une classe de relation peut lier plus de deux classes d'entité. Voici les dénominations des classes de relation selon le nombre d'intervenants:

		
une classe de relation ***récursive (ou réflexive)*** relie la même classe d'entité	une classe de relation ***binaire*** relie deux classes d'entité	une classe de relation ***ternaire*** relie trois classes d'entité
		
	une classe de relation ***n-aire*** relie n classes d'entité	

Les classes de relations sont représentées par des *octogones* (parfois des *ellipses* ou des *exagones*) dont l'intitulé décrit le type de relation qui relie les classes d'entité (généralement un *verbe*). On définit pour chaque classe de relation un identificateur de la forme **Ri** permettant de désigner de façon unique la classe de relation à laquelle il est associé.

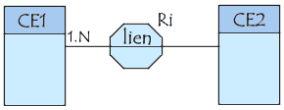


On peut éventuellement ajouter des propriétés aux classes de relation. Ces propriétés sont alors appelées *propriétés d'association*. Ce sont des propriétés qui ne peuvent appartenir à aucune classe d'entité faisant partie de l'association. Par exemple, la *note* est une propriété d'association entre une classe d'entité *matière* et une classe d'entité *étudiant*.

2.3.4 La cardinalité

Les cardinalités permettent de caractériser le lien qui existe entre une entité et la relation à laquelle elle est reliée. La cardinalité d'une relation est composée d'un couple comportant une borne maximale et une borne minimale, intervalle dans lequel la cardinalité d'une entité peut prendre sa valeur:

- la borne minimale (généralement 0 ou 1) décrit le nombre minimum de fois qu'une entité peut participer à une relation
- la borne maximale (généralement 1 ou n) décrit le nombre maximum de fois qu'une entité peut participer à une relation



Une cardinalité 1.N signifie que chaque entité appartenant à une classe d'entité participe au moins une fois à la relation.

Une cardinalité 0.N signifie que chaque entité appartenant à une classe d'entité ne participe pas forcément à la relation.

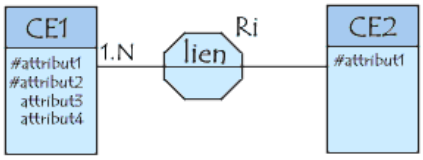
2.3.5 Les identifiants

Un *identifiant* est un ensemble de propriétés (une ou plusieurs) permettant de désigner une et une seule entité. La définition originale est la suivante :

L'identifiant est une propriété particulière d'un objet telle qu'il n'existe pas deux occurrences de cet objet pour lesquelles cette propriété pourrait prendre une même valeur.

Les attributs d'une classe d'entité permettant de désigner de façon unique chaque instance de cette entité sont appelés *identifiants absolus*.

Le modèle conceptuel des données propose de faire précéder d'un **#** les identifiants, ou de *les souligner*.



Ainsi, chaque classe d'entité doit posséder au moins un attribut identifiant, et l'ensemble de ses attributs identifiants doivent être renseignés à la création de l'entité.

2.4 Le modèle Logique de Données (MLD)

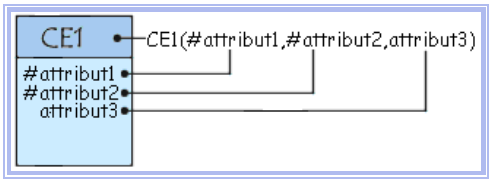
2.4.1 Description

Le modèle logique des données consiste à décrire la structure de données utilisée sans faire référence à un langage de programmation. Il s'agit donc de préciser le type de données utilisées lors des traitements.

Ainsi, le modèle logique est dépendant du type de base de données utilisé.

2.4.2 Traduction d'une classe d'entités

Chaque classe d'entité du modèle conceptuel devient une relation dans le modèle logique. Les identifiants de la classe d'entité sont appelés clés de la relation, tandis que les propriétés standards deviennent des attributs de la relation.



2.4.3 Traduction d'une classe de relation

Le passage du modèle conceptuel au modèle logique au niveau des classes de relation se fait selon les cardinalités des classes d'entité participant à la relation :

- si une seule des classes d'entités possède une cardinalité (1,1) : la relation issue de cette classe d'entités sera modifiée en important l'identifiant de la classe d'entité participante comme clé étrangère et les éventuelles propriétés d'association comme attributs de relation.
- si les deux classes d'entités possèdent chacune une cardinalité différente de (1,1) : l'association devient une nouvelle relation avec pour attributs les identifiants des deux classes d'entités complétés par les éventuels propriétés d'association.

2.5 CAS PRATIQUES

2.5.1 Cas pratique N°1

Un nouvel opérateur de Télécoms nommé 4GTEL vient de s'installer au Cameroun. Dans un premier temps, il veut couvrir quelques grandes villes, notamment Yaoundé, Douala et Bafoussam. Chaque ville est quadrillée en cellules et dans une cellule, on pourra y installer une BTS. Chaque cellule est caractérisée entre autre par un code, des coordonnées géographiques (longitude, latitude, altitude) et une population. Ces coordonnées géographiques sont celles du lieu d'installation de la BTS dans la cellule. Une cellule englobe plusieurs quartiers et un quartier peut appartenir à plusieurs cellules différentes. Chaque ville possède au moins une BSC caractérisée entre autre par un code, ses coordonnées géographiques, et gérant une ou plusieurs BTS. Ces BSC sont gérées au niveau de chaque ville par une MSC caractérisée entre autres par un code et ses coordonnées géographiques. Chaque station (BTS, BSC ou MSC) est constituée d'équipements (Routeurs, Commutateurs, Switchs, Groupe Electrogènes, etc.) La société gère aussi des techniciens qui enregistrent à chaque intervention leurs activités sur un équipement, notamment la panne observée sur l'équipement, le listing des opérations menées, les observations particulières, l'état final du matériel (fonctionnel ou non fonctionnel). Chaque technicien est caractérisé entre autres par un code, son nom, son Email, sa ville de résidence son téléphone et son diplôme de recrutement.

Le modèle logique de données issu du modèle conceptuel de données est le suivant :
Station (***CodeStation***, NomStation, TypeStation, SuperficieStation, LongitudeStation, LatitudeStation, AltitudeStation, Cellule.CodeCellule*, Station.codeStation*)
Ville (***CodeVille***, NomVille, RegionVille, DepartementVille, PopulationVille)
Cellule (***CodeCellule***, NomCellule, SuperficieCellule, PopulationCellule, Ville.CodeVille*)
Equipement (***CodeEquip***, NomEquip, TypeEquipement, EtatEquip, Station.CodeStation*)
Technicien (***CodeTech***, NomTech, TelefTech, VilleTech, EmailTech, DiplomeTech)
Panne (***CodePanne***, NomPanne, CategoriePanne, DescriptifPanne)
PanneEquipement (***CodePanneEquip***, Equipement.CodeEquip*, Panne.CodePanne*, DatePanne)
Intervention (***CodeIntervention***, Technicien.CodeTech*, PanneEquipement.CodePanneEquip*, DateIntervention, DescriptifIntervention, ResultatIntervention)

2.5.2 Cas pratique N° 2

On désire gérer des concessions forestières dans un pays. Chaque concession forestière appartient à une région et est caractérisée par un code unique dans le pays, un nom, les coordonnées géographiques du point de référence, les activités humaines dans le milieu, les essences de bois dans le milieu, les espèces fauniques etc.

On voudrait à un moment donné localiser par exemple toutes les concessions forestières :

1. Qui ont une certaine proportion d'une essence comme le Bété, le Baobab ou l'Iroko.
2. Qui hébergent une espèce animale donnée (Lion, Éléphant…)
3. Qui font l'objet d'exploitation minière
4. Où on cultive un aliment particulier (Maïs, Sorgho,…)

Le modèle logique de données issu du modèle conceptuel de données est le suivant :

Animal(**CodeAnimal**, NomPilAnimal, FamilleAnimal, NomScAnimal)

Arbre(**CodeArb**, DiametreArb, HauteurArb, FamilleArb, NomScArb, EssenceArb, QualiteArbre, LongArb, LatArb, AssAnCoupe*)

AssietteCoupe(**CodeAss**, SuperficieAss, PossibiliteAss, LongAss, LatAss, Concession*, DateDebExpAss, DureeExpAss)

Bille(**CodeBille**, DiametreBille, LongueurBille, VolumeBille, QualiteBille, Arbre*)

Commercialise(**pers**, **animal**)

CommetDans(**Pers**, **Infraction**, **Dpt**, montantInfr)

Concession(**CodeConc**, SuperficieConc, TypeConc, EtatConc, LongConc, LatConc, Pers*, DateDebTit, DateFinTit)

Contient(**Concession**, **Animal**, Nombre)

Departement(**CodeDpt**, NomDpt, LongDpt, LatDpt, PopDpt, Prov*)

Emploie(**Prov**, **Pers**, FonctionPers, DateFoncPers)

Entre(**Bille**, **Usine**, DateEntree)

Est_Associe(**Localite**, **Concession**, Proximite)

Ethnie(**CodeEthnie**, NomEthnie)

Exerce(**Pers, Dpt**, **FonctionDpt**, DateFonc)

Exporte(**Pers**, **Bille**, VolumeExport, DateExport)

Fabrique(**Usine**, **Produit**, **Destination**, Volume, AnneeFabr)

Habite(**Ethnie**, **Localite**, NbFemme, NbHomme, NbEnfant)

Infraction(**CodeInfr**, TypeInfr, NomInfr)

Intervient(**Pers**, **Concession**, FoncPers, DateDebInt, DateArretInt)

Lie(**Concession**, **Texte**)

Localite(**CodeLoc**, NomLoc, LongLoc, LatLoc, AltLoc, PopLoc, Dpt*)

Paye(**Concession**, **Taxe**, DateTaxe, MontantTaxe)

Permis(**CodePer**, NomPer, TypePer, Pers*, montantPermis)

Personne(**CodePers**, NomPers, InteretsPers, BPPers, TelPers, EmailPers, TypePers)

Possede(**Concession**, **Ressource**, Proportion)

Produit(**CodeProd**, NomProd, EssenceProd, QualiteProd)

Province(**CodeProv**, NomProv, LongProv, LatProv, PopProv)

Ressource(**CodeRes**, NomRes, TypeRes)

Taxe(**CodeTaxe**, TypeTaxe, NomTaxe)

Texte(**CodeTexte**, DateTexte, TypeTexte, TitreTexte)

Transforme(**Usine**, **Bille**, DateTrans)

Travaille(**Perso**, **Localite**, FonctionLoc)

Usine(**CodeUs**, TypeUs, NomUs, SegmentActUs, LongUs, LatUs, CapTheoUs, CapReelUs, Localite*, Pers*, DateImpl)

On peut dès à présent rédiger des requêtes d'interrogation de la base de données.

CHAPITRE 3 INTRODUCTION A PHP/MYSQL

3.1 Exemple de code

3.1.1 Entête de page PhP

```
<HTML>
<HEAD>
<title>ENTREZ LE TITRE DE VOTRE PAGE ICI</title>
</HEAD>
```

3.1.2 Corps de page PhP

PhP Commence comme toute page HTML.

<BODY BGCOLOR=white>

Insertion d'une table

```
<table width="100%" border="0">
 <tr>
  <td width="17%" align="center" background="Chemin_Acces/NomImage ">
```

Insertion d'une image

```
<img src="Chemin/NomLogo" alt="Logo du projet" width="40%"></td>
  <td align="center" background=" Chemin_Acces/NomImage ">
```

Commentaire

```
<!-- BARRE D'ENTETE -->
Insérer le menu horizontal  </td>
 </tr>
 <tr>
  <td valign="top" background=" Chemin_Acces/NomImage " align="center"> </td>
  <td align="center" valign="middle"> Contenu du cadre principal</td>
 </tr>
```

Insertion d'un code Php

```
 <tr>
   <td colspan="2" align="center">
   <?
   #Récupération/Déclaration des paramètres de connexion
   $username= "MonLogin";# $_POST["login"];
   $password= "MonPassword"; #$_POST["password"];
   $hostname="localhost";
   $nombase="MaBase";
   $nomtable="UneTable";
```

```
  #On se connecte a MySql
  if(!($connexion=mysql_connect($hostname,$username,$password)))
   {
    echo "<font size=3 >Echec de connexion &agrave ; MYSQL</font>";
    exit();
   }
  else # Connexion à MYSQL réussie
   {
    #On se connecte a la base
    if(!mysql_select_db($nombase,$connexion))
     {
      echo ("Erreur de sélection de la base -->$nombase<--<br>");
            exit();
     }
    else #Connexion à la base de données réussie
     {
         // Requete SQL
         $requete=" Ma requête" ;
         //Récupération du résultat
         $requete=mysql_db_query($nombase,$requete) or die ('<p align=center><b><font
color=red>Vérifiez cette requête</font></b></p>
                        <br><br> '.$requete.' <br>'.mysql_error().'</p>');
                . // Récupération du nombre de lignes
                $nblignes=mysql_num_rows($requete);
 echo "<tr><td colspan='2' align='center'><hr>
         <table border=0 align=center>
                <tr><td><font size=+1 color=green face='Monotype Corsiva'><b> Entête de la
         requête</b></font></td><td width='3%'></td> ";
         for($lignes=1;$lignes<=$nblignes;$lignes++)
          {
           $ligne_pays=mysql_fetch_row($requete);
           echo"<td><a
href=resultat/resultat_info.php?resultat=$ligne_resultat[0]&Champ1_resultat=$ligne_resultat[1]>
$ligne_resultat[1]</a></td>";
          }
                echo"</tr></table>";
  }
mysql_close($connexion);
}
   ?>
  </td>
 </tr>
</table>
</BODY>
</HTML>
```

3.2 Les formulaires

```
<?
echo '<BR><br>
        <form method="post" action="Page_A_Ouvrir.php">
        <br>
        <CENTER><font size=+3 color="#006666" face="Monotype Corsiva, Bookman Old Style, Book Antiqua"><b>Veuillez vous identifier</b></font></CENTER><br><br>
        <table align="center">
        <tr><td><font size="=+2">Entrez votre login :</font></td><td><font size="=+1"><input type="text" name="login" size="15"></font></td></tr>
        <tr><td><font size="=+2">Entrez votre mot de passe :</font></td><td><input type="password" name="password" size="15"></td></tr>
        </table>
        <br><p align="center"><input type="submit" name="soumettre" value="VALIDER"></p>
        </form>';
?>
```

3.3 Récupération des variables de formulaire

```
<?
.#Récupération des paramètres de connexion
$username= $_POST["MonLogin"];
$password= $_POST["MonPassword"];
$hostname="localhost";
$nombase="MaBase";
$type= $_POST["type"];
$texte= $_POST["texte"];
#On se connecte a MySql
if(!($connexion=mysql_connect($hostname,$username,$password)))
 {
  echo "<br><p align=center ><font size=3 color=red>Echec de connexion à MYSQL</font></p>";
   exit();
 }
else # Connexion à MYSQL réussie
 {
  #On se connecte a la base
  if(!mysql_select_db($nombase,$connexion))
   {
    echo ("<br><p align=center ><font size=3 color=red>Erreur de sélection de la base -->$nombase<--</font></p><br>");
        exit();
   }
  else #Connexion à la base de données réussie
```

```
    {
                 // Nouvelle requête
                $requete="Une autre requête";
                .//Récupération du résultat
                $requete_resultat=mysql_db_query($nombase,$requete) or die ('<br><p
align=center><font color=red size=2>Vérifiez cette requète</font></p><br> '.$requete.'
<br>'.mysql_error());
                . // Numbre de colonnes et nombre de lignes
                $nbcol=mysql_num_fields($requete_resultat);
                $nblignes=mysql_num_rows($requete_resultat);
                . // Première ligne du tableau résultat
                  echo "<p align=center><font size=+2 color=green face='Monotype
Corsiva'><b>$texte</b></font></p>
                        <table border=1 align=center>
                         <tr>
                                <td bgcolor=blue><b><font color=white>N°</font></b></td>
                                <td                                          bgcolor=blue><b><font
color=white>Champ1</font></b></td>
                                <td bgcolor=blue><b><font color=white>Champ2 font></b></td>
                                <td                                          bgcolor=blue><b><font
color=white>Champ3</font></b></td>
                        </tr>";
                        for($lignes=1;$lignes<=$nblignes;$lignes++)
                         {
                        $ligne_requete=mysql_fetch_row($requete_resultat);
                                echo"<tr><td><b>$lignes</td>";// Début d'une nouvelle ligne
                                for($i=0;$i<$nbcol-2;$i++)
                                 {
                                        echo"<td><font              face='Arial              Narrow'><a
href='$ligne_requete[0].pdf'>$ligne_requete[$i]</font></td>";
                                 }
                                echo"<td><font                                          face='Arial
Narrow'>$ligne_requete[2]</font></td></tr>";// Fin de la ligne
                         }
                echo"</table>";
  }#Fin de connexion à la base de données
   mysql_close($connexion);
 }#Fin de connexion à MYSQL
   ?>
```

CHAPITRE 4 FONDAMENTAUX SUR JAVA

4.1 Introduction

- En 1995, Lorsque Sun Microsystem propose le langage de programmation JAVA, il est un outil pour le web. Il est utilisé pour ajouter l'interactivité aux pages web. Aujourd'hui, il est utilisé pour développer des applications diverses :
- Traitement de texte ;
- Serveur Web;
- Base de données relationnelles
- Et sur du matériel varié :
- Mainframe ;
- Téléphone ;
- PDA ;
- Télescope.

4.2 PRESENTATION DU LANGAGE

- Java est un langage de programmation qui est bien adapté au développement des applications utilisant Internet. Il est aussi un langage de programmation oriente objet. Il est portable ce qui signifie que les programmes développés en JAVA s'exécutent sans aucune modification sur les plates-formes Microsoft Windows, Apple Macintosh, Linux, Solaris, etc.
- La version embarquée de JAVA permet de développer des applications qui tournent sur les téléphones cellulaires, PDA, Téléviseur, etc.
- JAVA est plus proche des langages C, C++, Python, Visual Basic, et Delphi que d'un langage de description de page tel que HTML, ou du langage de script JavaSript.

4.3 Programmation interactive sur le web

- La popularité de JAVA provient de sa capacité à s'exécuter dans une page Web. Netscape, Internet Explorer, et d'autres navigateurs peuvent télécharger un programme JAVA inclus dans une page Web et l'exécuter sur la machine locale. Ces programmes appelés applet ou appliquette, apparaissent dans les pages Web comme des images. Différemment des images, une applet JAVA peut être interactive. L'utilisateur peut utiliser une applet pour entrer des données, Après

traitement recevoir des résultats. Les applets peuvent être utilisés pour créer des animations, des graphiques, des jeux, des menus de navigation, des présentations multimédia, et d'autres effets interactifs. Les applets sont télécharges sur le web comme des pages HTML, les images ou comme tout autre élément de la page web. Sur un navigateur incluant le plug-in JAVA, des que le téléchargement d'un applet se termine, il est immédiatement exécuté.

- Les applets sont écrits en langage JAVA, compilés dans une forme qui peut s'exécuter comme un programme et placés sur un serveur web. Beaucoup d'applet sont écrit en Java 1.0 ou Java 1.1, qui sont les deux premières versions du langage parce que beaucoup de navigateur ont mis du temps avant d'intégrer les nouvelles version du langage. Aujourd'hui il est possible de développer des applets en utilisant JAVA 2.
- Comme Visual C++, Visual Basic, et Delphi, JAVA est un langage robuste qui peut être utilisé pour développer un grand nombre de logiciel, supportant les interfaces graphiques, le réseau, la connectivité aux bases de données et d'autres fonctionnalités sophistiquées.
- Un programme JAVA qui ne s'exécute pas dans une page Web est Appelle Application.

4.4 Principales caractéristiques du langage

- Comme JAVA a été développé pour des systèmes électroniques embarqués et non pour les PC, il se devait d'être petit, efficace et facilement transportable d'une plate forme matériel a l'autre. Au même moment on observe que le web devient de plus en plus populaire.
- JAVA est petit : les programmes se chargent raisonnablement vite sur une page web ;
- JAVA est sécurisé : il protège la machine contre les programmes qui peuvent causer des dommages de manière accidentelle ou intentionnelle.
- JAVA est portable : les utilisateurs de Windows, Macintosh, Linux et d'autres systèmes d'exploitation peuvent exécuter le même programme dans leurs navigateurs sans aucune modification.

4.5 Versions de java

- Le langage Java a connu plusieurs évolutions depuis le JDK (*Java Developpement Kit*) 1.0 avec l'ajout de nombreuses classes et packages à la bibliothèque standard. Depuis le J2SE1.4, l'évolution de Java est dirigée par le JCP (*Java Community Process*) qui utilise les JSR (*Java Specifications Requests*) pour proposer des ajouts et des changements sur la plate-forme Java. Le

langage est spécifié par le JLS (*Java Language Specification*). Les modifications du JLS sont gérées sous le code JSR 901.

- **JDK 1.0** (23 janvier 1996 - 211 classes et interfaces) — Version initiale.
- **JDK 1.1** (19 février 1997 - 477 classes et interfaces) — De nombreux ajouts avec notamment :
  - Une refonte complète du modèle événementiel AWT ;
  - Les classes internes sont ajoutées au langage ;
  - JavaBeans ;
  - JDBC ;
  - Java Remote Invocation (RMI).
- **J2SE 1.2** (9 décembre 1998 - 1524 classes et interfaces) — Nom de code *Playground*. Cette version et les suivantes jusque J2SE 5.0 sont rebaptisées **Java 2** et la version nommée J2SE (Java 2 Platform, Standard Edition) remplace JDK pour distinguer la plateforme de base de la version J2EE (Java 2 Platform, Enterprise Edition) et de la version J2ME (Java 2 Platform, Micro Edition). Plusieurs ajouts avec notamment :
  - Le mot-clé strictfp
  - La réfection
  - l'API graphique Swing est intégrée.
  - Pour la première fois, la machine virtuelle Java de Sun inclut un compilateur "Juste à temps" (*Just in Time*).
  - Java Plug-in
  - Java IDL, une implémentation de IDL pour l'interopérabilité avec CORBA.
  - le framework Collections.
- **J2SE 1.3** (8 mai 2000 - 1840 classes et interfaces) — Nom de code *Kestrel*. Le changement le plus important avec notamment :
  - HotSpot JVM inclus (La machine virtuelle HotSpot sortit en Avril 1999 pour la machine virtuelle du J2SE 1.2)
  - Changement pour les RMI pour être basé sur CORBA.
  - JavaSound
  - JNDI (Java Naming and Directory Interface) inclus de base (disponible auparavant comme extension)

- JPDA (Java Platform Debugger Architecture)
- **J2SE 1.4** (*6 février 2002 - 2723 classes et interfaces*) — Nom de code *Merlin*. Ce fut la première révision de la plateforme sous JCP (*Java Community Process*) JSR 59. Les principaux changements sont :
- Le mot-clé assert (Spécifié dans JSR 41.)
- Les expressions rationnelles modélisées en s'inspirant du langage Perl.
- Le chaînage d'exception permet à une exception d'encapsuler l'exception de base niveau d'origine. (Spécifié dans JSR 51.)
- API de connexion (Spécifiée dans JSR 47.)
- l'API Image I/O pour lire et écrire des images dans des formats comme JPEG et PNG.
- Intégration d'un parser XML et du moteur XSLT nommé JAXP (Specifié dans JSR 5 et JSR 63.)
- Intégration des extensions de sécurité JCE, JSSE et JAAS.
- Java Web Start (introduit pour la première fois en mars 2001 pour J2SE 1.3 - Spécifié dans JSR 56.)
- **J2SE 5.0** (30 Septembre 2004 - 3270 classes et interfaces) — Nom de code *Tiger*. (Initialement numérotée 1.5, qui est toujours utilisé comme numéro de version interne). Développé par JSR 176, Tiger ajoute un nombre significatif de nouveautés (communiqué de presse) au langage :
- Programmation générique — (Spécifié par JSR 14.)
- Metadata — également appelées annotations, permet au langage de construire des classes et des méthodes étiquetées avec des données additionnelles qui peuvent être utilisées en tant que méta-données (Spécifiée dans JSR 175.)
- Autoboxing/unboxing — conversion automatique entre des types primitifs (comme le type **int**) et le Wrapper de classe correspondant (comme la classe **Integer**) (Spécifié dans JSR 201).
- Énumérations — le mot-clé ***enum*** permet de créer une liste ordonnée de valeurs sans type. Auparavant, ceci pouvait seulement être réalisé par des entiers constants (Spécifié dans JSR 201).
- Varargs — la syntaxe Object ... utilisée dans une déclaration de méthode permet de spécifier un nombre variable d'arguments pour cette méthode. C'est un fonctionnement équivalent à la fonction "printf" en C.

- Imports statiques — Cette fonctionnalité permet d'utiliser les constantes d'une classe sans spécifier le nom de cette classe et sans passer par "l'anti-pattern Constant Interface" (c'est l'expression utilisée sur le site de Sun).
- **Java SE 6** (11 décembre 2006 - ??? classes et interfaces) — Nom de code *Mustang*. (voir JSR 270). Une version bêta est sortie le 15 février 2006, une autre bêta en juin 2006, une version "release candidate" en novembre 2006, et la version finale le 12 décembre 2006. Avec cette version, Sun remplace le nom J2SE par Java SE et supprime le .0 au numéro de version.
- **Java SE 7** — Nom de code *Dolphin*. En 2006, cette version est encore à un stade de développement précoce. Le développement commencé à l'été 2006 s'est achevé en 2008. C'est la première version sous la licence GPL.
- En plus des changements au niveau du langage, des changements plus importants ont eu lieu au fil des années qui ont conduit des quelques centaines de classes dans le JDK 1.0 à plus de 3000 dans J2SE 5.0. Des API entières, comme Swing ou Java2D ont été ajoutées et beaucoup de méthodes de l'original JDK 1.0 ont été déclarées *deprecated* (c'est-à-dire obsolètes et pouvant être supprimées à tout moment).
- Un Kit de développement Java est gratuitement disponible sur le site de Sun à l'adresse http://java.sun.com. Cette disponibilité est un facteur déterminant dans la croissance rapide du langage. En addition au Kit de développement, il existe des outils de développement gratuits et commerciaux.
- Parmi les outils commerciaux on peut citer : JBuilder, Microsoft Visual J++, Symantec Visual Café, Tek-Tools Kawa.
- Parmi les gratuits, nous avons : Bluette, JDK commander, Kopi Suite, Sash for Windows, NetBeans, Eclipse etc.
- Les programmes de secours ont été teste avec **J2SE 5.0**

4.6 Pourquoi choisir java?

Les applets Java augmentent l'interactivité du contenu d'une page Web et plusieurs sites leaders utilisent cette technologie pour délivrer les informations et attirer des nouveaux utilisateurs. Les forces de JAVA restent :

Sa nature orienté objet,

Sa facilite d'apprentissage et sa portabilité.

JAVA est orienté objet en ce sens qu'il permet de concevoir un programme comme un ensemble séparé d'objets interagissant entre eux. (Un objet contient à la fois l'information et la manière d'accéder et de modifier cette information) - une manière efficace de créer un programme informatique qui sera facilement modifiable et utilisé plus tard pour résoudre d'autres problèmes.

JAVA est facile à apprendre : JAVA a été crée par le projet Green à cause de leur rejet du C++ pour sa complexité. C++ est un langage avec de nombreuses instructions très puissantes, mais facile à employer incorrectement.

JAVA a pour intention d'être facile à écrire, à compiler, à déboguer et à apprendre contrairement aux langages orientés objet. Sa syntaxe s'inspire fortement de celle de C++.

En résume, JAVA est facile à apprendre que plusieurs autres langages de programmation. Une personne sans une grande expérience en programmation peut facilement le prendre en main.

JAVA est portable : JAVA s'exécute sur une variété de plates formes matérielles. Ceci est un atout que JAVA possède par rapport aux autres environnements de programmation (Visual Basic, Visual C++...) développés exclusivement pour Microsoft Windows.

Dans la plupart des langages de programmation, lorsqu'on compile un programme, le compilateur traduit le code source en code machine (instruction spécifique au processeur de votre machine). Si vous compilez votre programme sur un système Windows, il sera exécutable sur les autres systèmes Windows mais pas sur un Mac, un Palm pilot ou tout autre machine. Si vous voulez utiliser le même programme sur une autre plate forme, vous devez d'abord transférer votre code source sur la nouvelle plate forme et le recompiler pour produire un code machine spécifique pour ce système. Dans la plupart des cas, la modification du code source est nécessaire pour que le programme compile sans erreurs.

Les programmes Java sont compilés en code machine pour une machine virtuelle (un programme qui simule le fonctionnement d'une autre machine). Ce code machine est appelé ***bytecode*** et la machine virtuelle interprète ce code en le traduisant en des instructions spécifiques au processeur de la machine hôte. La machine virtuelle est communément appelée Interpréteur Java et chaque environnement supportant doit avoir un interpréteur Java spécialement conçu pour son système d'exploitation et son processeur.

JAVA est aussi portable au niveau du code source. Un programme java est sauvegardé comme un fichier texte avant d'être compilé. Et ces fichiers sources peuvent être créé sous n'importe quelle

plate forme supportant JAVA. Par exemple, vous pouvez écrire un programme java pour Windows 98, le charger dans une machine linux à travers Internet et le compiler.

L'interpréteur java est disponible à plusieurs endroits (http://java.sun.com). Pour les applets, l'interpréteur est soit intégré au navigateur, soit installé comme un plug-in. L'interpréteur java pose quelques problèmes significatifs de performance. Le byte code java s'exécute plus lentement que le code machine produit à l'aide des langages compilés tels que C ou C++.

4.7 Installation du sdk

Si vous utilisez Microsoft Windows ou Apple MacOS system, vous avez probablement un interpréteur java installé qui peut exécuter un programme java. Habituellement, cet interpréteur est une partie du navigateur Web et ne peut exécuter que des applets.

4.7.1 Sélectionner un outil de développement

Pour développer un programme java, vous avez besoin de plus qu'un interpréteur. Vous allez avoir besoin d'un compilateur, d'un interpréteur et d'autres outils permettant de créer, d'exécuter et de déboguer des programmes. Les programmes de ce cours sont testés avec le SDK 1.4 qui est un ensemble d'outils en ligne de commande incluant un compilateur, un interpréteur, un appletviewer, un archiveur de fichiers et plusieurs autres programmes.

Un programme en ligne de commande est un programme qui s'exécute au prompt d'une fenêtre MS-DOS

Un exemple de commande que vous pouvez tapez lorsque vous utilisez le Kit de développement est : `javac test.java`

Cette commande lance le programme javac intégré dans le Kit de développement JAVA 2 SDK 1.5 de lire un fichier source nommé `test.java` et de le convertir en un fichier contenant du byte code. Les personnes qui sont confortables avec MS-DOS, Linux et d'autres environnements en ligne de commande seront plus à l'aise avec le SDK 1.5. Toutes les autres personnes devront s'habituer à ce type d'environnement en ligne de commande.

4.7.2 Installation du sdk sous Windows.

SDK 1.5 est disponible pour les plates formes suivantes Windows 95, Windows 98, Windows NT, Windows 2000, Windows XP.

Avant d'installer le SDK, vous devez vous assurer qu'aucune autre version n'est installée sur votre système. Avoir une autre version installée pourra causer des problèmes de configuration lorsque vous utilisez le SDK.

Fermer toutes les autres applications avant de commencer l'installation du SDK.

Pour lancer le programme d'installation sur un système Windows, double-cliquer sur le fichier d'installation du SDK et suivez les instructions indiquées par l'assistant d'installation SDK.

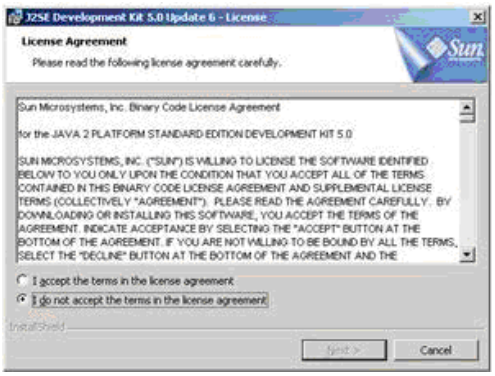


Figure1.1. Fenêtre de lancement de l'installation de J2SE.

L'assistant d'installation du SDK vous suggère un chemin d'installation par défaut. Les fichiers et les programmes nécessaires au bon fonctionnement du SDK seront copiés dans ce répertoire.

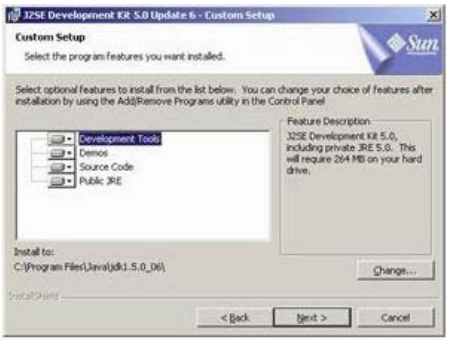


Figure1.2. Fenêtre de sélection des composants à installer.

Par défaut, l'assistant d'installation installe tous les composants :

- les fichiers programmes nécessaires pour créer, compiler et tester les projets java ;
- les fichiers d'en-tête des interfaces natives utilisées seulement par les programmeurs qui veulent combiner leurs codes java avec des programmes écrits dans d'autres langages ;
- les Démos qui sont des exemples de programmes écrits en JAVA qu'on peut utiliser pour apprendre plus sur le langage ;

- les sources java qui constituent le code source de près de mille programmes constituant la librairie de classes java.

Après avoir choisi les différents composants à installer, cliquer sur le bouton suivant de l'assistant de configuration pour installer le SDK sur le système.

Figure1.3. Fenêtre de fin d'installation du programme.

4.7.3 Configuration du SDK

Après que l'assistant de configuration ait installé le SDK sur le système, il faut ajuster quelques variables d'environnement pour faire référence au SDK. Les utilisateurs MS-DOS expérimentés achèvent la configuration du SDK en modifiant deux variables système :

Éditer la variable système ***PATH*** et ajouter la référence au répertoire ***bin*** du SDK

Supprimer la variable ***CLASSPATH*** si d'autres programmes java ne sont pas utilisés.

4.7.3.1 Configuration des variables path et classpath sous windows NT, 2000

Pour modifier une variable d'environnement dans les systèmes Windows NT, 2000 et XP, cliquer sur le menu ***démarrer -> panneau de configuration -> système*** ; Cliquer ensuite sur l'onglet « ***Environnement*** ». Une boîte de dialogue s'ouvre où apparaissent les valeurs des variables d'environnement. La variable PATH doit être listée parmi les variables systèmes ou les variables d'utilisateur. PATH contient une liste de répertoires séparés par des "**;**"

Au début de la variable PATH, insérer le chemin d'accès au répertoire ***BIN*** du SDK suivi d'un "**;**"

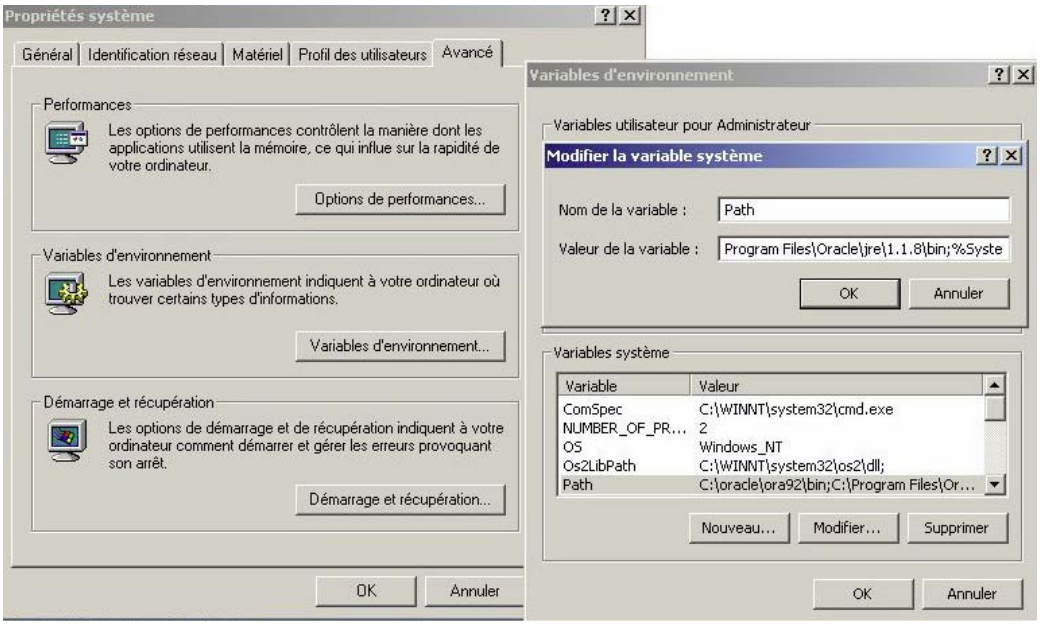


Figure1.4. Fenêtre de configuration des variables d'environnement sous Ms NT et 2000.

Après avoir effectué cette modification, rechercher une variable ***CLASSPATH***. Si une telle variable n'existe pas, alors la configuration est bonne. Cliquer sur le bouton « *Appliquer* » pour enregistrer les modifications et redémarrer l'ordinateur.

Par contre, si la variable ***CLASSPATH*** existe, placer le curseur à la gauche de la valeur de celle-ci et insérer le chemin d'accès au fichier tools.jar.

Après avoir fixé les valeurs de la variable ***CLASSPATH***, cliquer sur le bouton « *Appliquer* » pour enregistrer les modifications et redémarrer l'ordinateur.

4.7.3.2 Configuration des variables *path* et *classpath* sous windows xp

Pour modifier une variable d'environnement dans les systèmes Windows, XP, cliquer sur le menu ***démarrer -> panneau de configuration -> Perfoprmance et Maintenance -> Système*** ; Cliquer ensuite sur l'onglet « ***Avancé*** ». Une boîte de dialogue s'ouvre cliquer sur le bouton ***variables d'environnement***. Une boîte de dialogue s'ouvre où apparaissent les valeurs des ***variables d'environnement***. La variable PATH doit être listée parmi les variables systèmes ou les variables d'utilisateurs. PATH contient une liste de répertoires sépares par des "**;**"

Sélectionnez la ligne qui contient la variable PATH et cliquez sur le bouton modifier qui se trouve en dessous.

Au début de la variable PATH, insérer le chemin d'accès au répertoire **BIN** du SDK suivi d'un ";"

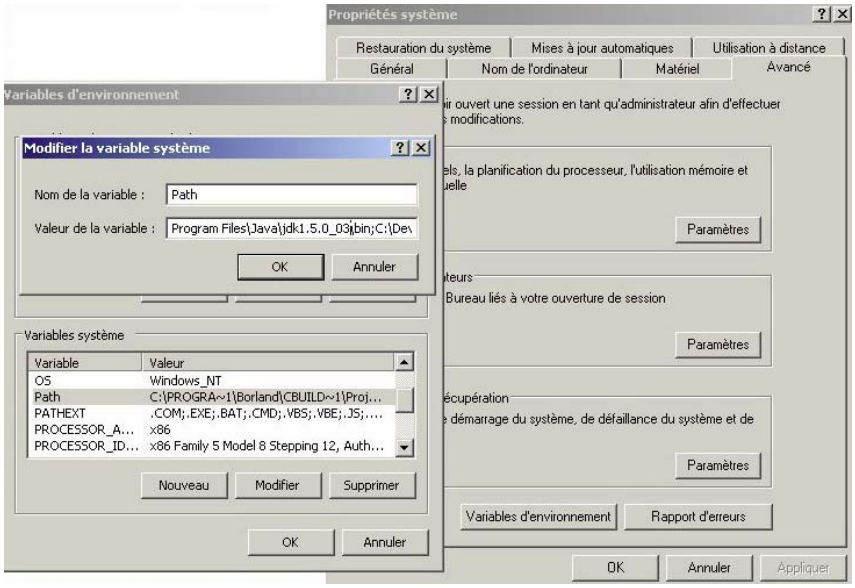


Figure1.5. Fenêtre de configuration des variables d'environnement sous Ms XP.

Après avoir effectué cette modification, rechercher une variable CLASSPATH. Si vous ne la trouvez pas alors votre configuration est bonne. Cliquer sur le bouton OK pour enregistrer les modifications et redémarrer votre ordinateur.

Dans le cas ou la variable CLASSPATH existe, placer le curseur à la gauche de la valeur de celle-ci, insérer le chemin d'accès au fichier tools.jar.

Après avoir fixe les valeurs de la variable CLASSPATH, cliquer sur le bouton OK pour enregistrer les modifications et redémarrer votre ordinateur.

4.8 Le premier programme java

Maintenant que vous avez termine d'installer votre SDK, vous êtes près à travailler avec JAVA

4.8.1 Création du fichier source

Un programme java commence par le code source qui est une série d'instructions crées en utilisant un éditeur de texte ou un logiciel de traitement de texte. Le fichier doit être sauvegarde au format texte avec l'extension `.java.` Les utilisateurs de Windows peuvent utiliser les programmes Bloc note, Word Pad, pour créer leur code source.

Sous Unix et Linux, les éditeurs Emacs, pico et vi sont disponibles. Les utilisateurs de Macintosh peuvent utiliser SimpleText pour créer leur code source JAVA.

Le SDK n'inclut pas d'éditeurs de texte. Mais beaucoup d'autres outils de développement java ont un éditeur de texte intègre.

4.8.2 Écrire un programme java

Lancer un éditeur de texte quelconque et saisir exactement le programme JAVA présenté ci-après en respectant les parenthèses, les accolades et les marques de ponctuation présents dans le code. Respecter la casse c'est à dire les majuscules et les minuscules doivent être saisies exactement comme dans le texte. Sauvegarder votre document sous le nom `HelloWorld.java.`

```
public class HelloWorld {
public static void main(String[] arguments) {
System.out.println("Hello world");
}
}
```

NB. Sous Windows, un éditeur de texte comme Bloc Note ajoute une extension `.txt` à tous fichiers source java lors de l'enregistrement. Pour résoudre ce problème, dans la boîte de dialogue « *Enregistrer* », taper le nom du fichier entre doubles quottes. Le fichier source java est sauvegardé avec l'extension `.java`.

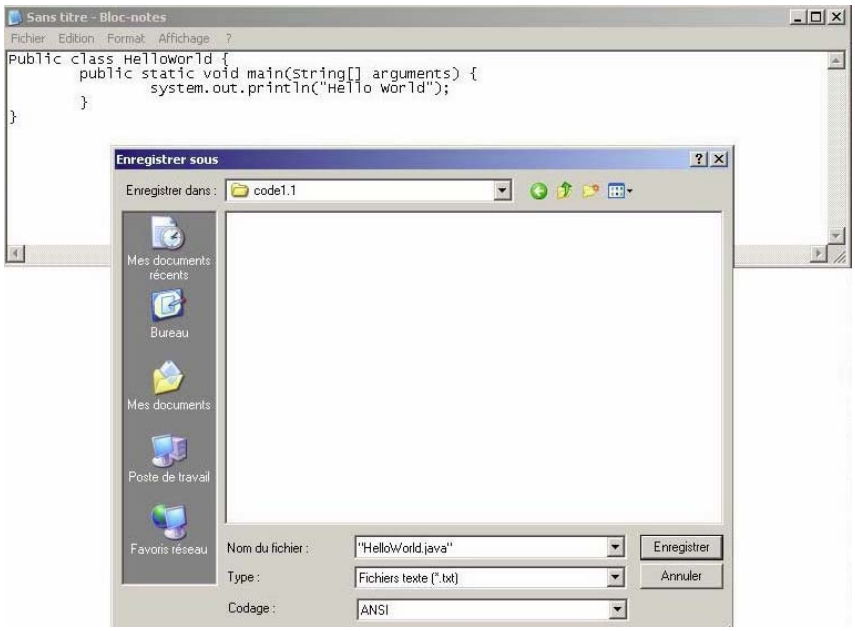


Figure1.6. Fenêtre d'enregistrement d'un code java à partir de Ms Bloc Notes.

4.8.3 Compiler et exécuter un programme java sous Windows

Avec le SDK on a besoin de l'outil en ligne de commande `javac` qui est le compilateur Java. Le compilateur lit un fichier source ***.java*** et crée un autre fichier nommé ***.class*** qui est exécutable par la machine virtuelle.

Les utilisateurs Windows doivent ouvrir une fenêtre MS-DOS et changer de répertoire si nécessaire. A partir du répertoire contenant le programme java, entrer la commande suivante:

Figure1.6. Fenêtre de compilation d'un programme java.

```
javac HelloWord.java
```

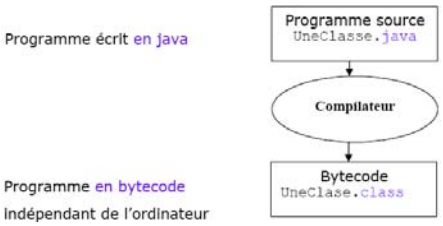
Si le compilateur ne renvoit aucun message d'erreurs, cela signifie que le programme est correct et qu'un fichier `HelloWord.class` a été crée. Le fichier d'extension `.class` contient le bytecode qui sera exécute par l'interpréteur java. Si une erreur apparaît alors que la configuration est bonne alors vérifier le fichier source pour vous rassurer qu'il n'y a aucune erreur. Lancer l'exécution du fichier ***.class*** obtenu, utilisant l'interpréteur java. L'interpréteur du SDK est appelé java et s'exécute en ligne de commande.

Pour exécuter le programme `HelloWord`, taper la commande suivante:

```
java HelloWord
```

Figure1.7. Fenêtre d'exécution d'un programme java.

La classe *HelloWorld* est public, donc le fichier qui le contient doit s'appeler (en tenant compte des majuscules et minuscules) `HelloWorld.java`



CHAPITRE 5 DEBUTER EN JAVA

5.1 Instructions et expressions

Toutes les tâches à accomplir dans un programme Java peuvent être décomposées en une série d'instructions. Une instruction est une simple commande écrite dans un langage de programmation qui provoque l'apparition d'un événement. Une instruction représente une action unique exécutée par un programme Java.

Nous avons ci-dessous quelques exemples d'instructions Java :

```
int poids = 295;
System.out.println(" je pense donc je suis");
song.duree = 230;
```

Certaines instructions peuvent convertir une valeur. Ce type d'instruction est appelé expression.

Une expression est une instruction donc le résultat est une valeur à produire. La valeur peut être stockée pour être utilisée plus tard dans le programme. Elle peut être utilisée immédiatement dans une autre instruction ou abandonnée. La valeur produite par une instruction est appelée valeur de retour. Certaines instructions produisent une valeur de retour numérique, par exemple lorsqu'on additionne deux nombres ensemble. Certaines produisent des valeurs booléenne (Vrai ou Faux) et d'autres peuvent produire des objets java.

Plusieurs programmes java listent une instruction par ligne. Ceci est une décision de formatage qui permet de distinguer facilement le début et la fin d'une instruction. Chaque instruction en java se termine par le caractère "**;**"

Un programmeur peut mettre plus d'une instruction par ligne et son programme compilera avec succès comme illustré ci-dessous.

```
dante.speed = 2; dante.temperature = 510;
```

Les instructions en java sont groupées à l'aide de l'accolade ouvrante "**{**"et de l'accolade fermante "**}**". Un groupe d'instructions se trouvant entre ces deux symboles est appelé bloc d'instructions.

5.2 Variables et type de données

Une variable est une zone de la mémoire ou une information peut être stockée quand un programme est en cours d'exécution. La valeur de la variable peut être changée à n'importe quel point dans le programme. Pour créer une variable, il faut lui donner un nom et identifier quel type d'informations il doit stocker. On peut aussi lui donner une valeur initiale à la création.

Il existe trois types de variables en java :

- les variables d'instance ;
- les variables de classe ;
- les variables locales.

Une ***variable d'instance*** est utilisée pour définir les attributs d'un objet.

Une variable de classe définit des attributs pour une classe toute entière d'objets et est appliquée à toutes les instances.

Une variable locale est utilisée à l'intérieur de la définition d'une méthode ou dans un petit bloc d'instructions à l'intérieur de la méthode. Elle peut être utilisée seulement quand la méthode ou le bloc d'instructions est exécuté par l'interpréteur java et après, elle cesse d'exister.

5.2.1 La Déclaration des variables

Avant d'utiliser une variable en java, il faut d'abord la créer en déclarant son nom et le type d'information qu'elle doit stocker. Le type d'informations est d'abord listé, suivit par le nom de la variable. Ci-après sont présentés quelques exemples de déclaration de variable en java

```
int loanlength;
String message;
boolean gameOver;
```

Les variables locales peuvent être déclarées à n'importe quel endroit dans une méthode comme toute instruction java, mais elles doivent être déclarées avant d'être utilisées. La place normale de déclaration des variables se situe immédiatement après l'instruction qui nomme et identifie la méthode. Dans l'exemple qui suit, trois variables sont déclarées dans la méthode main().

```
public static void main(string[] arguments) {
int total;
String reportTitle;
boolean active;
}
```

On peut déclarer plusieurs variables de même type dans la même instruction en séparant les variables par les ",". L'instruction qui suit crée trois variables de type String nommées respectivement `rue, ville, etat.`

```
String rue, ville, etat ;
```

Les variables peuvent être initialisées lors de leur création en utilisant le signe "=" suivi de la valeur de la variable. Par exemple :

```
int zipCode = 02134;
int box = 350;
boolean pbs = true;
String name = "Zoom" , city = "Boston" , state = "MA";
```

Comme l'indique la dernière instruction, on peut assigner des valeurs à des variables de même type en les séparant par des virgules. *Les variables locales doivent être initialisées avant d'être utilisées dans un programme sinon le programme ne se compilera pas correctement*. Pour cette raison, une bonne pratique consiste à déclarer chaque variable locale en l'initialisant.

Les variables d'instances et de classes sont automatiquement initialisées par le compilateur suivant le type de la variable :

- les variables numériques sont initialisées à 0 ;
- les variables Caractères sont initialisées à '0' à compléter ;
- les variables booléennes sont initialisées à false ;
- les variables objets sont initialisées à null.

5.2.2 Les noms de variables

Les noms de variables en java peuvent débuter par une lettre alphabétique, le caractère de soulignement (_), ou le symbole dollar($). Ils ne peuvent pas commencer par un nombre. Après le premier caractère, un nom de variable peut inclure n'importe qu'elle combinaison de lettres et de nombres.

Au moment de nommer ou d'utiliser une variable en java, toujours se rappeler que java est sensible à la casse. La variable nommée `Rose` est différentes de la variable `rose`.

Les noms de variable dans ce cours sont définis en concaténant plusieurs noms significatifs pour faciliter la compréhension des programmes. Pour avoir des noms significatifs les règles suivantes sont appliquées:

- la première lettre de la variable est en minuscule ;
- tous les noms successifs formant la variable débutent par une lettre majuscule ;
- toutes les autres lettres sont en minuscule ;

Les noms de variable suivantes respectent cette règle de nommage :

```
Button loadFile ;
int areaCode ;
boolean quitGame ;
String nomEmploye ;
```

5.2.3 Les types de variables

La déclaration d'une variable doit en principe inclure le type d'information qui doit être stockée. Il peut être l'un des types suivants :

- un type de base ;
- le nom d'une classe ou d'une interface ;
- un tableau.

5.2.3.1 Les types de base

Il existe 8 types de base en JAVA que l'on peut regrouper en quatre : `les entiers, les nombres à virgule flottante, les caractères, les booléens.` Ils sont généralement appelés type primitifs parce qu'ils sont une partie intégrante du langage java ce qui rend leur utilisation plus efficace. Ces types primitifs ont la même taille et les mêmes caractéristiques indépendamment du système d'exploitation utilisé. Il existe 4 types primitifs pour stocker les entiers. Le tableau ci-dessous présente la taille et la valeur maximale et minimale qu'ils peuvent stocker.

Tableau 2.1 Types entiers

Type	Size	Valeurs pouvant être stockées (min à max)
byte	8 bits	-128 à 127
short	16 bits	-32 768 à 32767
int	32 bits	-2 147 493 648 à 2 147 493 647
long	64 bits	-9 223 372 036 854 775 808 à 9 223 372 036 854 775 807

Tous ces types entiers sont signés c'est à dire qu'ils peuvent contenir des nombres positifs et négatifs. Le type utilisé pour une variable dépend de l'intervalle des valeurs que cette variable doit stocker. Aucun de ces types ne peut contenir une valeur plus petite ou plus grande que celle contenue dans le tableau. Il convient dès lors de choisir le type d'une variable en prêtant une attention particulière aux valeurs qu'elle doit contenir ou stoker.

Un autre type de nombre qui peut être utilisé est le nombre à virgule flottante qui peut être de type `float` ou de type double. Les nombres à virgule flottante représentent des nombres avec une partie décimale. Le type `float` est suffisant dans la majorité des cas parce qu'il permet de stocker des nombres allant de 1.4E-45 à 3.4E+38. Pour avoir une précision plus grande, le type double peut être utilisé avec des valeurs de 4.9E-324 à 1.7E+308.

Le type `char` est utilisé pour représenter les caractères individuels tels que les lettres, les chiffres, les ponctuations et les autres symboles. Le dernier des huit types primitifs de java est le type `boolean`. Il ne peut prendre que 2 valeurs `true` ou `false`. Tous ces types primitifs sont en minuscule et doivent être utilisés tel quel en java. Il existe des classes qui portent le nom de certains primitifs mais qui sont différents parce qu'ils sont écrits avec des caractères majuscules au début. Par exemple il y a le type `Boolean` et le type `Char` qui sont des classes et qui ont des fonctionnalités différentes dans un programme java.

5.2.3.2 Les types de données classe

En addition aux 8 types primitifs, une variable peut avoir une classe comme type comme dans les exemples qui suivent :

```
String lastName = "Hopper";
Color hair;
VolcanoRobot vr;
```

Lorsqu'une variable a une classe comme type, la variable fait référence à un objet de cette classe ou à une sous classe de cette classe.

Le dernier exemple dans la liste qui précède `VolcanoRobot vr` crée une variable nommée vr qui réserve une zone mémoire pour l'objet VolcanoRobot. Il est à noter que cet objet n'existe pas encore mais dans la suite, il sera associé à une variable.

Référencer une super classe comme un type de variable est utile lorsque la variable va représenter l'une ou l'autre des sous classes de celle-ci. Par exemple, considérons la hiérarchie des classes avec `CommandButon` comme super classe et trois sous classes: `RadioButton`, `CheckboxButton` et `ClickButton`. Si on crée une variable de type `CommandButton`, alors elle peut être utilisée pour référencer un objet `RadioButton, CheckboxButton, ClickButton`. Ainsi, déclarer une variable de type Object signifie qu'on peut l'associer avec n'importe quel objet.

5.3 Assigner des valeurs a des variables

Après avoir déclaré une variable, une valeur peut lui être affectée en utilisant l'operateur d'affectation qui est le symbole "=". Les exemples ci-dessous présentent quelques instructions d'affectation.

```
idCode = 8675309 ;      accountOverdrawn = false ;
```

5.4 Les constantes

Les variables sont utilisées pour stocker des informations qui peuvent changer au cours de l'exécution d'un programme. Si la valeur d'une variable ne doit jamais changer lors de l'exécution d'un programme, on peut utiliser un type spécial de variable appelé `constante`.

Une constante ou une variable constante est une variable avec une valeur qui ne change jamais. Les constantes sont utiles lorsqu'il faut définir une valeur partagée par toutes les méthodes d'un objet. En d'autres termes, il permet de donner un nom significatif à une valeur qui ne change pas mais étant accessible par toutes les méthodes d'un objet. En JAVA, on peut créer des constantes pour tous les types de variables : d'instances, des classes et locales.

Pour déclarer une constante, utiliser le mot clé **`final`** avant la déclaration de variable et assigner une valeur initiale à cette variable comme dans les exemples ci-dessous

```
final float PI = 3.141592 ;
final boolean DEBUG = false ;
final int PENALITY = 25 ;
```

Il est conseillé d'utiliser des noms de variable en majuscule pour les constantes, ceci pour les distinguer des noms de variable. Les constantes peuvent être utiles pour nommer les différents états d'un objet. Elles apportent une clarté au programme en le rendant plus facile à comprendre. Pour illustrer ce point, considérons les instructions suivantes :

```
final int LEFT = 4;
final int RIGHT = 6;
final int UP = 8;
final int DOWN = e;
this.direction = 4;
this.direction = LEFT
```

5.5 Commentaires

Une des manières les plus importantes pour augmenter la lisibilité d'un programme est d'utiliser les commentaires. Un commentaire est une information incluse dans un programme pour le bénéfice de l'humain qui essayera de lire le programme pour savoir ce qu'il fait. Le compilateur java ignore les commentaires lorsqu'il compile un programme source. Il existe 3 type de commentaires utilisés en java :

❖ la première manière d'ajouter un commentaire à un programme java est de le précéder de 2 caractères "//". Chaque caractère après le *slash* jusqu'à la fin de ligne est considéré comme un commentaire.

```
int credit Hours = 3; // set up credit for course
```

Dans cet exemple, tout ce qui est compris entre // et la fin de ligne est un commentaire et est ignoré par le compilateur java.

❖ La deuxième manière de faire un commentaire qui tient sur plus d'une ligne est de commencer le texte par les caractères "**/*"** et le terminer par ceux-ci "***/**". Tout ce qui se trouve entre ces deux délimiteurs est considéré comme un commentaire.

/* Ce Programme affiche la liste des fichiers et des répertoires qui se trouvent dans le répertoire courant*/

❖ Le denier type de commentaire se veut lisible par les humains et par la machine. En commençant un commentaire avec les caractères "**/****" au lieu de "**/***" et en le terminant par "*/", le commentaire est interprété comme la documentation officielle sur comment les méthodes `public` de la classe fonctionnent. Ce type de commentaire peut être interprété avec les utilitaires tels que ***javadoc*** inclus dans le SDK. Le programme *javadoc* utilise des commentaires officiels pour créer un ensemble de documents html qui documente le programme, sa hiérarchie de classe et ses méthodes. Toute la documentation officielle sur la librairie des classes java est accompagnée de la documentation produite à partir des commentaires à l'aide du programme *javadoc*. On peut consulter la documentation java2 sur le web à l'adresse suivante : *http://java.sun.com/j2se/docs.html*

5.6 Les littéraux

Un littéral est un nombre, un texte ou toute autre information représentée directement par sa valeur.

Dans l'instruction suivante : `int year = 2000 ;` 2000 est le littéral parce qu'il représente directement la valeur 2000. Les nombres, les caractères, les chaines de caractères sont tous des exemples de littéraux.

5.6.1 Les littéraux numériques

Java possède plusieurs types de littéraux entiers. Le nombre 4 par exemple est un littérale de type *int*. On peut aussi le considérer comme un littéral de type *byte* ou *short* parce que chacun de ces types peux bien contenir la valeur 4. Un littéral plus grand qu'un *int* est automatiquement considère comme étant de type *int*.

On peut indiquer qu'un littéral est de type *long* en ajoutant un " L" ou "l" à la fin du nombre. Par exemple l'instruction présentée ci-après traite 4 comme un *entier long*:

```
peineTotal = peinetotal +4L;
```

Pour représenter un nombre négatif comme un littéral, précéder le nombre par le signe "–" comme suit: -45

Pour utiliser un littéral entier en octal, le précéder par un 0 ; par exemple, le nombre entier 777 sera représenté en octal par 0777. Les entiers hexadécimaux sont utilisés comme des littéraux en ajoutant les caractères 0x au début du nombre comme dans 0x12 ou 0x FF.

Les littéraux en virgule flottante utilise le (.) comme séparateur décimale. L'instruction ci-après utilise un littéral pour initialiser une variable de type *double* :

```
double myGPA = 2.25 ;
```

Tous les littéraux en virgule flottante sont considérés comme étant de type doubles ; pour les déclarer comme des float ajouter la lettre F (F ou f) à la fin du littéral.

Exemple : `float piValue=3.1415927F ;`

On peut utiliser la notation en exposant dans la représentation des littéraux en virgule flottante en utilisant les lettres E ou e suivie de la valeur de l'exposant qui peut être un nombre négatif. Les instructions ci- après utilisent la notation exponentielle :

```
double x = 12e22 ;
double y=19E-95 ;
```

5.6.2 Les littéraux de type booléen

Les valeurs booléennes `true` et `false` sont aussi des littéraux. Ces deux valeurs sont les seules à pouvoir être utilisées pour initialiser une variable de type boolean.

```
boolean chosen = true;
```

Il faut remarquer l'absence de quotte autour du littéral *true*.

5.6.3 Les Littéraux de type caractère

Les littéraux caractères sont exprimés par un seul caractère entre quotte. 'a', 'w' , '$', '3'. Il faut se familiariser avec le code ASCII qui incluent 128 caractères constitués de lettres, chiffres, ponctuations, et d'autres caractères utiles en informatique. Java supporte un millier d'autres caractères additionnels à travers son support du standard 16-bit Unicode.

Certains littéraux représentent des caractères qui ne sont ni imprimables ni accessibles à partir du clavier. Le tableau ci-après présente la liste de ces caractères spéciaux.

Tableau 2.2 : Caractère d'échappement

Caractères d'échappement	Signification
\ n	retour de ligne
\ t	Tabulation
\ b	Backspace
\ r	retour chariot
\ f	Formfeed
\\	antislash
\'	quotte
\ "	double quotte
\ d	Octal
\	Hexadécimal
\ud	Caractère Unicode

5.6.4 Les littéraux de type chaîne de caractères

Le dernier littéral utilisé en java représente des chaînes de caractères. Une chaîne de caractères en java est un objet et non un type primitif, ils ne sont pas stockés dans des tableaux comme en C. Les chaines de caractères étant des objets en Java, des méthodes sont disponibles pour concaténer, modifier ou comparer deux chaînes. Un littéral chaîne de caractères est une série de caractères entre double quotte comme présenter ci-après:

```
String quitMsg= "Êêtes vous sûr de vouloir quitter";
String passwd = "MjVf";
```

Les chaînes de caractères peuvent inclure des caractères d'échappement comme ceux cités dans le tableau précédent :

```
String example = "Socrates demande, "; à compléter
System.out.println("Sincèrement, ")
Strings title="Java est un langage très populaire"
```

Quand une chaîne littérale est utilisée, Java la stocke comme un objet de type chaîne de caractères. Il n'y a toutefois pas lieu de créer explicitement un objet comme on le ferai avec d'autres objets, ce qui les rend facile à utiliser comme les autres types primitifs.

5.7 Les expressions et les opérateurs

Une expression est une instruction qui retourne une valeur. La plupart des expressions sont mathématiques comme celles présentées ci-après :

```
int x = 3;
int y = x;
int z = x * y;
```

Toutes ces trois instructions sont des expressions. Elles convertissent des valeurs qui peuvent être affectées à des variables. La première affecte le littéral 3 à la variable x. La seconde affecte la valeur de la variable x à la variable y. l'operateur de multiplication est utilisé pour multiplier la valeur de x et de y et l'expression retourne le résultat de la multiplication. Ce résultat est affecté à la variable z.

Une expression est une combinaison de variables, de littéraux et d'opérateurs. Elles peuvent être des appels à des méthodes, car les méthodes peuvent renvoyer une valeur à l'objet ou la classe qui les a appelées.

La valeur convertie par une expression est appelé *valeur de retour*. Cette valeur peut être affectée à une variable et utilisée de plusieurs autres façons dans un programme java.

La plupart des expressions en Java utilisent les operateurs tel que la multiplication (*).

Les opérateurs sont des symboles spéciaux, utilisés par des fonctions mathématiques, par certains types d'affectations et pour des comparaisons logiques.

5.7.1 Les opérateurs arithmétiques

Il existe cinq opérateurs utilisés pour accomplir les opérations arithmétiques en Java.

Tableau 2.3 : Opérateur arithmétique

Opérateur	Signification	Exemples
+	Addition	3+4
-	Soustraction	5 – 7
*	Multiplication	5 * 5
/	Division	14 / 7
%	Modulo	20 % 7

Chaque opérateur prend deux opérandes. L'opérateur de soustraction peut également être utilisé pour trouver l'opposé d'un nombre. Ce qui revient à le multiplier par "–"

1. Une chose qu'il faut se rappeler c'est le type des nombres lorsqu'on effectue une division. Si le résultat d'une division est affecté dans une variable de type entière, on ne garde que la partie entière car, les entiers ne peuvent pas stocker les nombres en virgule flottante.

Par exemple, l'expression 39 / 9 retourne 3 si elle est affecter à un entier. L'opérateur modulo retourne le reste de la division d'un nombre entier par un autre.

Il est à noter que beaucoup d'opérateurs qui utilisent les entiers retournent des résultats de type *int* à la place du type original des opérandes.

Listing 2.1

```
1: class Weather {
2: public static void main(String[] arguments) {
3: float fah = 86;
4: System.out.println(fah + " degrees Fahrenheit is ...");
5: // To convert Fahrenheit into Celsius
6: // Begin by subtracting 32
7: fah = fah - 32;
8: // Divide the answer by 9
9: fah = fah / 9;
10: // Multiply that answer by 5
11: fah = fah * 5;
12: System.out.println(fah + " degrees Celsius\n");
13:
14: float cel = 33;
15: System.out.println(cel + " degrees Celsius is ...");
16: // To convert Celsius into Fahrenheit
17: // Begin by multiplying it by 9
18: cel = cel * 9;
19: // Divide the answer by 5
20: cel = cel / 5;
21: // Add 32 to the answer
22: cel = cel + 32;
23: System.out.println(cel + " degrees Fahrenheit");
24: }
25: }
```

De la ligne 3 à 12 de ce programme Java, une température en Fahrenheit est convertie en degré Celsius en utilisant les opérateurs arithmétiques.

Ligne 3 : déclaration et initialisation de la variable fah de type `virgule flottante`.

Ligne 4 : affichage de la valeur courante de la variable fah

Ligne 5 : le premier commentaire expliquant ce que fait le programme. Ce commentaire est ignoré par le programme Java.

Ligne 7 : La valeur de fah est diminuée de 32

Ligne 9 : La valeur de fah est divisée par 9.

Ligne 11 : la valeur de fah est multipliée par 9.

Ligne 12 : affichage de la valeur finale de fah convertie en degré celsius.

La même chose est répétée entre les lignes 14 à 23 mais dans le sens contraire. Une température en Celsius est convertie en Fahrenheit.

Ce programme fait aussi usage de l'instruction `system.out.println()` plusieurs fois. La méthode `System.out.println()` est utilisée dans une application pour afficher les chaînes de caractère et d'autres informations sur l'entrée standard, qui est généralement l'écran.

`System.out.println()` prend en argument une chaîne de caractères. Pour utiliser plusieurs arguments il faut les combiner avec l'opérateur + pour obtenir une chaine unique.

5.7.2 L'affection

Affecter une valeur à une variable est une expression, parce qu'elle produit une valeur. Grâce a cette propriété, il est possible dans une même expression d'assigner une valeur à plusieurs variables suivant la syntaxe suivante : `x = y = z=7;`

A la fin de l'exécution de cette expression toutes les variables ont la valeur 7. L'évaluation de l'expression se fait de la droite vers la gauche. Et l'expression qui se trouve à droite de l'opérateur d'affectation est d'abord effectuée :

```
int x = 5 ;          x= x + 2 ;
```

Dans l'expression `x= x + 2`, la première chose qui est faite est l'addition. Le résultat de cette addition qui est la valeur 7 est affecté à la variable `x`.

Modifier le contenu d'une variable à travers des expressions est une pratique courante en programmation. Il existe plusieurs opérateurs utilisés uniquement pour ce but.

Le tableau 3.4 présente ces opérateurs et les expressions qui leurs sont fonctionnement équivalentes:

Tableau 2.4 : Opérateurs d'affectation étendus

Expression	Signification
x += y	x = x + y
x -= y	x = x - y

Expression	Signification
x *= y	x = x * y
x /= y	x = x / y

5.7.3 Opérateur d'incrémentation et de décrémentation

Une opération couramment utilisée est l'ajout ou la soustraction de 1 à une variable. Il existe des opérateurs spéciaux pour cette opération.

L'opérateur d'incrémentation est "`++`" alors que celui de décrémentation est "`--`". Ces opérateurs peuvent être placés avant ou après une variable entière. Le code ci-après présente un exemple :

```
int x=7 ;       x=x++ ;
```

Dans l'expression `x=x++`, x est incrémenté de 7 à 8.

Les opérateurs d'incrémentation et de décrémentation peuvent être placés après une variable. Dans ce cas ils sont appelés respectivement *opérateur de post incrémentation* et *opérateur de post décrémentation*.

Lorsque ces opérateurs sont placés avant la variable, on parle *d'opérateur de pré incrémentation* et *pré décrémentation*.

Considérons l'exemple suivant :

```
int x, y, z ;
x=42 ;
y=x++ ;
z=++x ;
```

Ces deux expressions (`y=x++ ; z=++x ;`) sont très différentes. parce que nous avons l'opérateur de post et de pré incrémentation qui agissent différemment sur les variables. Lors d'une pré incrémentation, la valeur de la variable est modifiée avant l'affectation. Et dans le cas d'une post incrémentation, l'affectation est effectuée avant la modification de la variable. Après l'évaluation des instructions ci-dessus, ***y a pour valeur 42, z vaut 44 et x vaut 44***.

5.7.4 Opérateurs de comparaisons

Java a plusieurs opérateurs de comparaison qui permettent de comparer les variables et des littérales. Ces opérateurs sont utilisés dans des expressions qui retournent des valeurs booléennes (`true` ou `false`)

Tableau 2.5 : Opérateur de comparaison

Opérateur	signification	Exemple
= =	Egal	X
!=	différent	X != 3
<	inférieur	X<3

>	supérieur	X >3
<=	Inferieur ou égal	X <= 3
>=	Supérieur ou égal	X >= 3

L'exemple ci-après montre l'utilisation d'un opérateur de comparaison

```
boolean hip ;
int age = 33 ;
hip = age < 25 ;
```

L'expression age < 25 produit un résultat booléen ; la variable age étant supérieure à 25, il prend la valeur `false`

5.7.5 Les opérateurs logiques

Java possède les opérateurs logiques suivant : AND, OR, XOR, NOT.

L'opérateur AND est représenté par l'opérateur & ou &&. Lorsque deux expressions booléennes sont liées par l'opérateur & ou &&, l'expression booléenne résultante retourne la valeur vraie si les deux expressions booléennes ont simultanément la valeur vraie.

```
bolean unusual = (age<21) & (wifeAge >78) ;
```

Cette expression combine deux expressions age < 21 et wifeAge > 78. Si ces deux expressions sont évaluées à `vraie`, la valeur `true` est affectée à la variable *unusual*. Dans toute autre circonstance, la valeur false est affectée à *unusual* ;

L'opérateur OR est représenté par le symbole "|" ou "||". Lorsque deux expressions booléennes sont liées par l'opérateur "|" ou "||", l'expression booléenne résultante retourne la valeur vraie si au moins une des deux expressions booléennes est évaluée à `vraie`.

L'opérateur XOR est représenté par le Symbole "^". Si deux expressions booléennes sont liées par l'opérateur XOR, l'expression booléenne résultante retourne la valeur `vraie` lorsqu'une et une seule des expressions est évaluée à `vraie`.

L'operateur NOT est représenté par le symbole "!". Il retourne la valeur contraire d'une expression booléenne.

Exemple : si "age <30" retourne la valeur `true`, alors "!(age <30)" retourne la valeur false.

5.7.6 La priorité des opérateurs

Lorsque plus d'un opérateur est utilisé dans une expression, Java établit un ordre de précédence pour déterminer l'ordre dans lequel chaque opérateur sera évalué. Dans beaucoup de cas, la valeur d'une expression dépend de l'ordre de précédence des opérateurs. Par exemple,

considérons l'expression suivante : `y=6+4/2.` La variable y peut recevoir la valeur 5 ou la valeur 8 en fonction de l'opération arithmétique qui sera évaluée en premier lieu. En général, l'ordre de priorité décroissant des opérateurs est le suivant :

- l'incrémentation et la décrémentation ;
- les opérateurs arithmétiques ;
- les opérateurs de comparaison ;
- les opérateurs logiques ;
- les opérateurs d'affectation.

Si deux opérateurs ont la même priorité, l'expression est évaluée de la gauche vers la droite.

Le tableau 2.6 présente les principaux opérateurs de java par ordre de priorité décroissant.

Tableau 2.6 : Priorité des opérateurs

Opérateur	**commentaires**
`. [] ()`	Les parenthèses sont utilisées pour grouper les expressions. L'opérateur est utilisé pour accéder aux méthodes et aux variables à l'intérieur d'une classe. les crochets sont utilisés pour les tableaux.
`++ -- ! – instanceof`	L'opérateur *instanceof* retourne la valeur `true` ou `false` si l'objet est une instance de la classe nommée comme paramètre.
`New (type) expression`	L'opérateur New est utilisé pour créer des nouvelles instances de classe.
* / %	Multiplication, division, modulo
+ -	Addition, soustraction
<< >> >>>	Décalage à gauche, décalage à droite
< > <= >=	Comparaisons
== !=	Egalité, différence
&	ET
^	XOR
|	OU
&&	ET logique
||	OU logique
? :	Abbreviation de *if… then…else…*
= += -= *= /= %= ^=	Affectations étendues
&= |= <<= >>= >>>=	Affectations étendues

Revenons à l'expression y=6+4/2. Le tableau 2.6 montre que la division est plus prioritaire que l'addition, ainsi la valeur de y de l'exemple précédent sera égale à 8.

Pour changer l'ordre dans lequel les expressions doivent être évaluées, placer des parenthèses autour de ces expressions pour qu'elles soient évaluées en premier. Par exemple, l'expression y=

(6+4) /2 retourne la valeur 5 parce que 6+4 est d'abord évalué puis le résultat 10 est divisé par 2. Les parenthèses sont aussi utilisées pour augmenter la lisibilité des expressions. Si la priorité d'une expression n'est pas immédiatement claire, l'ajout des parenthèses pour imposer un ordre d'évaluation peut rendre cette instruction plus facile à comprendre.

5.7.7 Arithmétique des chaînes de caractères

L'opérateur + est utilisé pour concaténer des chaînes de caractères. Concaténer signifie mettre deux chaînes côte à côte. Dans plusieurs exemples vous allez voir des instructions qui ressemblent à :

```
String prenom = "Raymond";
System.out println("Tout le monde aime "+prenom) ;
```

Ces deux instructions provoquent l'affichage suivant :

Tout le monde aime Raymond

L'opérateur + combine les chaînes de caractères, les objets et les variables pour former une chaîne simple. Travailler avec l'opérateur de concaténation est facile en Java parce qu'il permet de manipuler les variables de n'importe qu'elle type comme des chaînes de caractère. Si une partie d'une opération de concaténation contient des chaînes de caractère ou des chaînes constantes, tous les éléments de cette expression sont traités comme des chaînes de caractères :

```
System.out.println(4 + " Score et "+7+" ans plus tard.")
```

Cette instruction produit la sortie suivante à l'écran :

4 Score et 7 ans plus tard

Il existe un opérateur d'affectation étendu noté `+=` qui permet d'ajouter quelque chose à la fin d'une chaine. Par exemple considérons les instructions suivantes :

```
String monNom = "Efrem Zimbalist";
monNom += " Jr. " ;
```

Cette instruction est équivalente à : `monNom +=monNom+" Jr." ;` Cet exemple change la valeur de la variable "monNom" en ajoutant Jr. a la fin de *Efrem Zimbalist*.

CHAPITRE 6 TABLEAU, INSTRUCTION DE CONTRÔLE ET BOUCLE LES

6.1 Tableaux

Jusqu'ici, les variables sont utilisées de manière individuelle pour stocker les données. Pour enregistrer un ensemble d'informations, il faut faire appel à une nouvelle structure de données appelée tableau. Un tableau permet d'enregistrer un ensemble de données ayant le même type de données ou appartenant à la même classe. Les éléments du tableau sont stockés de manière contiguë, ce qui facilite l'accès aux données. A chaque élément du tableau, est associé un indice qui indique son rang dans le tableau. Cet indice comme généralement par « 1 », mais il y a des environnement dans lesquels il commence par « 0 ». Un tableau peut contenir n'importe quel type de données mais lorsque qu'il est créé, le type de l'information qu'il contient ne peut plus changer. Par exemple, on peut avoir un tableau d'entiers, un tableau de chaînes de caractères, un tableau de tableaux. Mais, on ne peut avoir un tableau qui contient à la fois des chaînes de caractères et des entiers. Java implémente des tableaux d'une manière différente des autres langages de programmation. En java, un tableau est un objet qui a ses attributs et ses méthodes. Pour créer un tableau en java, il faut :

- Déclarer une variable qui sera le nom du tableau
- Créer un nouvel objet tableau qui sera affecté à la variable tableau
- Stocker les informations dans le tableau

6.1.1 Déclaration des variables de type tableau

La première étape dans la création d'un tableau consiste à définir le nom de la variable tableau. La variable tableau indique le type ou la classe des éléments que le tableau peut contenir. Pour différencier le nom d'un tableau des autres noms de variables, on leur ajoute des crochets. Les instructions ci-dessous sont des exemples de déclaration des tableaux :

```
String[ ] request;  Point[ ] targets;   float[ ] donations;
```

On peut aussi déclarer un tableau en mettant les crochets sur les noms de variables au lieu des types

```
String request[ ];  Point targets[ ];   float donations[ ];
```

6.1.2 Création des objets tableau

Après avoir déclaré une variable tableau, la prochaine étape est de créer un objet qui sera assigné à la variable tableau. Pour le faire, on utilise l'opérateur New et ensuite on initialise le contenu du tableau directement. Comme les tableaux sont des objets en java, on peut utiliser l'opérateur new pour créer une nouvelle instance d'un tableau comme indiquer ci-dessous :

```
String[ ] players = new String[ 10 ];
```

Cette instruction crée un tableau de dix chaînes de caractères. Lorsqu'on crée un tableau en utilisant l'opérateur new, on doit indiquer le nombre d'éléments que doit contenir le tableau :

```
int[ ] temps = new int[ 99];
```

En créant un tableau avec l'instruction `new`, tous les éléments sont automatiquement initialisés (`0` pour le tableaux de type numérique, `false` pour les booléens, `"\0"` pour les tableaux de caractères, la valeur `null` pour les objets). Il est tout aussi possible de créer un tableau en l'initialisant comme dans l'exemple ci-dessous :

```
Point[ ] markup = { new Point(1,5), new Point(3,3), new
Point(2.3)};
```

Tous les éléments dans le tableau doivent être du même type. Lorsqu'un tableau est créé de la sorte, la taille du tableau est égale aux nombres d'éléments entre accolade. L'exemple précédent crée un tableau d'objets Point nommé `markup` qui contient trois éléments. Comme les chaînes de caractères peuvent être créées sans l'utilisation de l'opérateur `new`, on peut procéder de la même manière avec les tableaux de chaînes.

```
String[ ] titles = {"Mr.", "Mrs.", "Ms.", "Miss", "Dr."};
```

L'instruction ci-dessus déclare un tableau d'objet chaîne de caractères nommé `titles`.

6.1.3 Accéder aux éléments d'un tableau

Après l'initialisation d'un tableau, on peut accéder aux valeurs stockées, changer, ou tester chaque élément du tableau. On accède aux valeurs d'un tableau en utilisant son nom suivie de l'indice du tableau entre crochet:

```
testScore[40]=920;
```

Cette instruction affecte la valeur 920 au quarantième élément du tableau. Il et important de retenir qu'en Java, le premier élément du tableau commence à l'indice `0`. Par exemple pour un tableau de 12 éléments les indices varie de 0 à 11.

Considérons les instructions suivantes :

```
float[ ] rating = new float[ 20];
rating[ 20]=3.22F;
```

Un programme avec ces deux lignes de code produit une erreur de compilation, car l'élément d'indice 20 n'existe pas. La variable d'instance `length` de la classe `array` retourne le nombre d'éléments du tableau. Par exemple l'instruction

```
System.out.println("Elements: " + rating.length);
```

imprime le nombre d'éléments contenu dans le tableau *rating*.

6.1.4 Changer les valeurs des éléments d'un tableau

On peut utiliser l'opérateur d'affectation pour modifier la valeur d'un élément du tableau :

```
temperature[4]=85;
day[0]="Sunday";
manager[2]=manager[0];
```

Avec Java, les tableaux sont des structures de données faciles à créer et à modifier. Par rapport à d'autre langage de programmation, java offre plus de fonctionnalité pour la manipulation des tableaux. L'exemple ci-dessous présente quelque unes de ces fonctionnalités.

L'application `CentFrancs` utilise trois tableaux d'entiers pour représenter le nombre de pièces de cent francs frappées à Yaoundé et à Abidjan entre les années 1993 et 1995.

Listing 3.1

```
CentFrancs.java
1: class CentFrancs {
2: public static void main(String[] arguments) {
3: int[] Yaounde = { 15000006, 18810000, 20752110 };
4: int[] Abidjan = new int[Yaounde.length];
5: int[] total = new int[Yaounde.length];
6: int average;
7:
8: Abidjan[0] = 15020000;
9: Abidjan[1] = 18708000;
10:Abidjan[2] = 21348000;
11:
12: total[0] = Yaounde[0] + Abidjan[0];
13: total[1] = Yaounde[1] + Abidjan[1];
14: total[2] = Yaounde[2] + Abidjan[2];
15: average = (total[0] + total[1] + total[2]) / 3;
16:
17: System.out.println("1993 production: " + total[0]);
18: System.out.println("1994 production: " + total[1]);
19: System.out.println("1995 production: " + total[2]);
20: System.out.println("Production moyenne: "+ average);
```

```
21: }
22: }
```

Ce programme produit les sorties suivantes:

```
1993 production: 3002006
1994 production: 37518000
1995 production: 42100110
Production moyenne : 36546038
```

La classe créée ici a trois variables d'instance de type tableau d'entiers. Le premier tableau nommé `Yaounde`, est déclaré et initialisé à la ligne 3, et contient trois entiers. Le second et le troisième tableau sont déclarés entre les lignes 4 et 5. `Abidjan` contient la production des pièces de Cent francs de la ville d'Abidjan. `Total` contient la somme des productions de `Yaounde` et d'Abidjan. Aucune valeur initiale n'est spécifiée pour les éléments des tableaux `Abidjan` et `Total`. Pour cette raison les éléments de ces tableaux sont initialisés à 0 par le compilateur. La variable `Yaounde.length` est utilisée pour définir les tableaux `Abidjan` et `Total` avec le même nombre d'éléments que `Yaounde.` Les autres instructions de la méthode `main` effectuent les actions suivantes :

Ligne 6: création d'une variable de type entière nommée `average`

Ligne 8 à 10 : initialisation des éléments du tableau `Abidjan`

Ligne 12 à 14 initialisation du tableau `Total`.

Ligne 15 calcule de la moyenne de la production avec le tableau

Ligne 17 à 20 affiche le total par année et la production moyenne.

Si l'application `CentFrancs` avait des tableaux avec 100 éléments, alors le programme serait plein de code `répétitif`. Généralement pour le parcours des tableaux, une structure `itérative` est utilisée, ce qui raccourci la longueur du code.

6.1.5 Tableaux multidimensionnels

Java ne supporte pas les tableaux multidimensionnels, mais on peut obtenir un tableau multidimensionnel en déclarant un tableau de tableau. Par exemple considérons un programme qui voudrait enregistrer un entier pour chaque jour de l'année et organiser ces jours en semaine. Une manière d'organiser ces données est d'avoir un tableau de 52 éléments. Chaque élément de ce tableau sera un tableau 7 éléments.

```
int[ ][ ] dayvalue = new int[52] [7];
```

Ce tableau de tableaux contient au total 364 entiers, chaque cellule représentant un jour de l'année. On peut fixer la valeur du premier jour de la dixième semaine grâce à l'expression suivante :

```
dayValue[10] [1]=14200;
```

La variable `length` peut aussi être utilisée comme avec les autres tableaux.

6.2 Les blocs d'instructions

Les instructions en java sont regroupées en blocs. Le début d'un bloc est représenté par la parenthèse ouvrante, la parenthèse fermante représentant la fin du bloc. Un bloc peut contenir des variables et des méthodes dans la définition d'une classe. Il peut aussi être utilisé pour définir les instructions d'une méthode. Une notion importante que l'on doit noter à propos des blocs d'instructions est qu'ils créent une portée pour les variables locales qui sont créées dans ce bloc. La portée d'une variable est la partie d'un programme ou une variable existe et peut être utilisée. Par exemple, la méthode `testBlock` contient un bloc.

```
void testBlock() {
int x=10;
{ //debut du block
int y=40;
y=y+x
//end of block
}
}
```

Il y a deux variables qui sont définies dans cette méthode : `x` et `y`. La portée de la variable `y` est le bloc dans lequel elle est définie. Elle ne peut être utilisée que dans ce bloc. La variable `x` est créée à l'intérieur de la méthode mais en dehors du bloc interne. Ainsi elle peut être utilisée n' importe où dans la méthode. On peut modifier la valeur de `x` n' importe où dans la méthode et cette valeur sera retenue.

6.3 Structure de contrôle « if »

L'un des aspects clés de la programmation est la capacité à décider de ce qui sera fait. Ceci grâce aux instructions conditionnelles. Une instruction conditionnelle est une instruction qui est exécutée seulement si une condition spécifique est vrai. L'instruction conditionnelle basique est la structure if. La structure if utilise une expression booléenne pour décider si une instruction doit être exécutée. Si l'expression retourne la valeur vraie, l'expression est exécutée.

```
if(age>39)
System.out.println("You call that a haircut")
```

Pour exécuter une instruction lorsque l'expression booléenne est évaluée à *Faux*, le mot clé `else` peut être utilisé.

```
if(blindDateAttractive == true)
restaurant = "Benihana 's";
else
restaurant = "Burritos-to-go";
```

6.4 La structure conditionnelle switch

Une pratique couramment utilisée dans les langages de programmation consiste à tester une variable par apport à des valeurs et en fonction du résultat exécuter des actions. L'instruction `switch` est basée sur des tests.

```
switch(grade) {
case 'A' :
  System.out.println("Great job");
  Break ;
case 'B' :
  System.out.println("Good job");
  Break ;
case 'C' :
  System.out.println("You can do better");
  Break ;
default :
  System.out.println("Considering cheating");  }
```

L'implémentation Java de `switch` est limitée. Les tests et les valeurs ne peuvent être que des types de bases (`byte, char, short, ou int`). La variable de test `grade` est comparée successivement avec les valeurs `A, B` et `C.` Dans le cas où `grade` ne contient aucun des trois caractères `A, B, C,` l'instruction `default` est exécutée.

```
switch (operation) {
case '+' :
  add(object1, object2);
  break;
case '-' :
  subtract(object1, object2);
  break;
case '*' :
  multiply(object1, object2);
  break;
case '/' :
  divide(object1, object2);
  break;}
```

L'instruction break permet d'arrêter prématurément l'exécution de la boucle

`switch` lorsque la valeur de test est trouvée.

Listing 3.2

```
DayCounter.java
1: class DayCounter {
2: public static void main(String[] arguments) {
3: int yearIn = 2001;
4: int monthIn = 2;
5: if (arguments.length > 0)
6: monthIn = Integer.parseInt(arguments[0]);
7: if (arguments.length > 1)
8: yearIn = Integer.parseInt(arguments[1]);
9: System.out.println(monthIn + "/" + yearIn + " has "
10: + countDays(monthIn, yearIn) + " days.");
11: }
12:
13: static int countDays(int month, int year) {
14: int count = -1;
15: switch (month) {
16: case 1:
17: case 3:
18: case 5:
19: case 7:
20: case 8:
21: case 10:
22: case 12:
23: count = 31;
24: break;
25: case 4:
26: case 6:
27: case 9:
28: case 11:
29: break;
31: case 2:
32: if (year % 4 == 0)
33: count = 29;
34: else
35: count = 28;
36: if ((year % 100 == 0) & (year % 400 != 0))
37: count = 28;
38: }
39: return count;
40: }
41: }
```

Cette application utilise des arguments passés en ligne de commande (mois, année), puis affiche le nombre de jour du mois dans cette année. Par exemple pour les années bissextiles, le mois de février comporte 29 jours au lieu de 28.

Après compilation du programme, la commande ci-après permet de connaître le nombre de jours au mois de février 2000.

```
java DayCounter 2 2000
```

Le programme affiche le résultat suivant :

Figure 3.1. Résultat du programme « DayCounter »

```
2/2000 has 29 days.
```

6.5 La boucle « for »

La boucle « for » est utilisée pour répéter une instruction jusqu'à ce qu'une condition soit rencontrée. On fait usage de la boucle for lorsque le nombre d'itérations est connu à l'avance. La boucle for en java a la syntaxe suivante :

```
for (initialisation; test; increment) {
instruction;
}
```

Le début de la boucle `for` a trois parties:

- `initialisation`: c'est une expression qui initialise la variable de boucle. Cette expression peut déclarer la variable de boucle. Dans ce cas, la variable de boucle est locale à la boucle.
- `test:` c'est le test qui est d'abord effectué avant chaque parcours de la boucle. `Test` doit être une expression booléenne ou une fonction qui retourne une expression booléenne. Si le test est évalué à *vrai*, l'instruction de boucle est exécutée. Sinon, la boucle est arrêtée et l'instruction suivant la boucle est exécutée.
- `l'incrément:` c'est n'importe quelle expression ou appel de fonction. Dans la plupart des cas, l'incrément permet de modifier la variable de boucle pour que l'expression `test` à un moment soit évaluée à faux.

`Instruction` est la partie de la boucle qui est exécutée à chaque itération. Comme avec la structure de contrôle « `if` », `instruction` peut être une *instruction unique* ou un *bloc d'instructions*. L'exemple ci-dessous montre un exemple d'utilisation de la boucle `for`.

```
String [] salutation = new String[10] ;
int i ; //la variable de boucle
for(i=0 ; i < salutation.length ; i++)
salutation[i] = "Mr."
```

Dans cet exemple, la variable `i` joue le rôle de variable de boucle. Il compte le nombre de fois que la boucle est exécutée. Avant chaque parcours de la boucle, la variable de la boucle est comparée à `salutation.length`, qui représente le nombre d'éléments dans le tableau salutation. Quand la variable de boucle est supérieure ou égale à `salutation.length`, l'exécution de la boucle est arrêtée. `i++` est l'élément final de la boucle. Il provoque l'incrémentation de la variable de boucle à chaque itération. Sans cette instruction, la boucle ne pourra pas s'arrêtée (boucle infini).

Listing 3.3

```
HalfLoop.java
1: class HalfLoop {
2: public static void main(String[] arguments) {
3: int[] Yaounde = { 15000006, 18810000, 20752110 };
4: int[] Abidjan = { 15020000, 18708000, 21348000
};
5: int[] total = new int[Yaounde.length];
6: int sum = 0;
8: for (int i = 0; i < Yaounde.length; i++) {
9: total[i] = Yaounde[i] + Abidjan[i];
10: System.out.println((i + 1993) + " production: "
11: + total[i]);
12: sum += total[i];
13: }
15: System.out.println("Average production: "
16: }
18: }
```

Ligne 8: la boucle est créée avec une variable entière nommée i. La variable de boucle est incrémentée de 1 à chaque itération. La boucle s'arrête quand `i` est égale ou supérieur à `Yaounde.length`, le nombre total d'éléments du tableau `Yaounde`.

Ligne 9 – 11: la valeur total d'un élément du tableau total est calculée en utilisant la variable de boucle `i` et le résultat est afficher.

Ligne 12: les valeurs totales sont additionnées dans la variable `sum` pour calculer la moyenne plus tard.

6.6 Les boucles « while » et « do loop »

Comme avec la boucle `for`, les boucles `while` et « `do loop` », permettent à un bloc de code Java de se répéter jusqu'à ce qu'une condition soit rencontrée. Ces boucles sont généralement utilisées lorsque le nombre d'itérations n'est pas connu à l'avance.

6.6.1 La boucle « while »

La boucle `while` permet de répéter une instruction tant que la condition de boucle est *vraie*. L'exemple suivant est un exemple de boucle `while` :

```
while(i < 0 ) {
x = x * i++; //le corp de la boucle
}
```

La condition qui accompagne le mot clé `while` est une expression booléenne (i<10). Si l'expression retourne vraie la boucle `while` exécute le corps de la boucle et teste la condition encore. Ce processus est répété jusqu'à ce que la condition devienne fausse. Le listing 3.4 montre un exemple de boucle `while` qui permet de copier un tableau d'entiers en tableau de réels.

Listing 3.4

```
CopyArrayWhile.java
1: class CopyArrayWhile {
2: public static void main(String[] arguments) {
3: int[] array1 = { 7, 4, 8, 1, 4, 1, 4 };
4: float[] array2 = new float[array1.length];
6: System.out.print("array1: [ ");
7: for (int i = 0; i < array1.length; i++) {
8:   System.out.print(array1[i] + " ");
9: }
10: System.out.println("]");
12: System.out.print("array2: [ ");
13: int count = 0;
14: while ( count < array1.length && array1[count] != 1) {
15:   array2[count] = (float) array1[count];
16:   System.out.print(array2[count++] + " ");
17: }
18: System.out.println("]");
19: }
20: }
```

Le programme affiche les résultats suivants:

```
array1: [ 7 4 8 1 4 1 4]
array2: [7.0 4.0 8.0 1.0 4.0 1.0 4.0]
```

Les lignes 3 et 4 déclarent les tableaux ;

Lignes 6 et 10 : Affichage des résultats ;

Ligne 13 et 17: Ce bout de code copie le contenu du tableau array1 dans le tableau array2 en excluant les cellules contenant la valeur 1. Il faut aussi noter que la valeur entière de array1 sont converties en virgule flottante lors de la copie.

6.6.2 La boucle « Do . . Loop »

La boucle do LOOP a le même fonctionnement que la boucle `while,` à la seule différence que la condition de boucle est la dernière instruction de la boucle.

L'avantage ici est que le corps de la boucle est exécuter au moins une fois avant le test de la condition de boucle.

```
long i=1;
do {
i *= 2;
System.out.print(i+ " ");
} while (i < 3000000000000L);
```

L'exemple précédent présente un bout de code qui utilise la boucle `do loop` pour doubler la valeur de i jusqu'à ce que i soit supérieur ou égale à trois milliards.

6.7 Les instructions break, et continue

Le mot clé BREAK, lorsqu'il est utilisé dans une boucle provoque la sortie immédiate de la boucle.

```
int count = 0;
while (count <array1.length) {
if (array[ count] == 1)
break
array2[ count]= (float) array1[ count];
}
```

Le mot clé `continue` interrompt le passage de la boucle en cours, mais ne quitte pas la boucle comme break.

```
int count = 0;
int count2 = 0;
while (count++ <= array1.length) {
if (array1[ count]== 1)
continue;
array2[ count2++]; = (float)array1[ count];
}
```

L'instruction *continue* est utilisée pour passer à l'itération suivante lorsque l'élément courant du tableau array1 est égal à 1.

6.8 L'opérateur conditionnel

Une alternative à l'utilisation de `if else` est l'opérateur conditionnel parfois appelé *opérateur ternaire* a cause de ses trois opérandes. L'opérateur conditionnel est une expression qui retourne une valeur

```
test ? trueresult : falseresult;
```

`test` est une expression logique qui retoune `true` ou `false`. Si `test` est évaluée à *vrai*, l'opérateur conditionnel retourne `trueresult`. Sinon, il retourne `falseresult`.

```
int a,b,max;
max = a > b ? a : b;
```

L'expression conditionnelle précédente retourne le maximum de a et b.

CHAPITRE 7 UTILISER LES CLASSES EN JAVA

Java utilise les objets pour accomplir les taches. Dans ce chapitre, les objets sont intensément abordés. Les points suivants seront abordés :

- la création des objets (instances) ;
- tester et modifier les classes et les variables d'instance dans ces objets ;
- les appels de méthode d'objet ;
- conversion d'objet et des types d'objet d'une classe à une autre.

7.1 La création d'objet

La programmation Java s'appuie le plus souvent sur la définition d'un ensemble de classes. Les classes sont des moules qui permettent de créer des objets. Ces objets, qui sont aussi appelés instances encapsulent des données et des programmes.

Généralement on utilise des classes pour créer des objets et ensuite on travaille avec eux. Dans cette section est abordée la création d'un nouvel objet à partir de n'importe quelle classe.

7.1.1 L'opérateur « new »

Pour créer un nouvel objet, l'opérateur `new` est utilisé suivie du nom de la classe dont ont veut créer une instance, terminé par des parenthèses :

```
String name = new String() ;
URL adresse = new URL(http://www.prerfect.com);
VolcanoRobot robbie = new VolcanoRobot();
```

Les parenthèses sont importantes; ne jamais les omettre. Les parenthèses peuvent être vides, auquel cas un objet de base est crée ; ou les parenthèses peuvent contenir des arguments qui déterminent les valeurs initiales des variables d'instances.

Les exemples ci-après présentent des objets crées avec des arguments :

```
Random seed = new Random(6068430714) ;
Point pt = new Point(0,0) ;
```

Le nombre et le type des arguments que l'on peut utiliser dans les parenthèses avec l'opérateur new sont définis par la classe elle-même en utilisant une méthode spécial appelez constructeur. Lorsqu'on essaye de créer une nouvelle instance de classe avec un nombre incorrecte d'argument ou de type d'argument, une erreur apparait à la compilation du programme.

Voici un exemple de création d'objet en utilisant un nombre et type d'arguments différents : La classe `StringTokenizer`, qui est une partie dupackage `Java.util` divise une chaîne de caractère en une série de sous chaînes courtes appelées `tokens`. Une chaîne est divisée en `tokens` en utilisant un caractère comme délimiteur de sous chaîne. Par exemple le texte "`02/20/67`" peut être séparé en trois `tokens` : 02, 20, et 67, en utilisant le slash(/) comme délimiteur.

Le Programme ci-dessous créer un objet `StringTokenizer` en utilisant l'opérateur `new` de deux manières différentes et affiche les `tokens` que contient chaque objet.

Listing 4.1 : ShowTokens.java

```
1: import java.util.StringTokenizer;
3: class ShowTokens {
5: public static void main(String[] arguments) {
6: StringTokenizer st1, st2;
8: String quote1 = "VIZY 3 -1/16";
9: st1 = new StringTokenizer(quote1);
10: System.out.println("Token 1: " + st1.nextToken());
11: System.out.println("Token 2: " + st1.nextToken());
12: System.out.println("Token 3: " + st1.nextToken());
14: String quote2 = "NPLI@9 27/32@3/32";
15: st2 = new StringTokenizer(quote2, "@");
16: System.out.println("\nToken 1: " + st2.nextToken());
17: System.out.println("Token 2: " + st2.nextToken());
18: System.out.println("Token 3: " + st2.nextToken());
19: }
20:
```

Lorsque vous compiler et exécuter ce programme, les sorties ressemblent à :

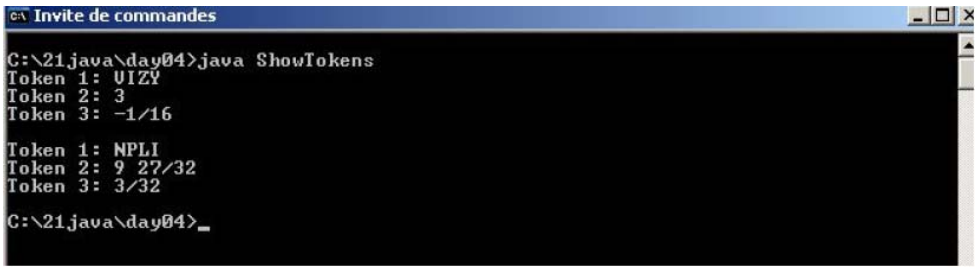


Figure 4.1. Exécution du programme en lignes de commandes

Dans cette exemple deux objets différents `StringTokenizer` sont crées en utilisant deux arguments différents pour le constructeur se trouvant après le `new`.

La première instance (ligne 9) utilise `new StringTokenizer()` avec un seul argument nommé quote1. Cette instruction créer un objet `StringTokenizer` qui utilise le délimiteur par défaut : l'espace, la tabulation, le retour de ligne, le retour chariot. Si un de ces caractères apparaît dans la chaîne, il est utilisé pour séparer les `tokens`. La chaîne `quote1` contient des espaces, ce caractère est utilisé comme séparateur pour diviser les `tokens`. La ligne 10 à 12 affiche la valeur des trois `tokens : VIZY, 3`, et `-1/16.`

Le second objet `StringTokenizer` dans cet exemple possède deux arguments lorsqu'il est construit à la ligne 14 : La chaîne `quote2` et le caractère @. Ce second argument indique que le caractère @ doit être utilisé comme délimiteur entre les différents `tokens`. L'objet `StringTokenizer` créé à la ligne 15 contient 3 `tokens : NPLI, 9 27/32` et `3/32`.

7.1.2 L'operateur new

Lors de l'utilisation de l'opérateur new, plusieurs choses sont faites :

- la nouvelle instance de la classe est créée,
- une partie de la mémoire lui est réservée ;
- une méthode spéciale contenue dans la classe est appelée. Cette méthode particulière est appelée constructeur.

Les constructeurs sont des méthodes spéciales qui permettent de créer et d'initialiser les nouvelles instances d'une classe. Le constructeur initialise le nouvel objet et ses variables, crée tout autre objet dont l'objet à besoin, et exécute toutes les autres opérations nécessaires à l'initialisation de ce dernier. Plusieurs constructeurs peuvent être définis dans la classe avec un nombre d'arguments différents. L'usage de l'opérateur `new`, indique les arguments adaptés et appelle le constructeur approprié est appelé.

7.1.3 La gestion de la mémoire

La gestion de la mémoire en Java est dynamique et automatique. Lorsqu'un nouvel objet est créé, java alloue automatiquement l'espace mémoire dont il a besoin. On n'a pas besoin d'allouer explicitement la mémoire pour les objets. Java le fait pour vous. La gestion de la mémoire en java étant automatique, la fonction de désallocation de la mémoire n'est plus nécessaire. Dans beaucoup de circonstances, à la fin d'utilisation d'un objet créé, java sera capable de déterminer

que l'objet ne sera plus utilisé et le supprimera automatiquement. Lorsqu'un programme est en cours d'exécution, périodiquement, java recherche les objets non utilisés et libère l'espace qu'ils occupaient en mémoire. Le processus responsable de cette opération est appelé *ramasse miette* (garbage collection) et il est entièrement automatique.

7.1.4 Accès aux variables d'instance

7.1.4.1 Accès aux valeurs

L'accès à la valeur d'une variable d'instance se via la notation ".". Avec cette notation, le nom d'une variable de classe ou d'instance a deux parties : à la gauche du point, on a la référence à l'objet ou la classe et à la droite du point, on a le nom de la variable. C'est une manière de faire référence à une variable d'instance ou de classe ou à une méthode. Par exemple, pour l'objet nommé `myCustomer` possédant une variable appelée `orderTotal`, on fait référence à cette variable de la manière suivante : `myCustomer.orderTotal`. Cette manière d'accéder aux variables est une expression et chaque élément à la droite ou la gauche du point est aussi une expression. L'expression à la droite du point peut aussi être un objet qui possède aussi ses variables d'instance. Par exemple dans notre cas, `orderTotal` peut avoir un attribut nommé `layaway`. On utilise la notation ci-après pour accéder à l'attribut `layaway` : `myCustomer.orderTotal.layaway`. Les expression possédant la notation "." sont évaluées de la gauche vers la droite.

7.1.4.2 Changer la valeur d'une variable d'instance

Changer la valeur dune variable d'instance est une chose relativement facile. Il suffit 'utiliser l'opérateur d'affectation et d'indiquer par la suite à la droite du symbole d'affectation la valeur à assigner.

```
myCustomer.orderTotal.layaway=true ;
```

Cet exemple affecte la valeur `true` à la variable d'instance `layaway.` Le Listing 4.2 est un exemple de programme qui test et modifie les variables d'instance de l'objet `Point`. `Point` est inclut dans le package `java.awt` et utilise le système de coordonnées `x, y`.

Listing 4.2 code source de SetPoint.java

```
1: import java.awt.Point;
3: class SetPoints {
5: public static void main(String[] arguments) {
```

```
6: Point location = new Point(4, 13);
8: System.out.println("Starting location:");
9: System.out.println("X equals " + location.x);
10: System.out.println("Y equals " + location.y);
12: System.out.println("\nMoving to (7, 6)");
13: location.x = 7;
14: location.y = 6;
16: System.out.println("\nEnding location:");
17: System.out.println("X equals " + location.x);
18: System.out.println("Y equals " + location.y);
19: }
20: }
```

7.1.5 Les variables de classe

Les variables de classe sont des variables qui sont définies et stockées dans la classe elle-même. Leur valeur s'applique à la classe et à toutes ces instances. On définit une variable de classe en incluant le mot clé statique avant le type de la variable au moment où on la déclare. Par exemple, considérons la définition partielle de la classe ci-dessous :

```
class FamilyMember{
static String surname = "Mendoza";
String name;
int age;
}
```

Les instances de la classe `FamilyMember` ont chacune des valeurs différentes pour les attributs `name` et `age` mais la variable de classe `surname` a une seule valeur identique pour toutes les instances de la classe `FamilyMember`.

Pour accéder à une variable de classe, on utilise toujours la notation ".". Pour accéder ou modifier la valeur de la variable de classe, vous pouvez utiliser le nom de la classe à la gauche de l'opérateur ".", puis le nom de l'attribut.

```
FamilyMember dad = new FamilyMember() ;
System.out.println("Family's surname is : " + dad.surname) ;
System.out.println("Family's surname is:" +
FamilyMember;surname);
```

Rappelez vous que la valeur d'une variable de classe affecte toutes ces instances. Pour cette raison, il est bon de commencer le nom d'une variable d'instance par le nom de classe. Ainsi votre code est plus lisible et facile à déboguer.

7.2 Appel de méthode

Appeler une méthode dans un objet est similaire a l'usage d'une variable d'instance : l'opérateur point est utilisé. L'objet donc vous appeler la méthode se trouve à gauche de l'opérateur point et le nom de la méthode et ses arguments se trouvent à la droite :

```
mycustomer.addToOrder(itemNumber, price, quantity) ;
```

Il est à noter que toutes les méthodes doivent avoir des parenthèses même si elles n'ont pas d'argument :

```
myCustomer.cancelAllOrder() ;
```

Le Listing 4.3 montre en exemple l'appel de quelques méthode défini dans la class `String`. La classe String inclut des méthodes qui permettent de tester et de modifier des chaînes de caractère.

Listing 4.3 CheckString.java

```
1: class CheckString {
3: public static void main(String[] arguments) {
4: String str = "Nobody ever went broke by buying IBM";
5: System.out.println("The string is: " + str);
6: System.out.println("Length of this string: "
7: + str.length());
8: System.out.println("The character at position 5: "
9: + str.charAt(5));
10: System.out.println("The substring from 26 to 32: "
11: + str.substring(26, 32));
12: System.out.println("The index of the character v: "
13: + str.indexOf('v'));
14: System.out.println("The index of the beginning of the "
15: + "substring \"IBM\": " + str.indexOf("IBM"));
16: System.out.println("The string in upper case: "
17: + str.toUpperCase());
18: }
19: }
```

Les messages ci-dessous s'affiche sur votre écran lorsque vous exécuter le programme :

```
The string is: Nobody ever went broke by buying IBM
Length of this string: 36
The character at position 5: y
The substring from 26 to 32: buying
The index of the character v: 8
The index of the beginning of the substring "IBM": 33
The string in upper case: NOBODY EVER WENT BROKE BY BUYING
IBM
```

A la ligne 4, une nouvelle instance de la classe String est créée en utilisant une chaîne de caractère. Dans le reste du programme, quelques méthodes de la classe String sont appelées pour illustrer leur utilisation.

Ligne 5 : cette ligne imprime la valeur de la chaîne créée à la ligne 5.

Ligne 7 : La méthode `length()` est utilisée pour calculer la longueur de la chaîne.

Ligne 9 : Un appel à la méthode `charAt()` est fait. Cette méthode retourne le caractère qui se trouve à la position indique par son argument. Dans notre cas `charAt(5)` retourne y. Il faut remarquer que les positions des chaînes commencent à 0 au lieu de 1.

Ligne 11 : La méthode `substring()`, qui prend deux arguments de type entier indiquant un intervalle, retourne la sous chaîne dont les positions sont indiquées par l'intervalle. `substring` peut être appelé avec un seul argument ; dans ce cas, on extrait la sous-chaîne commençant à la position indiquée par l'argument jusqu'à la fin de la chaîne.

Ligne 13 : La méthode `indexOf()` retourne la position de la première occurrence du caractère indiqué en argument dans la chaîne. Les littéraux de type caractère sont indiqués entre quottes (') et les littéraux chaîne entre double quotte (").

Ligne 15 : Cette ligne présente un autre usage de la méthode `indexOf()` qui prend en entrée une chaîne et retourne le début de cette sous-chaîne dans la chaîne de départ.

Ligne 17 : La méthode `toUpperCase()` est utilisée pour convertir la chaîne en majuscule

7.2.1 La valeur de retour d'une méthode

Une méthode peut retourner la référence à un objet, un type de base, ou pas de valeur du tout. Dans le programme `CheckString`, toutes les methodes de l'objet `str` retourne des valeurs qui sont affichées. Par exemple `charAt()` retourne le caractère qui se trouve à la position indiquée. La valeur retournée par une méthode peut aussi être stockée dans dans une variable :

```
String label = "From" ;
String upper = label.toUpperCase();
```

Dans l'exemple précédent, l'objet `upper` contient la valeur retournée par l'appel de méthode `label.toUpperCase()`.

7.2.2 Les méthodes de classes

Les méthodes de classe comme les variables de classe, s'appliquent à une classe toute entière et non à une instance. Les méthodes de classe sont utilisées pour effectuer des opérations qui concernent toutes les instances de la classe à la fois. Par exemple la classe `Math` définie dans le paquetage `java.lang` contient un large ensemble de fonctions mathématique comme méthode de classe. Il n'existe aucune instance de la classe `Math`, mais on peut continuer à utiliser ces méthodes avec des arguments de type numérique ou booléen. Par exemple, la méthode de classe `Math.Max()`, prend deux arguments, et retourne le plus grand des deux. On n'a pas besoin de

créer une instance de la classe `Math`. Il peut être appelé à n'importe quel endroit où vous avez besoin d'en faire usage :

```
int maximumPrice = Math.max(firstPrice, secondPrice) ;
```

La notation point est utilisé pour appeler une méthode de classe. Comme avec une variable de classe, le nom de la classe se trouve à la gauche de l'opérateur. Comme signalé plus haut, ce procédé augmente la lisibilité du code.

```
String s, s2;
s= "item";
s2=s.valueOf(5);
s2=String.valueOf(5);
```

7.3 La référence à un objet

Une référence est une adresse qui indique où une variable d'instance ou une méthode d'instance est stockée. Vous n'utilisez pas un objet lorsque vous l'affectez à une variable ou lorsque vous le passez comme argument à une méthode. Vous n'utilisez pas une copie de l'objet mais plutôt une référence à cet objet. Pour mieux illustrer cette différence, le programme ci-après montre comment les références fonctionnent.

Listing 4.4 : Source ReferencesTest

```
1: import java.awt.Point;
3: class ReferencesTest {
4: public static void main(String[] arguments) {
5: Point pt1, pt2;
6: pt1 = new Point(100, 100);
7: pt2 = pt1;
9: pt1.x = 200;
10: pt1.y = 200;
11: System.out.println("Point1: " + pt1.x + ", " + pt1.y);
12: System.out.println("Point2: " + pt2.x + ", " + pt2.y);
13: }
14: }
```

Le programme produit les sorties suivantes :

```
point1 : 200, 200
point2 : 200, 200
```

Ligne 5 : deux variables du type Point sont créées

Ligne 6 : un nouvel objet Point est assigné à la variable `pt1`

Ligne 7 : la valeur de pt1 est affectée à la variable `pt2`

La ligne 9 à 12 affiche les valeurs de `pt1` et `pt2`

Vous vous attendiez à ce que les valeurs de `pt1` et `pt2` soient différentes. Mais les résultats affichés indiquent que ce n'est pas le cas. Comme vous pouvez le voir, les attributs `x` et `y` de

pt1 sont modifiés et ce qui provoque la modification des attributs de l'objet pt2. Ce changement est causé par l'instruction de la ligne 7 qui créé une référence de pt2 vers pt1.
Pt2 est une référence à pt1 ;

7.4 Conversion de type entre objet et type de base

Lorsque vous envoyez des arguments à une méthode, ou utilisez une variable dans une expression, vous devez faire usage du bon type de données. Si une méthode nécessite un argument de type int, le compilateur java renvoie un message d'erreurs dans le cas où le type de l'argument est un float.
Parfois, vous allez avoir une valeur dans votre programme java qui n'est pas d'un type approprié. Dans ce cas, vous allez utiliser l'opérateur de cast pour changer de type. Par exemple, vous pourrez convertir un float en int. Le processus de conversion de type consiste à produire une nouvelle valeur avec un type différent de la valeur source. Dans cette section, nous allons présenter trois formes de conversion de type.

7.4.1 L'opérateur de cast et la conversion de type entre type primitif

L'opérateur de cast permet de convertir les types de base entre eux. Dans la plupart des cas, la conversion des types se fait entre les types numériques. Les valeurs booléennes ne peuvent prendre que la valeur true ou false et peuvent être utilisées dans une opération de cast.
Dans beaucoup d'opération de conversion entre type primitif, le type de destination est généralement plus large que le type source, ainsi la conversion est plus simple. Un exemple est la conversion d'un byte en int. Un byte permet de stocker les valeurs entre -128 et 127 et int de – 2100000 à 2100000, il y a alors assez d'espace pour un int d'accueillir un byte.
En générale on peut utiliser un byte ou un char comme un int, ; un int peut être utilisé comme un long ou comme un float, et n'importe quel type numérique comme un double. Dans beaucoup de cas un type plus large offre une précision plus grande qu'un type plus petit. Lorsque la conversion se fait d'un type plus petit vers un type plus grand il n ya pas de perte d'information. Dans le cas contraire on peut perdre des informations. Vous devez utiliser l'opérateur de cast lorsque vous voulez convertir une valeur de type plus large vers un type plus petit. La conversion explicite prend la forme suivante :

```
(typename) value ;
```

Dans l'exemple précédent, `typename` est le nom du type de données vers lequel on convertit `value`. `typename` peut être `short`, `int`, `float` ou `double`. `value` est une expression dont le résultat est converti dans le type `typename`. Par exemple :

```
int x=5 ; long y=10 ; int z
z=(int)(x/y) ;
```

`z` aura pour valeur 0.

7.4.2 Conversion entre objets

Les instances de classe peuvent aussi être converties en d'autres instances de classe, avec une restriction : la classe source et la classe de destination doivent être reliées par une relation d'héritage. Une classe doit être la super classe d'une autre. Pour utiliser une sous classe à la place d'une super classe, une conversion explicite doit être effectuée :

`(classename)value`

Par exemple considérons les classes `Employee` et `VicePresident`. La classe `VicePresident` est une sous-classe de la classe `Employee` avec plus d'information pour indiquer que le vice-président possède des privilèges par rapport aux autres employés :

```
Employee emp = new Employee() ;
VicePresident veep = new VicePresident() ;
emp = veep; //Pas de conversion nécessaire
veep= (VicePresident) emp ; //conversion nécessaire
```

7.4.3 Conversion entre les types de base et les objets

Quelque chose que vous ne pouvez pas faire tout le temps est de convertir un type de base en type objet et vice versa. Les types de base et les types objets sont très différents en Java et vous ne pouvez pas faire des conversions entre eux ou les utiliser de manière interchangeable. Comme alternative, le paquetage `java.lang` inclut des classes qui correspondent chacune à un type de base : `Float`, `Booleean`, `Byte`, et ainsi de suite. La plupart de ces classes ont le même nom que le type de base qu'il représente mais avec le nom commençant par une lettre majuscule. Deux classes font exception a cette règle : le type `int` qui est représenté par la classe `Integer` et char par `Character`. Java traite les types de données et leurs classes associées de manière différente, et dans un programme il n'est pas possible de faire usage de l'un à la place de l'autre.

En utilisant les classes qui correspondent au type de base on peut créer des objets qui stockent des valeurs de ce type. L'instruction qui suit créer un objet `Integer` avec la valeur `7801` :

```
Integer dataCount = new Integer(7801) ;
```

Lorsque vous créez un objet de cette manière vous pouvez l'utiliser comme n'importe quel objet. Si vous voulez plus tard l'utiliser comme un type de base vous avez des méthodes qui vous permettent de le faire. Par exemple si vous voulez faire usage de la valeur entière `7801` stockée dans l'objet `dataCount`, l'instruction suivante doit être utilisée :

```
int newCount= dataCount.intvalue();//retourne la valeur 7801
```

Une conversion dont vous avez besoin dans vos programmes est de convertir une chaîne de caractère en valeur numérique. Lorsque vous voulez un entier comme résultat, la méthode de classe `parseInt()` de la classe `Integer` peut être utilisée. La chaîne à convertir est le seul argument à envoyer à la méthode `parseInt()`, comme dans le cas cidessous :

```
String pennsylvania="65000";
int penn = Interger.parseInt(pennsylvania) ;
```

7.5 Comparaison des valeurs d'objets et de classes

En addition à l'opération de conversion, les trois opérations suivantes sont souvent réalisées sur les objets :

- comparer les objets
- rechercher la classe de chaque objet
- vérifier si un objet est une instance d'une classe donnée

7.5.1 Comparaison des objets

Les opérateurs de comparaison présentés au chapitre précédent marchent dans la plus part des cas pour les type de base mais pas pour les objets. Si vous essayez d'utiliser ces opérateurs pour comparer des objets, le compilateur Java produira des messages d'erreur. L'exception a cette règle concerne l'opérateur d'égalité (==) et l'opérateur de différence (!=). Lorsqu'ils sont utilisés pour comparer deux objets ces méthodes ne fonctionnent pas comme vous pourriez l'espérer. En effet au lieu de comparer la valeur de deux objets, il détermine si les deux objets ont la même référence. Pour comparer deux objets et avoir un résultat ayant un sens, il faut implémenter une méthode qui effectue la comparaison.

La classe `String` est un bon exemple. Il est possible d'avoir deux objets `String` différents qui ont la même valeur. Si vous utiliser l'opérateur `==`, pour comparer ces deux valeurs il seront considérés comme inégales. Pour comparer la valeur de deux objets de type `String`, la méthode de classe `equals()` est utilisée. La méthode compare les deux chaînes caractère par caractère et retourne la valeur `true` si elles ont la même valeur. Le listing 4.5 illustre bien la comparaison entre deux chaînes

Listing 4.5 EqualsTest.java

```
1: class EqualsTest {
2: public static void main(String[] arguments) {
3: String str1, str2;
4: str1 = "Free the bound periodicals.";
5: str2 = str1;
7: System.out.println("String1: " + str1);
8: System.out.println("String2: " + str2);
9: System.out.println("Same object? " + (str1 == str2));
11: str2 = new String(str1);
13: System.out.println("String1: " + str1);
14: System.out.println("String2: " + str2);
15: System.out.println("Same object? " + (str1 == str2));
16: System.out.println("Same value? " + str1.equals(str2));
17: }
18: }
```

Le programme affiche en sortie les résultats suivants :

```
String1: Free the bound periodicals.
String2: Free the bound periodicals.
Same object? true
String1: Free the bound periodicals.
String2: Free the bound periodicals.
Same object? false
Same value? true
```

La première partie de ce programme (ligne 3 à 5) déclare deux variables (`str1` et `str2`) et leur affecte la valeur "`Free the bound periodicals`". Comme appris plus haut, str1 et str2 maintenant pointent sur la même valeur et test d'égalité de la ligne 9 le prouve. Dans la seconde partie de ce programme (ligne 11) un nouvel objet str2 est créé avec la même valeur que str1. Maintenant nous avons deux objets différents comme le prouve le test effectué à la ligne 15 qui retourne la valeur `false`. Le test effectué en utilisant la méthode `equals()` (ligne 16) retourne la `true` car les deux chaînes ont la même valeur.

7.5.2 Comment déterminer la classe d'un objet

Lorsque vous voulez connaître la classe d'un objet, java vous propose une méthode nommée `getName()` qui retourne le nom de la classe comme une chaîne de caractères. L'exemple ci – dessous indique comment utiliser cette méthode.

```
String name = key.getClass().getName() ;
```

La méthode `getClass` est définie dans la classe `Object` et est disponible pour tous les autres objets. Un autre test intéressant est l'opérateur `instanceof`. Cet opérateur a deux opérantes : la référence à un objet à droite et le nom de la classe à gauche.

L'expression retourne les valeurs `true` ou `false` suivant que l'objet est une instance de la classe spécifique ou pas. Par exemple :

```
"Texas" instanceOf String // true
Point pt = new Point(10, 10) ;
pt instanceOfString // false
```