

Principes algorithmiques et programmation

Contents

1. Description du dossier pédagogique	4
1.1. Finalité de la section	4
1.2. Finalité de l'unité de formation algorithme et programmation	4
1.3. Programme du cours	5
1.4. Capacités terminales - Evaluation	5
2. Démarche pédagogique.....	6
2.1. Ressources	6
3. Installation de Java et Eclipse	7
3.1. Installation de Java.	7
3.2. Installation d'Eclipse	8
4. Principes algorithmiques et programmation	10
4.1. Première liste d'exercices liés au projet	10
4.2. Création d'un projet et d'une classe Java	21
4.3. Exécution d'un programme java.	24
4.3.1. Byte Code – Machine Virtuelle (JVM)	27
4.4. Premier programme "élaboré"	29
4.4.1. Variables et affectations	29
4.4.2. Séquencement des instructions - Debug.....	31
4.4.3. Importance des types de donnée.....	32
4.5. Le codage de l'information.....	32
4.5.1. Les nombres binaires.....	33
4.5.2. Synthèse sur les nombres et types entiers en java.....	37
4.5.3. Représentation des nombres à virgule flottante (norme IEEE754).....	37
4.5.4. Les conversions numériques	44
4.5.5. Les objets nombres en Java.....	45
4.5.6. Le codage hexadécimal	47
4.5.7. Le codage ASCII.....	49
4.5.8. L'Unicode.....	50
4.5.9. Les caractères	51
4.6. Algorithmes - Pseudo Code.....	52
4.6.1. Exercice.....	54
4.7. Programmation structurée – les fonctions et procédures.....	55
4.7.1. Procédures et fonctions	55
4.7.2. Pseudo code.....	57
4.7.3. Exercices	58

4.8.	Saisies utilisateur	59
4.8.1.	Exemple.....	59
4.8.2.	Pseudocode.....	60
4.8.3.	Exercices	60
4.9.	Chaînes de caractères.....	61
4.9.1.	La classe <code>String</code>	61
4.9.2.	Exercice.....	64
4.10.	Structures alternatives	65
4.10.1.	Structure alternative if	65
4.10.2.	Structure alternative - Switch Case	70
4.10.3.	L'opérateur ternaire ? :	72
4.11.	Tableaux (Arrays)	73
4.11.1.	Tableau unidimensionnel.....	73
4.11.2.	Tableaux multidimensionnels.....	76
4.12.	Les structures répétitives.....	81
4.12.1.	La boucle REPETER (faire) ... TANT QUE (jusqu'à ce que)	81
4.12.2.	La boucle TANT QUE	84
4.12.3.	La boucle FOR	86
4.12.4.	Les boucles imbriquées	89
4.13.	Gestion des exceptions.....	91
4.13.1.	Pseudo code	91
4.13.2.	Exercices.....	91
4.14.	Les Arrays List.....	92
4.14.1.	Autres possibilités de parcours et modifications des ArrayList	93
4.14.2.	Exercice	94
4.15.	Création de fichier texte et écriture.....	96
4.15.1.	Pseudo code	97
4.15.2.	Exercice	97
4.16.	Date et Heure	98
4.16.1.	Pseudo Code	99
4.16.2.	Exercice	99
4.17.	Variables globales et variables locales	100
4.18.	Passage de paramètres par adresse ou valeur.....	101
4.19.	Exercice récapitulatif – création du ticket de caisse.	104

1. Description du dossier pédagogique

Depuis le site du segec, il est possible de télécharger le dossier pédagogique de la section informatique de gestion

(<http://enseignement.catholique.be/segec/index.php?SECTEUR=7&id=1026&NIVEAU=>).

Celui-ci reprend, entre autres, la finalité de la section et le contenu des différents cours.

1.1. Finalité de la section

Le cours de principes algorithmiques et programmation se retrouve clairement dans certains points décrits dans les finalités de la section.

- Finalités générales

- répondre aux besoins et demandes en formation émanant des entreprises, des administrations, de l'enseignement et d'une manière générale des milieux socio-économiques et culturels.

- Finalités particulières

Conformément au champ d'activité et aux tâches décrites dans le profil professionnel ci-annexé, cette section doit permettre à l'étudiant :

- de développer des ensembles de compétences dans différents axes :
- la programmation dans différents environnements,
- les systèmes de gestion de base de données,
- l'analyse et la conception d'applications,
- de prendre conscience de l'évolution constante des métiers de l'informatique et des technologies de l'information et de la communication ;
- de mettre en oeuvre des comportements professionnels fondés sur le sens critique, l'auto – formation et le sens de la communication ;
- de mettre en pratique des compétences collectives (travail partagé ou en équipe) et des aptitudes à s'insérer dans des projets et des démarches de gestion de la qualité.

1.2. Finalité de l'unité de formation algorithme et programmation

Finalités particulières

L'unité de formation vise à permettre à l'étudiant :

- de développer des comportements professionnels ;
- développer des compétences collectives par le travail en équipe ;
- prendre conscience des compétences à développer en ce domaine pour répondre d'une manière appropriée à l'évolution des techniques et des besoins de la clientèle en ce domaine ;
- de mettre en oeuvre, d'une manière appropriée, des techniques, des méthodes spécifiques pour :
 - appréhender, globalement, la diversité méthodologique de la fonction de programmation dans le secteur des métiers de l'informatique et dans les besoins de la clientèle (entreprises publique et privée) ;
 - développer des compétences de base en utilisation d'un langage largement utilisé dans le monde des entreprises ;

- mettre en œuvre une démarche algorithmique cohérente.

1.3. Programme du cours

En disposant d'une structure informatique matérielle et logicielle opérationnelle et d'une documentation appropriée, l'étudiant sera capable de :

- identifier les différents langages de programmation existants
- choisir le langage de programmation approprié à son application
- mettre en œuvre une méthodologie de résolution de problème (observation, résolution, expérimentation, validation) et de la justifier en fonction de l'objectif poursuivi ;
- concevoir, construire et représenter des algorithmes, en utilisant :
 - les types de données élémentaires,
 - les figures algorithmiques de base (séquence, alternative et répétitive),
 - les instructions,
 - les portées des variables,
 - les fonctions et procédures,
 - la récursivité,
 - les entrées/sorties,
 - les fichiers,
 - les structures de données de base (tableaux et enregistrements) ;
- traduire de manière adéquate des algorithmes en respectant les spécificités du langage utilisé ;
- documenter de manière complète et précise les programmes développés ;
- produire des tests pour valider les programmes développés.

1.4. Capacités terminales - Evaluation

Pour atteindre le seuil de réussite, en disposant d'une structure informatique matérielle et logicielle opérationnelle et d'une documentation appropriée, face à un problème mettant en jeu des algorithmes de base, dans le respect du temps imparti, l'étudiant sera capable de

- mettre en œuvre une stratégie cohérente de résolution du problème posé ;
- concevoir, de construire et de représenter l' (les) algorithme(s) correspondant(s) ;
- justifier la démarche algorithmique et les choix mis en œuvre ;
- développer des programmes en respectant les spécificités du langage choisi ;
- mettre en œuvre des procédures de test.

Pour la détermination du degré de maîtrise, il sera tenu compte des critères suivants :

- de la qualité et de la pertinence de la démarche algorithmique,
- de la rigueur et du respect des spécificités du langage,
- du degré de précision et de la clarté dans l'emploi des termes techniques.

2. Démarche pédagogique

L'approche pédagogique de ce cours se veut résolument pratique. Les notions théoriques sont construites au fur et à mesure qu'elles sont nécessaires à la compréhension et à la résolution d'exercices de programmation. Ceux-ci sont eux-mêmes destinés à accomplir un projet de programmation plus vaste, présenté sous forme d'un cahier de charges issue d'une demande client.

Etant donné la forte hétérogénéité du public de promotion sociale, il est nécessaire à certains moments de pouvoir différencier les tâches. L'important est d'offrir à chacun la possibilité de progresser quelles que soient ses connaissances antérieures, son expérience éventuelle en algorithme, en programmation et en java.

Pour ce faire, le projet "brasserie" à réaliser, les exercices qui y sont associés sont disponibles dès le début du cours au §4.1. Des variantes du cahier des charges sont aussi proposées dans le projet. Ces variantes sont en quelque sorte des améliorations de la demande initiale. Il existe aussi une "training list" comportant une série d'exercices dont les difficultés sont progressives. Contrairement au projet, ces exercices n'ont aucun rapport, ni avec le projet, ni entre eux ; ils servent à décontextualiser les notions vues dans le projet brasserie et les appliquer dans d'autres domaines. Pour les plus aguerris, les passionnés, il est possible d'aller bien plus loin que ce que demande le dossier pédagogique pour ce cours.

Historiquement, le langage de programmation enseigné à l'IRAM est le Java. Ce langage courant issu du C\ C++ permet à l'utilisateur de se former aussi indirectement à ces langages. Java ou ses dérivés sont utilisés pour le développement d'applets pour le Web et pour des applications (non liées au web).

Il s'agit d'un langage orienté objet (POO) ce qui présente le bénéfice d'introduire l'UE de POO qui suivra. Paradoxalement, l'inconvénient du Java est qu'il s'agit d'un POO et qu'il inclut donc directement des notions et des spécificités liées à la POO. Beaucoup de termes visibles à l'écran et fonctionnalités dans l'environnement de développement seront donc éludés pour l'instant en attendant d'approfondir la POO. Il faut garder à l'esprit que le plus important dans ce cours n'est pas de former des experts en Java mais de former les esprits à l'analyse et à l'algorithme. Java n'est alors qu'un champ d'application de ces algorithmes, un exemple d'environnement de programmation et de mise à l'épreuve de ceux-ci.

L'apprentissage de la programmation se fait de diverses manières mais principalement, elle se construit dans le temps à force de pratiquer. Il est donc impossible de réussir en étudiant « à fond » seulement trois jours avant l'examen. La lecture d'ouvrages et la consultation de codes source réalisés par des programmeurs confirmés sont vivement recommandées.

2.1. Ressources

Internet regorge d'informations et de code Java. Parmi ceux-ci citons :

- <http://www.tutorialspoint.com/java/>
- <https://openclassrooms.com/courses/apprenez-a-programmer-en-java>
- <https://java.developpez.com/>
- <http://isn.codelab.info/ressources/algorithmique/memo-pseudo-codes/>

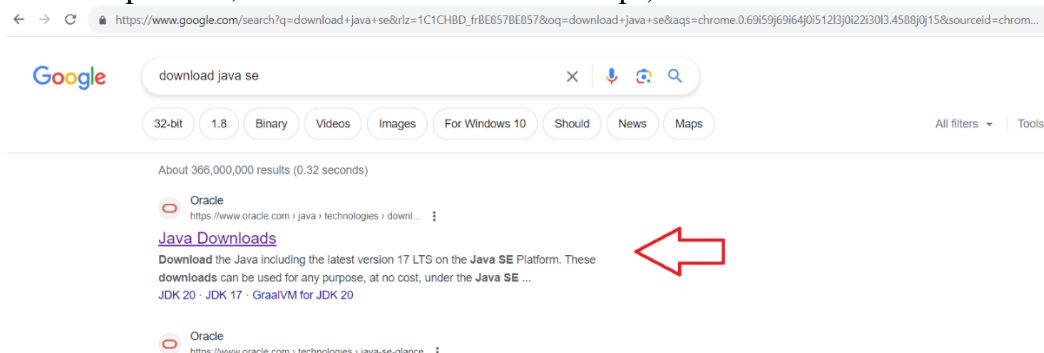
bibliographie :

- Programmer en Java (Claude Delannoy) – Eyrolles
- Exercices en Java (Claude Delannoy) - Eyrolles

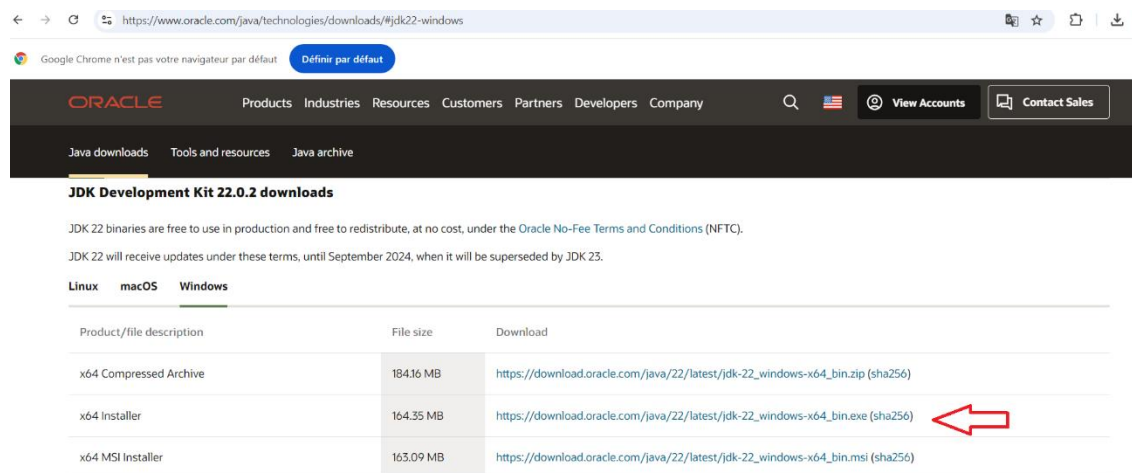
3. Installation de Java et Eclipse

3.1. Installation de Java.

Depuis un moteur de recherche, entrez "download Java SE". Sur le site d'Oracle, dans l'onglet Downloads, cliquez sur download pour télécharger la dernière version de Java Platform JDK (version 17 ou supérieure, la version évolue dans le temps).



Suivant votre OS, sélectionnez le bon fichier d'installation. Sachant que tous les codes et exercices présentés dans ce cours ont été testés sous Windows, que les PC de l'école tournent sur Windows, il vous est conseillé d'utiliser Java sous Windows. Il vous est néanmoins tout à fait possible d'utiliser Java avec un autre OS et c'est d'ailleurs un des points forts de Java. Cependant, en cas de problème de fonctionnement, une aide ne pourra être donnée pour chaque spécificité liée à votre OS ou logiciels annexes.

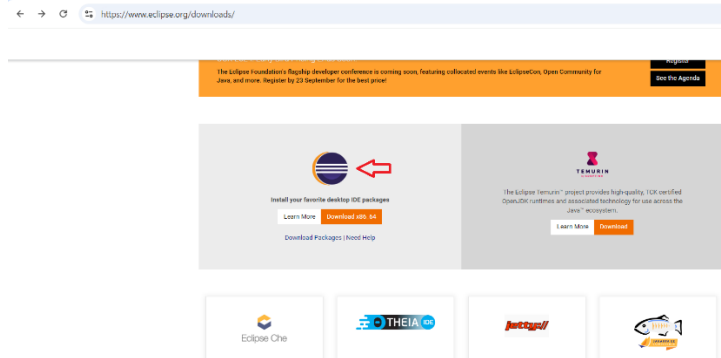


- Une fois le fichier téléchargé, lancez l'exécution de ce fichier d'installation et cliquez sur *Next* aux invitations qui surviennent durant l'installation.
- Redémarrez votre PC à la fin de l'installation.

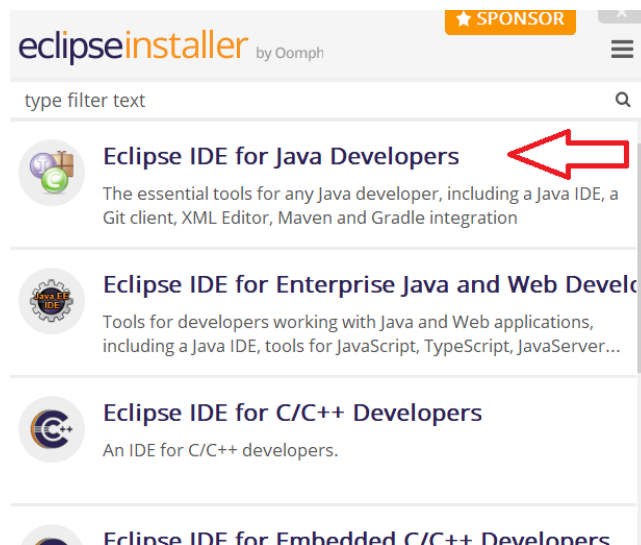
3.2. Installation d'Eclipse

Il existe différents environnements de développement c'est-à-dire des logiciels qui facilitent le codage des programmes. Les plus connus sont NetBeans et **Eclipse**. C'est ce dernier qui est utilisé dans le cadre de ce cours.

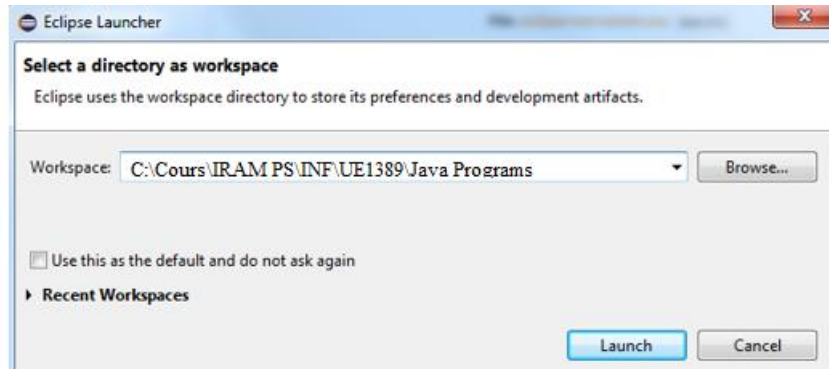
- Téléchargez Eclipse depuis le site [eclipse.org](https://www.eclipse.org/downloads/), cliquez sur le menu *Downloads*, pour télécharger Eclipse.



- Lancez l'exécution de l'installateur et choisissez *Eclipse IDE for Java Developers*.



Au lancement d'Eclipse, il faut sélectionner le chemin *workspace* ; il s'agit du répertoire par défaut où vont être stockés tous nos programmes Java. Il vous est vivement conseillé de créer, au préalable, un chemin et un répertoire du type IRAM PS\INF\UE1389\Java Programs\ et d'indiquer ce chemin dans le *Workspace's* directoy sans quoi il sera pénible de retrouver vos fichiers par la suite.



4. Principes algorithmiques et programmation

Rappelons que le cours est construit autour d'un projet de programmation à réaliser (voir fichier annexe Projet 1 Brasserie Le passe-temps).

4.1. Première liste d'exercices liés au projet

La première fonctionnalité à réaliser pour la brasserie est l'impression d'un ticket de caisse qui devra apparaître sous cette forme :

```

2018-11-26 17h47m45s-430.txt - Bloc-notes
Fichier Edition Format Affichage ?
Café-Brasserie
Le passe temps
26/11/18 17:47

Table:2
-----
Spa reine 25          2      2,20    4,40
Pepsi                2      2,20    4,40
Spaghetti Bolo       2     10,80   21,60
Tagl saum            4     16,90   67,60
Jus d'orange Looza   1      2,60    2,60
-----
TOTAL CONSOMMATIONS      11
-----
TOTAL TVA 21,00%          1,98   11,40
TOTAL TVA 12,00%          9,56   89,20
TOTAL TVA  6,00%          0,00    0,00
-----
TOTAL                      100,60
  
```

Avant de pouvoir créer ce ticket, on va réaliser des exercices plus simplistes mais toujours en relation avec cet objectif, l'encodage et l'impression d'un ticket de caisse.

- 1) PJ1Ex1-1 : On veut créer un programme qui affiche le texte sur une seule ligne :
Café-Brasserie : Le Passe-Temps.
- 2) PJ1Ex1-2 On veut créer un programme qui affiche le texte sur deux lignes par deux méthodes différentes. Le résultat de l'exécution doit être comme indiqué ci-dessous :
Café-Brasserie :
Le Passe-Temps
- 3) PJ1Ex1-3 Le programme calcule et affiche le prix total pour un type de consommation en fonction du nombre d'unités et du prix unitaire. Le nombre d'unités et le prix unitaire sont d'abord initialisés à des valeurs nulles pour ensuite être affectés à des valeurs quelconques.
- 4) PJ1Ex1-4-a Le programme calcule le montant du prix net à partir d'un prix brut donné et de la TVA (fixée à 21%). Le programme affiche. « Pour un prix brut xxx €, le prix net est de yyy € ». Le programme doit réaliser le calcul pour un prix brut de 100€ et aussi pour un prix brut de 125.5€. Pour information, le prix net est le prix brut auquel on a ajouté toutes les taxes et enlevé toutes les réductions éventuelles.

PJ1Ex1-4-b Le programme calcule le montant du prix net à partir d'un prix brut donné et de la TVA qui, cette fois, n'est pas forcément 21%. En utilisant une procédure avec paramètres, le programme affiche : « Pour un prix brut de xxx €, le prix net est de yyy

€ ». Pour le test du programme, appelez votre procédure au moins deux fois, une première fois pour un prix brut de 100€ et une TVA à 6%, une deuxième fois pour un prix brut de 125.5€ et une TVA à 21%.

PJ1Ex1-4-c Le programme calcule le montant du prix net à partir d'un prix brut donné et de la TVA qui, cette fois, n'est pas forcément 21%. En utilisant une fonction avec paramètres qui calcule le prix net, le programme affiche : « Pour un prix brut de xxx €, le prix net est de yyy € ». Pour le test du programme, appelez votre fonction au moins deux fois, une première fois pour un prix brut de 100€ et une TVA à 6%, une deuxième fois pour un prix brut de 125,5€ et une TVA à 21%.

- 5) PJ1Ex1-5 : Le programme calcule le montant de la TVA à partir d'un prix net (prix net = prix TVAC avec un taux de TVA fixé à 21%). Le programme affiche. « Pour un net de xxx €, le coût de la TVA s'élève à yyy € ». Une analyse préalable au pseudocode est nécessaire !

PJ1Ex1-5-b : Créez une procédure qui calcule et affiche le montant de la TVA à partir d'un prix net et un taux de TVA variables. Prendre 121€ et 21,0 %, 85,5€ et 12.5% pour les essais. Le programme affiche : « Pour un prix net de x € et une TVA de y %, le coût de la TVA s'élève à z € ».

PJ1Ex1-5-c : Créez une fonction qui calcule le montant de la TVA à partir d'un prix net et un taux de TVA variables. Prendre 121€ et 21,0 %, 85,5€ et 12.5% pour les essais. Le programme affiche : « Pour un prix net de x € et une TVA de y %, le coût de la TVA s'élève à z € ».

- 6) PJ1Ex1-6-a L'exercice est identique au précédent c'est-à-dire qu'il permet d'afficher le coût de la TVA à partir d'un prix net mais, cette fois, l'utilisateur entre le prix net au clavier répondant à l'invitation « *Entrez le prix net* ».

PJ1Ex1-6-b Améliorez l'exercice précédent en créant une première fonction de calcul du coût de la TVA *calculCoutTVA* (déjà réalisée au 5c), et une deuxième *getUserNetPriceInput()* qui renvoie la saisie de l'utilisateur (le prix Net) au format réel double précision.

- 7) PJ1Ex1-7 L'exercice est identique à l'exercice PJ1Ex1-4 c'est-à-dire qu'il permet de calculer le prix net à partir de la TVA et du prix brut mais, cette fois, l'utilisateur entre le prix brut et le taux de TVA répondant aux invitations « *Entrez le prix brut* » et « *Entrez le taux de TVA* ».

PJ1Ex1-7-b En plus de la fonction de calcul du prix net, créez une fonction qui renvoie le prix brut et une autre qui renvoie le taux de TVA.

PJ1Ex1-7-c Améliorez l'exercice PJ1Ex1-7-b qui consistait à calculer le prix net en fonction d'un prix brut et d'une TVA saisis par l'utilisateur. Remplacez les deux fonctions de saisie (*getUserGrossPriceInput()* et *getUserVATInput()*) par une seule *getUser_doubleInput()* avec cette fois, le texte d'invitation (« *Entrez le prix brut* », « *Entrez la TVA* ») passé en paramètre de la fonction.

- 8) PJ1Ex1-8-a L'utilisateur saisit un entier correspondant au N° de table répondant à l'invitation « Entrez le N° de table ». Le programme vérifie si le N° de table entré est correct. Si le N° est correct, le programme affiche « *Vous avez entré le numéro de table X* » (avec X le N° de table entré). Si le nombre est supérieur à 20 ou inférieur à 1, on affiche « *Numéro de table incorrect* ».

PJ1Ex1-8-b. Créez une fonction `getTableNumber()` qui permet de renvoyer le N° de table saisi par l'utilisateur répondant à l'invitation « *Entrez le numéro de table* ». La fonction vérifie si le N° de table entré est correct. Le N° doit être compris entre 1 et le N° de la dernière table qui est le `NumMax` passé en paramètre. Normalement, ce nombre est de 20 mais le programme doit avoir prévu une extension éventuelle de l'établissement ; c'est pourquoi le nombre 20 doit être une constante du programme. Si le nombre est supérieur au N° de table maximum ou inférieur à 1, la fonction affiche « *Numéro de table incorrect, le N° doit être compris entre 1 et N° max* ». Si le N° de table est incorrect, la fonction doit renvoyer -1 comme code d'erreur. Prévoyez plusieurs appels de la fonction de façon à la valider en testant tous les cas de figure.

- 9) PJ1Ex1-9 Créez une fonction `getAndCheckTableNumber()` qui permet de renvoyer le N° de table saisi par l'utilisateur répondant à l'invitation « *Entrez le numéro de table* ». La fonction vérifie si le N° de table entré est correct. La différence avec l'exercice précédent réside dans le fait que le message d'erreur doit être spécifique à l'erreur de saisie : si le N° est inférieur à 1, le programme affiche « *Saisie incorrecte ! Le N° de table doit être supérieur ou égal à 1* », si le nombre est supérieur à 20 (constante du programme), on affiche « *Saisie incorrecte ! Le N° table le plus élevé est NumMax. Contactez l'administrateur du programme si vous souhaitez augmenter ce nombre* », si le N° est correct, le programme `main()` affiche « *Vous avez entré le numéro de table X* » avec X le N° de table entré. Si le N° de table est incorrect, la fonction doit renvoyer -1 comme code d'erreur.

- 10) PJ1Ex1-10 Le programme calcule le prix net à partir de la TVA et du prix brut saisis par l'utilisateur. Le taux de TVA est choisi par l'utilisateur répondant à l'invitation "Pour un taux de TVA de 6% tapez 1, 2 pour 12% et 3 pour 21%". On affiche alors le texte : "Pour un prix brut de X €, le prix net est de Y € (TVA Z%)". Si l'entrée de l'utilisateur n'est pas dans les choix possibles, le programme s'arrête en indiquant « *Saisie du taux incorrecte* » et le prix brut n'est pas demandé. Utilisez les fonctions déjà développées pour récupérer le prix brut et calculer le prix net. Ajoutez une fonction `getVATChoice()` qui renvoie le taux de TVA choisi en %, -1.0 en cas de choix incorrect. Prévoyez tous les tests qui permettent de valider les fonctionnalités.

- 11) PJ1Ex1-11. L'utilisateur entre le N° de consommation répondant à l'invitation « *Entrez le N° de consommation* ». Selon le N° choisi, le programme affiche le nom de la consommation et son prix de vente qui peut être éventuellement réduit. Dans le cas où le N° n'existe pas, le programme affiche "N° non référencé" et se termine. Créez une fonction `getUser_intInput()` qui permet de récupérer un entier saisi par l'utilisateur avec comme paramètres : le message d'invitation, une valeur minimum et une valeur maximum admises (de 1 à 4 dans ce cas). Cette fonction renvoie -1 si le N° n'est pas correct. Après avoir choisi le N° de consommation, l'utilisateur doit répondre à la question "Happy Hour ? Y/N". Créez une fonction `checkAnswer()` ayant comme paramètres une question et une réponse attendue, qui renvoie vrai ou faux selon que la

réponse de l'utilisateur correspond (ou pas) à la réponse attendue ("Y" dans ce cas ci). Si l'utilisateur a répondu Y, le prix affiché est réduit de 50% par rapport au prix normal, autrement le prix indiqué est le prix normal. Parmi les N° de consommations, on considère seulement la N°1 Eau plate 3,00€, N°2 Coca Cola 3,00€, N°3 Bière pression 2,80€, N°4 Café long 2,90€. On doit obligatoirement recourir à la structure "switch case" pour la sélection de la consommation, la structure "if" pour les autres conditions rencontrées. Le programme affiche "*Prix de la consommation X : Y €*" avec X, le nom de la consommation et Y le prix de celle-ci.

Après réalisation du programme, critiquez la manière de stocker et d'exploiter les données des consommations et son implication dans la manière de devoir coder.

- 12) PJ1Ex1-12 Le programme est identique à l'exercice PJ1Ex1-10 mais on stocke les trois taux de TVA prédéfinis dans un tableau. Pour ce faire, déclarez un tableau `VAT[]` et modifiez la fonction `getVATChoice()` développée à l'exercice PJ1Ex1-10.
- 13) PJ1Ex1-13 Le programme est identique à l'exercice PJ1Ex1-11. mais on doit stocker les informations des consommations (dénominations et prix) dans deux tableaux `Names[]` et `Prices[]`. L'utilisation du "switch case" n'est, par contre, plus du tout requise.
- 14) PJ1Ex1-14 : Le programme contient un tableau à deux dimensions avec le N° d'emplacement des consommables dans la 1^{ère} colonne et le nombre d'unités en stock dans la 2^{ème}. Un autre tableau contient les dénominations des consommables.

Les tableaux sont initialisés comme suit :

index	N°emplacement	Unités en stock
0	19	92
1	6	16
2	14	27
3	4	72

index	Dénomination
0	Eau plate
1	Coca Cola
2	Bière pression
3	Café long

L'utilisateur entre le N° d'identifiant (1 pour l'eau plate, 2 pour le coca, ...) du consommable répondant à l'invitation "*Entrez le N° d'identifiant*". Si le N° d'identifiant est non référencé dans les tables, le programme indique "*N° non référencé*" puis s'arrête. (Pour cette entrée, utilisez la fonction déjà développée). Ensuite, l'utilisateur entre le nombre d'unités consommées répondant à l'invitation "*Entrez le nombre d'unités consommées pour X. Quantité actuellement en stock Y, emplacement Z*" (avec X le nom du consommable, Y la quantité en stock avant la modification et Z le N° d'emplacement). Le stock s'en trouve diminué de la valeur saisie. Si le nombre saisi dépasse la quantité actuelle en stock, le programme affiche "*Erreur de saisie ou de quantité en stock pour X ! Annuler ou Mettre à Zero le stock. A/Z*". Pour ce faire, réutilisez la fonction `checkAnswer()` développée pour l'exercice PJ1Ex1-13. Dans le cas où l'utilisateur n'a pas demandé la mise à zéro du stock, le programme n'impacte pas le stock et affiche le message "*Modification annulée*" ; en cas de remise à zéro, le stock pour ce produit est mis à 0 et le programme affiche "*Stock 0 pour X*". Le programme affiche, au début et à la fin du programme, un récapitulatif, ligne par ligne, avec l'identifiant, le nom de la

consommation, l'emplacement et la quantité en stock (en tenant compte de la modification effectuée à la fin). Pour ce faire, créez une procédure `showStock()`. Les tableaux peuvent être déclarés en variables globales ; dans ce cas, ils ne doivent pas être passés en paramètres à la procédure.

- 15) PJ1Ex1-15 Le programme est semblable à l'exercice PJ1 Ex1-9
L'utilisateur saisit un entier correspondant au N° de table où les consommations ont été prises répondant à l'invitation « *Entrez le N° de table* ». Le programme vérifie si le N° de table entré est correct. Si le N° est inférieur à 1, le programme affiche « *Saisie incorrecte ! Le N° de table doit être supérieur ou égale à 1* », si le nombre est supérieur à 20, on affiche « *Saisie incorrecte ! Le N° table le plus élevé est 20. Contactez l'administrateur du programme si vous souhaitez augmenter ce nombre* », si le N° est correct, le programme affiche « *Vous avez entré le numéro de table XX* » avec X le N° de table entré. La différence avec l'exercice PJ1 EX1-9 réside dans le fait que, aussi longtemps que le N° de table entré n'est pas valide, le programme repose la question « *Entrez le N° de table* ».
- 16) PJ1Ex1-16 Le programme indique que le stock, pour une consommation précise, est constant et fixé à 100 unités initialement. Au fur et à mesure des validations de ticket, le stock va diminuer. Réalisons donc un programme semblable à cela. L'utilisateur entre un nombre d'unités consommées répondant à l'invitation « *Entrez le nombre d'unités consommées* ». Le stock se trouve diminué de la quantité encodée, il est affiché le message. « *Nombre d'unités en stock XX* » (avec XX le stock actuel). L'opération se répète jusqu'à ce que le stock soit à 0, alors le programme affiche « *Stock à 0 d'unités* ». Si jamais l'utilisateur entre un nombre plus grand que le stock actuel, le message d'erreur « *saisie incorrecte* » s'affiche mais les invitations « *Entrez le nombre d'unités consommées* » se poursuivent.
- 17) PJ1Ex1-17. Lors de l'élaboration du ticket de caisse, le serveur aura l'occasion de saisir des réponses aux invitations du programme telles que "*Voici une question... / A pour Annuler, V pour valider, Q pour quitter le programme*". Créez une fonction `getUserSpecificInput(...)` qui pose une question à l'utilisateur, et renvoie une réponse valide si elle correspond bien à la demande (A, V ou Q dans l'exemple) sans ambiguïté (donc ni "VQ" ni "AQ" ni "AVQ", ...une seule lettre est attendue). Dans le cas d'une réponse non valide, le message d'erreur s'affiche "*Erreur de saisie, votre choix doit être parmi XX..X, une seule lettre seulement*" (XX_X est AVQ dans notre exemple) puis la question est reposée. Les possibilités de réponse ne sont pas limitées à trois, les caractères attendus doivent se trouver en paramètre (`expectedAnswers`) de la fonction rassemblés dans une chaîne de caractères "AVQ" pour notre exemple. La fonction renvoie une chaîne de caractères même si elle ne comprend qu'un seul caractère (donc soit "A", "V" ou "Q" pour l'exemple). L'appel de la fonction peut être :
- ```
getUserSpecificInput("Voici une question / A pour Annuler, V pour valider, Q pour quitter le programme",
"AVQ") ;
```
- 18) PJ1Ex1-19 Le programme est semblable à l'exercice PJ1Ex1-16 mais, cette fois, le stock initial est saisi par l'utilisateur (au lieu d'être fixé à 100) répondant à l'invitation « *Entrez le stock initial* ». L'utilisateur entre ensuite le nombre d'unités consommées répondant à l'invitation « *Entrez le nombre d'unités consommées* ». Le stock se trouve diminué de la quantité encodée, il est affiché le message. « *Nombre d'unités en stock XX* ». L'opération

se répète jusqu'à ce que le stock soit à 0 dans ce cas le message « *Stock nul* » s'affiche. Si jamais l'utilisateur entre un nombre plus grand que le stock actuel, le message d'erreur « *saisie incorrecte* » s'affiche mais les invitations « *Entrez le nombre d'unités consommées* » se poursuivent. Remarque : il se peut que l'utilisateur entre un stock initial de 0 dans ce cas le message « *Stock nul* » s'affiche directement.

- 19) colonnes. La première colonne correspond au N° de la consommation qui est à une unité près l'index dans le tableau `Names[]` (la consommation N°1 porte l'index 0 dans le tableau `Names[]`). La deuxième colonne contient le nombre de consommations. Voici un exemple de contenu du tableau `Order[][]` :

|    |   |
|----|---|
| 3  | 2 |
| 1  | 3 |
| 12 | 4 |
| 37 | 1 |
| 36 | 3 |
| 0  | 0 |
|    |   |
|    |   |

```
int Order[][]={{3,2},{1,3},{12,4},{37,1},{36,3},{0,0},{0,0},{0,0},{0,0}};
```

Par convention, le tableau ne contient plus de données utiles quand on rencontre 0 et 0 sur une ligne. Créez une procédure `previewOrder()` utilisant la boucle TANT QUE, qui affiche le détail de la commande en indiquant le nom des consommations, le nombre d'unités et le prix net total par consommation. Le tableau `Order[][]` sera passé en paramètre à la procédure. Utilisez en variables globales les tableaux `Names[]` et `NetPrices[]` de l'exercice PJ1Ex1-27.

```
final static String Names[] = {"Spa reine 25 ", "Bru plate 50", "Bru pét 50", "Pepsi", "Spa orange",
"Schweppes Tonic", "Schweppes Agr", "Ice Tea", "Ice Tea Pêche", "Jus d'orange Looza", "Cécémel", "Red
Bull", "Petit Espresso", "Grand Espresso", "Café décaféiné ", "Lait Russe ", "Thé et infusions", "Irish
Coffee ", "French Coffee ", "Cappuccino", "Cécémel chaud", "Passione Italiano", "Amour Intense",
"Rhumba Caliente ", "Irish Kisses ", "Cuvée Trolls 25", "Cuvee Trolls 50", "Ambrasse-Temps
25", "Ambrasse-Temps 50 ", "Brasse-Temps Cerises 25", "Brasse-Temps Cerises 50", "La Blanche Ste
Waudru 25", "Blanche Ste Waudru 50", "Brasse-Temps citr 25", "Brasse-Temps citr 50", "Spaghetti Bolo
", "Tagl Carbonara", "Penne poulet baslc ", "Tagl American", "Tagl saum"};
```

```
final static double NetPrices[] =
{2.2,2.3,3.9,2.2,2.2,2.6,2.6,2.6,2.6,2.6,4.5,2.2,2.2,2.2,2.5,2.5,7.0,7.0,2.8,2.8,6.2,6.2,6.2,6
.2,2.9,5.5,2.7,5.1,3.1,5.8,2.6,4.9,2.6,4.9,10.8,11.2,12.2,14.5,16.9};
```

Pour l'exemple, l'affichage est tel que celui-ci :

```
Bru pét 50 3.9 2 7,80
Spa reine 25 2.2 3 6,60
Red Bull 4.5 4 18,00
Tagl Carbonara 11.2 1 11,20
Spaghetti Bolo 10.8 3 32,40
```

- 20) PJ1Ex1-20. Les noms des consommations sont stockés dans un tableau `Names[]` et leur prix net dans un autre tableau `NetPrices[]` (Voir Ex PJ1Ex1-27). En utilisant une boucle FOR, réalisez une procédure `showMenu()` qui affiche l'entièreté du menu. L'affichage comprend la liste des consommations avec leur prix net sous la forme ci-dessous :

```
Spa reine 25 cl 2,20 €
Bru plate 50 cl 2,3 €
Bru légèrement pétillante 50 cl 3,9 €
Pepsi, Pepsi max 2,20 €
Spa orange 2,20€
```

...

- 21) PJ1Ex1-21 Les noms des consommations sont stockés dans un tableau identique à celui de l'exercice précédent. L'utilisateur doit encoder le stock pour toutes les consommations répondant à l'invitation « *Entrez le stock pour X* » avec *X* le nom de la consommation. Le programme affiche ensuite un récapitulatif avec ligne par ligne, le nom des consommations avec le stock encodé précédemment. Réalisez le programme d'abord uniquement avec des boucles FOR, puis uniquement avec des boucles TANT QUE. Limitez les tableaux à 5 éléments de façon à raccourcir le temps de saisie des stocks.

--Récapitulatif--

Spa reine 25 cl Stock : 45  
 Bru plate 50 cl Stock : 120  
 Bru légèrement pétillante 50 cl Stock : 200  
 Pepsi, Pepsi max Stock : 240  
 Spa orange Stock : 90

(Exemple de résultat d'exécution après saisie)

- 22) PJ1 Ex1-22. On veut parcourir et afficher le contenu de tableaux ci-dessous.

| Index<br>tableau | Identifiant | Quantité en<br>stock | Quantité à<br>prévoir | Quantités vendues<br>semaine en cours | Index<br>tableau | Consommable                     |
|------------------|-------------|----------------------|-----------------------|---------------------------------------|------------------|---------------------------------|
| 0                | 1           | 56                   | 200                   | 55                                    | 0                | Spa reine 25 cl                 |
| 1                | 2           | 42                   | 200                   | 60                                    | 1                | Bru plate 50 cl                 |
| 2                | 3           | 62                   | 200                   | 125                                   | 2                | Bru légèrement pétillante 50 cl |
| 3                | 4           | 45                   | 200                   | 150                                   | 3                | Pepsi, Pepsi max                |
| 4                | 5           | 25                   | 200                   | 140                                   | 4                | Spa orange                      |
| 5                | 6           | 72                   | 200                   | 86                                    | 5                | Schweppes Tonic                 |
| 6                | 7           | 40                   | 200                   | 47                                    | 6                | Schweppes Agrumes               |
| 7                | 8           | 48                   | 200                   | 80                                    | 7                | Lipton Ice Tea                  |
| 8                | 9           | 24                   | 150                   | 75                                    | 8                | Lipton Ice Tea Pêche            |
| 9                | 10          | 36                   | 200                   | 90                                    | 9                | Jus d'orange Looza              |
| 10               | 11          | 15                   | 100                   | 55                                    | 10               | Cécémel                         |
| 11               | 12          | 25                   | 80                    | 23                                    | 11               | Red Bull                        |

Le tableau de gauche contient le N° d'identifiant, les quantités actuellement en stock, les quantités à prévoir et les quantités vendues la semaine cours, ceci pour chaque identifiant de consommation. Un autre tableau contient les noms des consommations. Le programme doit afficher ligne par ligne, le N° d'identifiant, le nom de la consommation et les différentes quantités en stock correspondantes séparés simplement par un espace. L'affichage doit donc être de ce type :

N° 1 Spa reine 25 cl 56 200 55  
 N° 2 Bru Plate 50 cl 42 200 60

...

Utilisez par exemple les tableaux suivants :

```
static int Stocks[][] = {{1, 56, 200, 55}, {2, 42, 200, 60}, {3, 62, 200, 125},
, {4, 45, 200, 150}, {5, 25, 200, 140}, {6, 72, 200, 86}, {7, 40, 200, 47}, {8, 48, 200, 80}, {9, 24
, 150, 126}, {10, 36, 200, 164}, {11, 15, 100, 85}, {12, 25, 80, 23}};
```

```
static String Names[] = {"Spa Reine", "Bru Plate", "Bru légèrement
pétillante", "Pepsi", "Spa Orange", "Schweppes Tonic", "Schweppes
```



```
Agrumes", "Lipton Ice Tea", "Lipton Ice Tea Pêche", "Jus d'orange
Looza", "Cécémel", "Red Bull"};
```

- 23) PJ1 Ex1-23 Le programme est semblable à l'exercice PJ1 Ex1-22 mais l'affichage à la console est amélioré pour obtenir un alignement correct des données. Ce programme servira également de base pour l'affichage des données du ticket de caisse. Les unités de la première colonne de nombres sont alignées au 35<sup>ème</sup> caractère ; au 40<sup>ème</sup> et 45<sup>ème</sup> caractère pour les deux autres colonnes d'entiers. Les textes sont donc alignés à gauche et les nombres à droite. Le N° d'identifiant ne doit plus être affiché. Créez une fonction `placeNumberRank (... , ... , ...)` qui ajoute un nombre à la fin d'une chaîne de caractères existante en spécifiant la position du chiffre de droite (paramètre *rank*, 35 pour les nombres de la première colonne par exemple). Cette fonction renvoie une chaîne de caractères contenant la chaîne originale à laquelle on a ajouté le nombre à la bonne position. Pour rendre la fonction utilisable pour des entiers, et des réels, le nombre est passé en paramètre sous forme de chaîne de caractères. Conseil : un espace est un caractère, invisible mais un caractère tout de même !

Exemple : la chaîne initiale "Spa Reine" doit devenir

"Spa Reine 56" après un appel de la fonction  
`placeNumberRank (... , ... , ...)`

L'affichage complet doit donc être celui-ci dessous. A la première ligne, le 6 de 56 se trouve au 35<sup>ème</sup> caractère, le 0 de 200 au 40<sup>ème</sup> caractère et le 5 de 45 au 45<sup>ème</sup> caractère.

|                           |    |     |     |
|---------------------------|----|-----|-----|
| Spa Reine                 | 56 | 200 | 55  |
| Bru Plate                 | 42 | 200 | 60  |
| Bru légèrement pétillante | 62 | 200 | 125 |
| Pepsi                     | 45 | 200 | 150 |
| Spa Orange                | 25 | 200 | 140 |
| Schweppes Tonic           | 72 | 200 | 86  |
| Schweppes Agrumes         | 40 | 200 | 47  |
| Lipton Ice Tea            | 48 | 200 | 80  |
| Lipton Ice Tea Pêche      | 24 | 150 | 126 |
| Jus d'orange Looza        | 36 | 200 | 164 |
| Cécémel                   | 15 | 100 | 85  |
| Red Bull                  | 25 | 80  | 23  |

- 24) PJ1Ex24. Le but de cet exercice est de récupérer toute la commande d'une table dans un ArrayList. Le serveur a, sous forme de tableau imprimé à côté de l'ordinateur, la liste de tous les consommables avec leur N° identifiant. Il entre le N° de la consommation par son N° d'identifiant répondant à l'invitation "Entrez le N° de consommable ou Q(Quitter)". Il précise ensuite la quantité de consommations pour ce type de consommation en répondant à l'invitation du programme "Nombre de consommations pour X ? /A(Annuler) /Q(Quitter)" X étant le nom de la consommation dont il vient de saisir le N° d'identifiant. Après une consommation, le serveur a le droit de valider la commande et le message d'invitation devient : "Entrez le N° de consommable ou Q(Quitter) V (Valider le ticket)". Ces opérations d'encodage se succèdent jusqu'au moment où le serveur valide le ticket en entrant la lettre V (majuscule ou minuscule). Le but est de créer une procédure `getOrder (...)` dont la signature est la suivante :

```
public static void getOrder(ArrayList<int[]> ord) { }
```

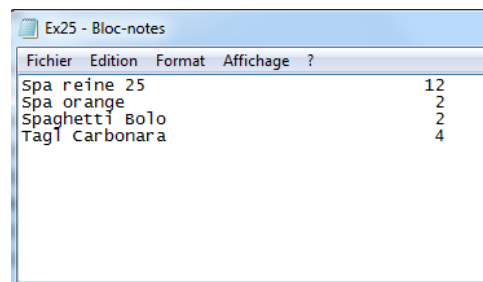
et qui va modifier le contenu de l'ArrayList vierge passé en paramètre. Il s'agit d'un ArrayList de tableaux à une dimension et à 2 cases () contenant, dans la 1<sup>ère</sup> case, l'index de la consommation (à 1 unité près étant donné que la consommation N°1 porte l'index 0 dans les tables de prix et de consommation) et, dans la deuxième, le nombre de consommations.

```
ArrayList<int[]>
```

|            |           |                    |   |
|------------|-----------|--------------------|---|
| <b>ord</b> |           |                    |   |
| 366712642  | → [1, 3]  | Spa Reine 25 Cl    | 3 |
| 257897541  | → [10, 2] | Jus D'orange Looza | 2 |
| 4878915841 | → [36, 2] | Spagh Bolo         | 2 |
| 9851478966 | → [37, 1] | Spagh Carbo        | 1 |

La commande est donc saisie dans un tableau dynamique (ArrayList) et non dans un tableau de taille figée comme c'était le cas pour l'exercice PJ1 EX19.

- 25) PJ1 Ex1-25 Ce programme doit créer un fichier texte contenant le détail de l'ArrayList créé à l'exercice PJ1 Ex1-24 soit par exemple :



| Fichier        | Edition | Format | Affichage | ?  |
|----------------|---------|--------|-----------|----|
| Spa reine 25   |         |        |           | 12 |
| Spa orange     |         |        |           | 2  |
| Spaghetti Bolo |         |        |           | 2  |
| Tag1 Carbonara |         |        |           | 4  |

Les nombres de consommations sont alignés à droite et les unités sont au 30<sup>ème</sup> rang. Réutilisez avantageusement les fonctions déjà développées en les déclarant **public** et en les appelant de cette manière (dans la classe de l'exercice 25) :

```
Ex24.getOrder(order);
Ex23.placeNumberToRank(...) ;
```

- 26) PJ1 Ex1-26 Le programme crée un fichier texte dans un répertoire choisi par l'utilisateur répondant à l'invitation « *Entrez le chemin du répertoire de destination* ». Dans le cas où le chemin n'est pas valide, le message d'erreur « *Chemin non valide* » survient, et le programme repose ensuite la question. Le fichier créé porte un nom variable suivant la date et l'heure de l'exécution du programme. Le nom du fichier doit être au format suivant : 2019-10-21 18h21m16s-457.txt Le contenu du fichier est celui-ci (avec la date heure au moment de la création) :

Café - Brasserie  
Le Passe-Temps  
21/10/2019 18:21

En fin de programme, on affiche le message « *Fichier créé avec succès!* ».  
Créer les fonctions :

- getActualFormattedDateTime(...) qui renvoie la date heure actuelle dans un format choisi et passé en paramètre.
- getUserPathDir (...) qui renvoie un chemin valide saisi par l'utilisateur
- fileCreation(file) qui crée le fichier passé en paramètre et qui renvoie true si la création a réussi.

- 27) PJ1 Ex1-27 A partir d'une interface console, le serveur encode d'abord le N° de table entre 1 et 20, répondant à l'invitation "*Entrez le numéro de table (de 1 à 20)/ Q (Quitter)*". Si le N° est inférieur à 1 ou ne correspond pas à un entier, le programme affiche « *saisie*

*incorrecte* », si le N° est supérieur à 20, le programme affiche « N° de table maximum 20, contactez votre développeur pour augmenter ce nombre ». Il entre ensuite le N° de la consommation par son N° d'identifiant répondant à l'invitation "Entrez le N° de consommable ou Q(Quitter)". Il a sous les yeux, sous forme d'un tableau imprimé, la liste de tous les consommables avec leur N° identifiant. Il précise ensuite la quantité de consommations pour ce type de consommation en répondant à l'invitation du programme "Nombre de consommations pour XXX ? /A(Annuler) Q(Quitter)" XXX étant le nom de la consommation dont il vient de saisir le N° d'identifiant. Le message d'invitation devient ensuite "Entrez le N° de consommable ou Q(Quitter) V(Valider le ticket)". Ces opérations d'encodage se succèdent jusqu'au moment où le serveur valide le ticket en entrant la lettre V (majuscule ou minuscule). Pour simuler la sortie du ticket, on enregistre un nouveau fichier texte sur le disque dur. Le nom de ce fichier est la date et l'heure du moment où le fichier est créé. Le format est de type 2019-10-21 18h21m16s-457.txt de façon à garantir qu'il n'y aura jamais de doublon même si un autre serveur encode un autre ticket quasi au même moment. Le contenu du ticket est tel qu'affiché ci-dessous

| Café-Brasserie      |    |       |        |
|---------------------|----|-------|--------|
| Le passe temps      |    |       |        |
| 26/11/18 17:47      |    |       |        |
| Table:2             |    |       |        |
| Spa reine 25        | 2  | 2,20  | 4,40   |
| Pepsi               | 2  | 2,20  | 4,40   |
| Spaghetti Bolo      | 2  | 10,80 | 21,60  |
| Tagli saum          | 4  | 16,90 | 67,60  |
| Jus d'orange Looza  | 1  | 2,60  | 2,60   |
| -----               |    |       |        |
| TOTAL CONSUMMATIONS | 11 |       |        |
| -----               |    |       |        |
| TOTAL TVA 21,00%    |    | 1,98  | 11,40  |
| TOTAL TVA 12,00%    |    | 9,56  | 89,20  |
| TOTAL TVA 6,00%     |    | 0,00  | 0,00   |
| -----               |    |       |        |
| TOTAL               |    |       | 100,60 |

Le programme se réfère, pour les N° de consommations, dénominations, prix TVAC et TVA à des tableaux internes au programme (voir les tableaux à la fin de l'énoncé)

- L'appui sur Q(Quitter) à n'importe quel message d'invitation arrête le programme sans enregistrement du ticket.
- Il ne doit pas être possible d'entrer un N° de table plus grand que 20. Si le N° de table entré est supérieur à 20, le programme indique "contactez votre développeur pour augmenter le nombre de tables".
- Après la première consommation entrée, il est possible de valider le ticket ; le message d'invitation devient alors "Entrez le N° de consommable. V(Valider), Q(Quitter)".

Pour l'amélioration de l'affichage :

L'unité du nombre de consommations par type de consommation est affichée au 31<sup>ème</sup> caractère.  
 Le cent du prix net pour une consommation est affiché au 40<sup>ème</sup> caractère.  
 Le cent du prix total pour un type de consommation est affiché au 47<sup>ème</sup> caractère

Tableaux à utiliser :

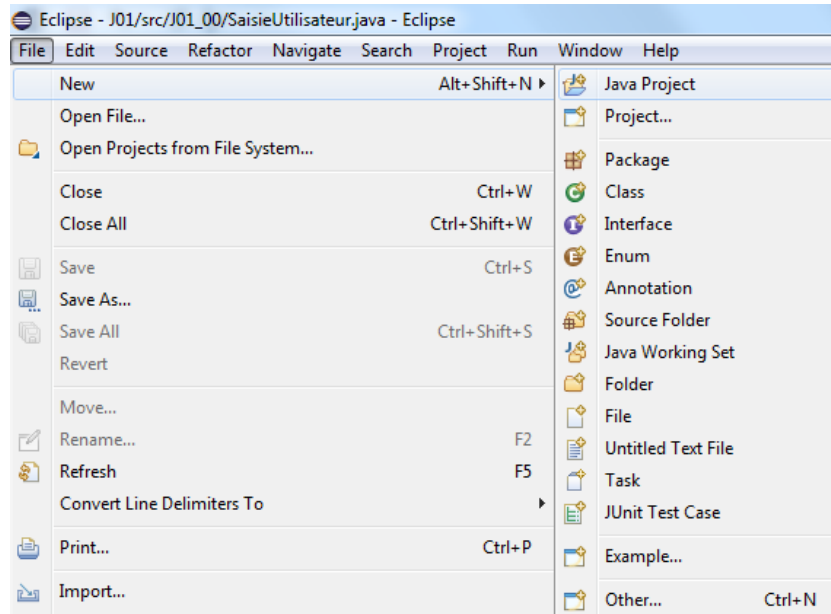
### Tableaux à utiliser :

```
final static String Names[] = {"Spa reine 25 ", "Bru plate 50", "Bru pét 50", "Pepsi", "Spa orange",
 "Schweppes Tonic", "Schweppes Agr", "Ice Tea", "Ice Tea Pêche", "Jus d'orange Looza", "Cécémel", "Red
Bull", "Petit Expresso", "Grand Expresso", "Café décaféiné ", "Lait Russe ", "Thé et infusions", "Irish
Coffee ", "French Coffee ", "Cappuccino", "Cécémel chaud", "Passione Italiano", "Amour Intense",
 "Rhumba Caliente ", "Irish Kisses ", "Cuvée Troll's 25", "Cuvee Troll's 50", "Ambrasse-Temps
25", "Ambrasse-Temps 50 ", "Brasse-Temps Cerises 25", "Brasse-Temps Cerises 50", "La Blanche Ste
Waudru 25", "Blanche Ste Waudru 50", "Brasse-Temps citr 25", "Brasse-Temps citr 50", "Spaghetti Bolo
", "Tagl Carbonara", "Penne poulet baslc ", "Tagl American", "Tagl saum"};
```

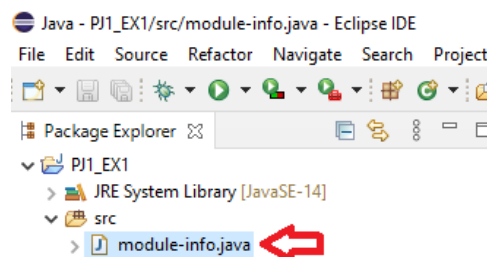
## 4.2. Création d'un projet et d'une classe Java

Réalisons l'exercice PJ1-EX1 ce qui nous permettra de décrire la procédure pour créer un projet et une classe Java :

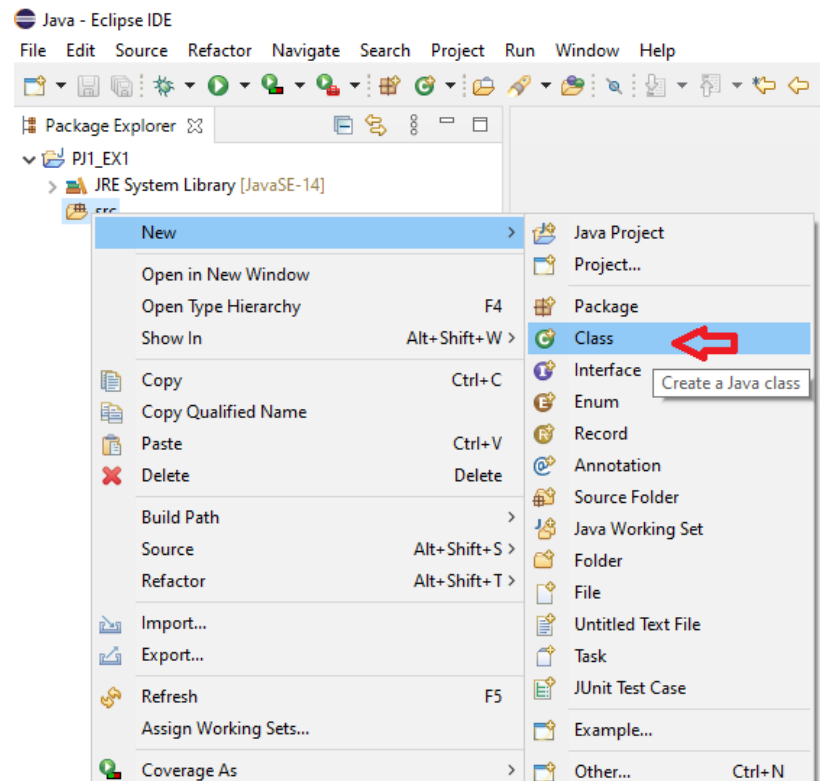
- Création d'un nouveau projet



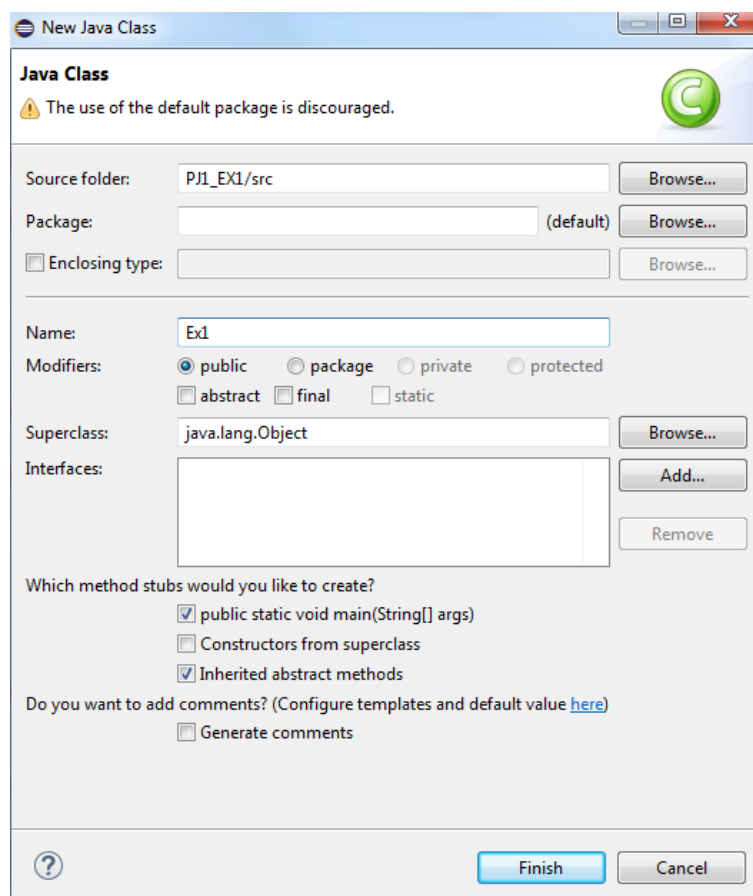
- Dans la fenêtre Project Name, entrez le nom du projet ici PJ1-EX1 qui contiendra tous les programmes des exercices relatifs à cet exercice 1 : création du ticket de caisse.
- Supprimez le fichier *module-info.java* par un click droit, *Supprimer* :



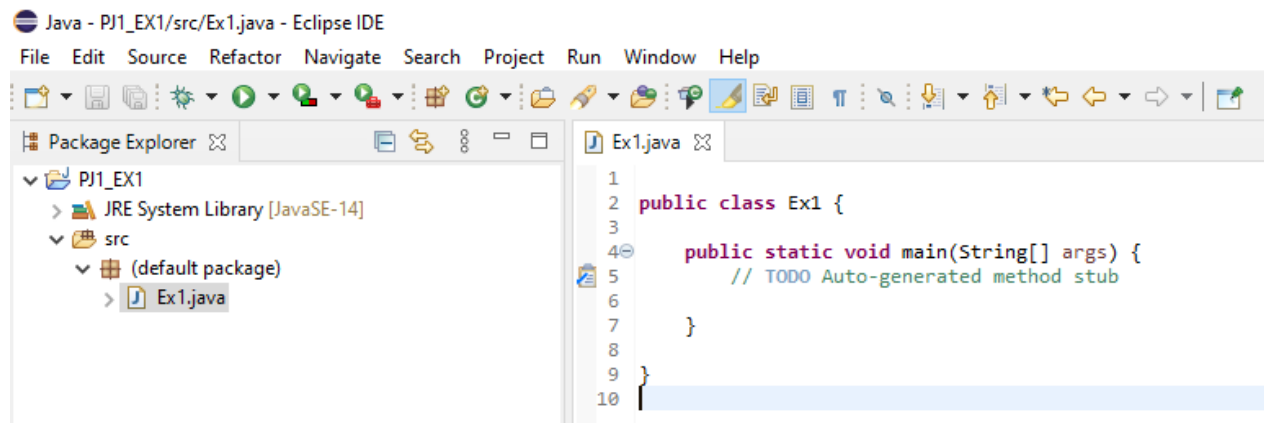
- Créez une nouvelle classe attachée à votre projet java en cliquant droit sur le répertoire "src" attaché à votre projet "PJ1\_EX1", puis en sélectionnant *New*, puis *Class*



- Dans la fenêtre de création de la classe, indiquez le nom de la classe comme *Ex1* (le nom du programme) et cochez "*public static void main*".



- L'environnement de travail apparaît maintenant sous cette forme :



On peut voir qu'il est généré automatiquement des écritures dont on ne pourra pas expliciter chacun des termes actuellement. Disons pour l'instant que le nom de notre classe *Ex1* se trouve après les mots réservés `public class` et est délimitée par une accolade gauche au-dessus et une accolade droite tout en bas.

Les deux barres obliques successives et le texte qui suit en vert sont des commentaires c'est-à-dire du texte qui ne sera pas exécuté mais qui sert au programmeur ou au lecteur du programme à documenter, expliquer le programme. Il peut être supprimé, cela n'affecte en rien l'exécution du programme.

Notre classe contient un seul programme qu'on appelle méthode nommée *main* (programme principal) qui est lui aussi délimité par une accolade gauche et une accolade droite. Il y a donc deux accolades droites qui se suivent et on indiquera les commentaires `//fin main` et `//fin class` pour les différencier. On va ajouter dans le programme *main* notre première instruction

```
System.out.print("Café Brasserie : Le Passe Temps");
```

Dans un premier temps, on peut considérer que ce sont des mots réservés qui permettent d'afficher un texte choisi défini entre les parenthèses et obligatoirement entre guillemets.

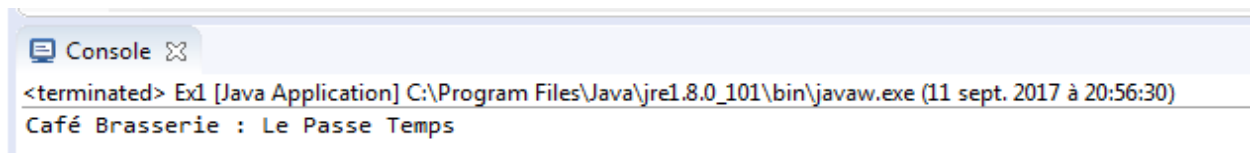
```
public class Ex1 {

 public static void main(String[] args) {

 System.out.print("Café Brasserie : Le Passe Temps");
 } //fin main

} //fin class
```

L'exécution du programme peut se faire via la commande Run dans la barre de menu. Le résultat est alors produit dans la console qui est une fenêtre d'interface.



Lorsqu'on lance le programme par la commande Run, le programme est d'abord ce qu'on appelle **compilé** c'est-à-dire traduit en langage exécutable par la machine ; on parle d'ailleurs de *langage machine*. Si le programme contient des erreurs de syntaxe (c'est-à-dire une faute d'écriture de langage Java, un peu comme s'il contenait des fautes d'orthographe pour un texte), on dit qu'il ne compile pas et ne pourra donc pas être exécuté.

Enlevez par exemple un guillemet et essayez la commande Run. Le programme ne compile plus car on ne respecte pas la syntaxe requise par java : un texte doit être délimité par des guillemets. On remarque d'ailleurs la croix qui apparaît à gauche sur la ligne d'instruction erronée.

```

1
2 public class Ex1 {
3
4 public static void main(String[] args) {
5
6 System.out.print(afé Brasserie : Le Passe Temps");
7 } //fin main
8
9 } //fin class
10

```

C'est là l'intérêt et le rôle de l'interface de développement Eclipse, celui d'assister à la programmation, car en réalité, comme on le verra plus tard, il est possible d'écrire du code java avec un simple éditeur de texte tel que notepad !

Réalisons le deuxième exercice PJ1Ex1-2 la seule différence est de vouloir écrire le texte en deux lignes et non une seule. Pour ce faire, essayez de placer plusieurs instructions

```

System.out.print ("Du texte") ;
System.out.println("Encore Du texte") ;

```

les unes à la suite des autres pour en comprendre la nuance. Essayez également de placer `\n` à l'intérieur du texte comme indiqué ci-dessous

```

System.out.print ("Du texte \nEncore du texte") ;

```

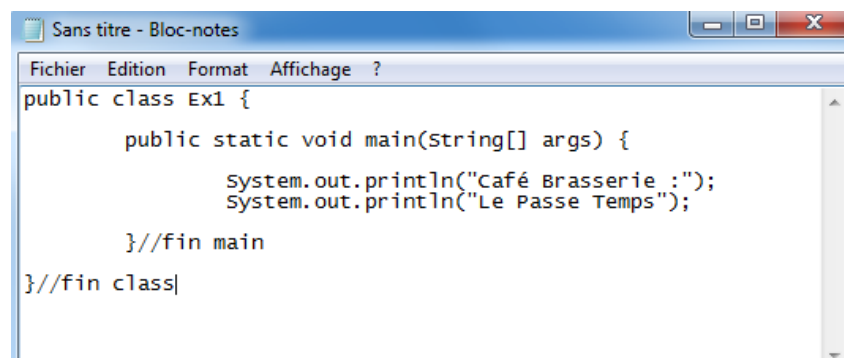
Réalisez ensuite l'exercice PJ1Ex1-2 de quatre manières différentes.

### 4.3. Exécution d'un programme java.

Nous avons vu que l'édition du programme pouvait être fait dans l'environnement d'Eclipse et que l'exécution du programme pouvait être visualisé dans la fenêtre console.

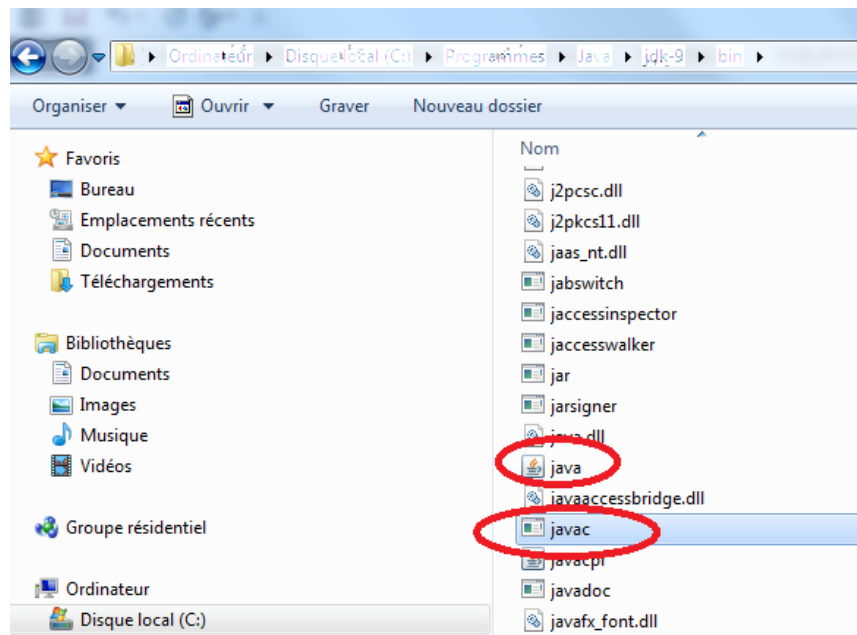
Voyons maintenant comment utiliser un éditeur de texte pour encoder le programme et la fenêtre de commande pour visualiser son exécution. Cet exercice nous permettra d'introduire les notions de code source, byte code, machine virtuelle, code machine, compilation et exécution.

- Lancer l'éditeur de texte Notepad, copier-coller le code de votre programme réalisé lors des exercices précédents.

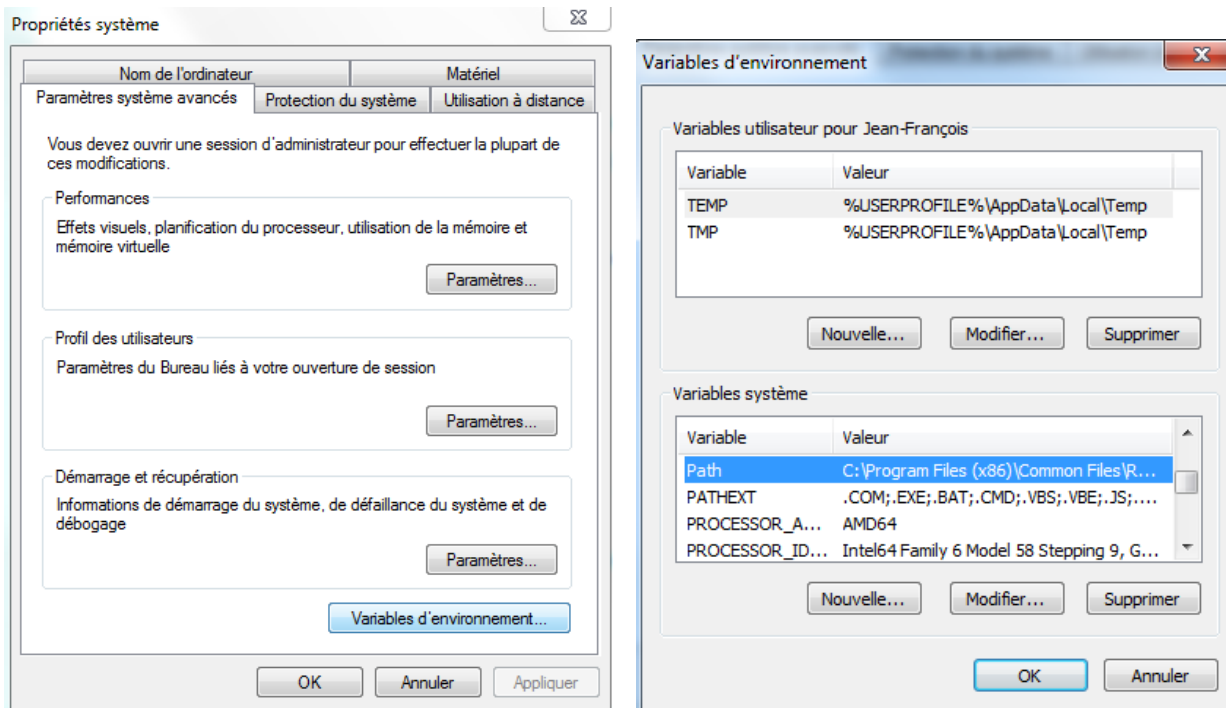




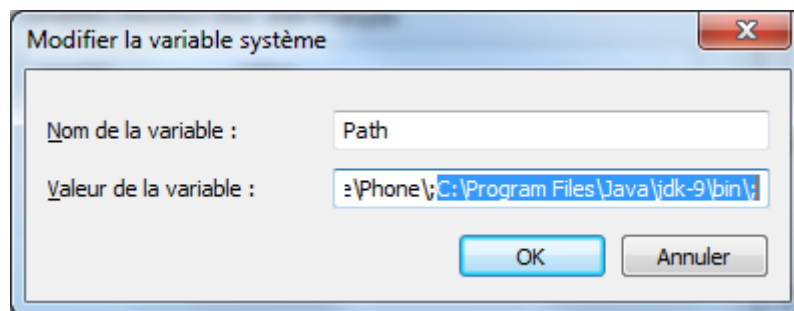
- Créez un dossier nommé JAVA PROGRAMS dans la racine de votre disque dur puis enregistrez-sous en donnant comme nom MonProgrammeJava.java donc en changeant l'extension de fichier initiale (.txt).
- On va permettre à la fenêtre de commande d'utiliser les programmes de compilation java à savoir *javac* et le programme d'exécution *java*. Pour ce faire, il faut repérer où se trouvent ces deux programmes et ajouter le chemin dans les variables d'environnement windows. Le chemin peut varier suivant le choix fait lors de votre installation, mais ces fichiers se trouvent habituellement dans C:\Program Files\Java\jdk-9\bin



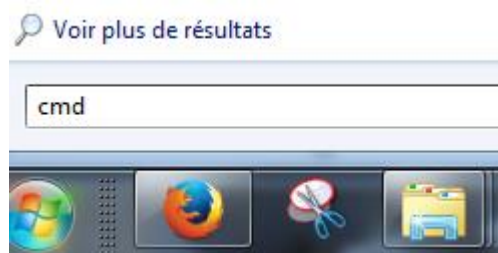
- Cliquez droit sur ces fichiers, puis propriétés pour copier le chemin.
- Allez dans le panneau de configuration windows, Système et sécurité, Système, puis paramètres avancés.
- Cliquez sur *Variables d'environnement* puis recherchez la variable *Path*, puis cliquez sur *Modifier*.



- Ajoutez le chemin copié précédemment à la ligne existante, juste après le dernier ; et ajoutez \; à la fin.



- Ouvrez une fenêtre de commande en tapant cmd depuis le menu démarrer windows



- Placez-vous dans le répertoire JAVA PROGRAMS en utilisant les commandes D : (ou C : Puis cd JAVA PROGRAMS.
- Entrez la commande `javac MonProgrammeJava.java` pour compiler le programme java. *Javac* signifie java compile.

```

C:\Windows\system32\cmd.exe
Erreur : impossible de trouver ou de charger la classe principale PROGRAMS
Causé par : java.lang.ClassNotFoundException: PROGRAMS

D:\>cd JAVA PROGRAMS

D:\JAVA PROGRAMS> javac MonProgrammeJava.java
MonProgrammeJava.java:1: error: class Ex1 is public, should be declared in a file named Ex1.java
public class Ex1 {
 ^
1 error

D:\JAVA PROGRAMS> javac MonProgrammeJava.java

D:\JAVA PROGRAMS> java MonProgrammeJava
Erreur : impossible de trouver ou de charger la classe principale MonProgrammeJava
Causé par : java.lang.NoClassDefFoundError: MonProgrammeJava (wrong name: MonProgrammeJava)

D:\JAVA PROGRAMS> java MonProgrammeJava
Café Brasserie :
Le Passe Temps

D:\JAVA PROGRAMS>

```

- Il refuse de compiler car le nom de classe Ex1 défini dans le fichier texte est différent du nom du fichier. Modifiez le nom de la classe : remplacez Ex1 par MonProgrammeJava dans le fichier texte et écrasez l'ancien fichier MonProgrammeJava.java.
- Retestez la commande *javac MonProgrammeJava.java*. Si la compilation a réussi, un fichier MonProgrammeJava de type fichier CLASS a été créé dans le répertoire JAVA PROGRAMS. C'est ce qu'on appelle le **byte code Java**. Remarquez qu'un même fichier avait été créé par Eclipse dans le répertoire de votre programme.
- Testez l'exécution du programme par la commande *java MonProgrammeJava*
- Ajoutez une instruction d'affichage dans le fichier texte (.java) :
 

```

System.out.println("Café Brasserie :");
System.out.println("Le Passe Temps");
System.out.println("salut");

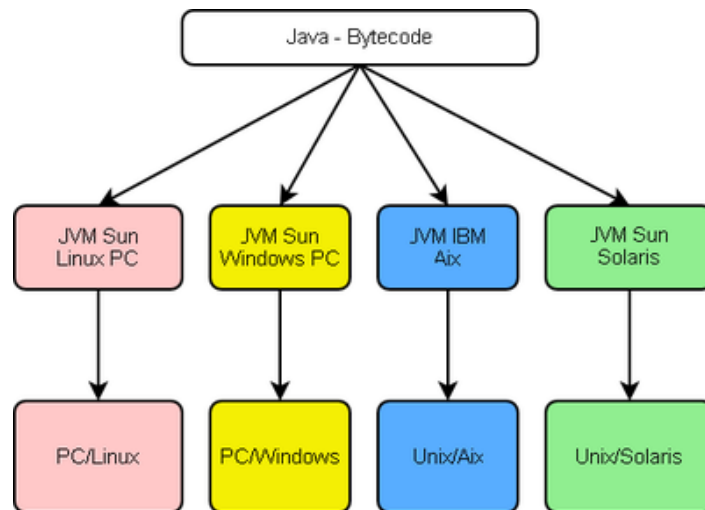
```
- Relancez l'exécution par la commande *java MonProgrammeJava* et remarquez que l'exécution n'est pas modifiée. Il faut compiler le nouveau programme pour que la modification soit effective. Compilez donc une nouvelle fois par *javac MonProgrammeJava.java* puis retestez l'exécution par la commande *java MonProgrammeJava*. Constatez que le programme a bien été modifié cette fois.

#### 4.3.1. Byte Code – Machine Virtuelle (JVM)

- Les instructions encodées, lisibles et compréhensibles par le programmeur s'appellent le **code source**.
- La compilation crée ce qu'on appelle un **bytecode** en Java, un code intermédiaire qui n'est pas encore un code machine directement exécutable par le processeur. Il s'agit d'un fichier de type .CLASS qui a été créé quand on a lancé la compilation depuis la fenêtre de commande.

- Le bytecode Java est un bytecode destiné à regrouper des instructions exécutables par une machine virtuelle java (JVM). Il désigne un flux d'octets binaire. Ce flux est habituellement le résultat de la compilation d'un code source, ce code source n'étant pas obligatoirement écrit en langage Java. Ce bytecode peut être exécuté sous de nombreux systèmes d'exploitation par une machine virtuelle Java appropriée et spécifique à la machine et au système d'exploitation.

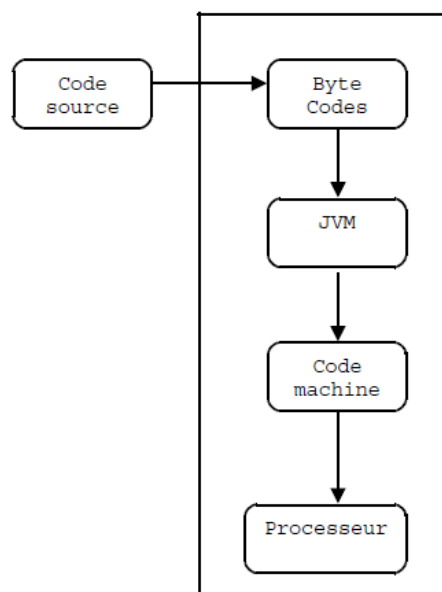
La **machine virtuelle Java** (en anglais *Java virtual machine*, abr. **JVM**) est un appareil informatique fictif qui exécute des programmes compilés sous forme de bytecode Java.



Architecture générale : illustration du slogan *Compile once, run everywhere*.

(source wikipédia)

La machine virtuelle Java **permet aux applications Java compilées en bytecode de produire les mêmes résultats quelle que soit la plate-forme, tant que celle-ci est pourvue de la machine virtuelle Java adéquate**. On parle alors de portabilité d'un programme, c'est-à-dire les conditions matérielles et logicielles qui permettent à un programme de pouvoir fonctionner. Les programmes java sont donc rendus portables par les machines virtuelles java.



*Schéma de principe de la programmation à l'exécution en Java.*

Le même programme compilé (byte code) peut donc fonctionner sur n'importe quelle plate-forme et sous n'importe quel système d'exploitation possédant une machine Java virtuelle (JVM). La machine virtuelle Java appelée **interpréteur** Java prend les 'byte codes' et convertit leurs instructions en commandes intelligibles pour un système d'exploitation (code machine). La machine virtuelle cherche une fonction publique 'main'. Si elle la trouve, elle l'exécute, sinon elle indique une erreur.

#### 4.4. Premier programme "élaboré"

Réalisons l'exercice PJ1\_EX3 dont voici l'énoncé :

PJ1Ex1-3 Le programme calcule le prix total pour un type de consommation en fonction du nombre d'unités et du prix unitaire. Le nombre d'unités et le prix unitaire sont d'abord initialisés à des valeurs nulles pour ensuite être affectés à des valeurs quelconques.

Cet exercice simple permet d'introduire les notions de variable, mémoire, affectation, type de données, conversions de type, (entiers, flottants), d'opérateurs arithmétiques, de séquençement des instructions et de la visualisation de celui-ci en vue de debugger. Rien que ça !

Créez une nouvelle classe nommée Ex3 dans le projet PJ1\_EX1, entrez et exécutez le code suivant :

```
public class Ex3 {
 public static void main(String[] args) {

 int intQuantitConso=0; //quantité de consommations commandées
 double dblPrixUnitaire=0.0; //prix unitaire de la consommation
 double dblPrixTotParConso=0.0 ;//prix total par type de consommation

 intQuantitConso = 3;
 dblPrixUnitaire = 2.5;
 dblPrixTotParConso = dblPrixUnitaire * intQuantitConso;

 System.out.print("Prix total pour cette consommation: " + dblPrixTotParConso +
 "€");
 } //fin main
} //fin class
```

Remparez ensuite la ligne de code `dblPrixUnitaire = 2.5;` par `dblPrixUnitaire = 10/4;` et examinez le résultat.

##### 4.4.1. Variables et affectations

`intQuantitConso`, `dblPrixUnitaire`, `dblPrixTotParConso` sont appelées des **variables**, c'est-à-dire que, comme en mathématique, une variable est une représentation symbolique contenant une valeur qui peut être quelconque ou presque. Pas tout à fait quelconque car elle doit correspondre tout de même au type de cette variable. Par exemple, si on se rappelle du cours de mathématique et que l'on voit une équation comme  $x+3=y-2$ , le type de  $x$  et  $y$  ne peut être que numérique.

Le type de la variable défini par les mots réservés qui précèdent les noms de variables. Dans l'exemple, deux types de variables sont présents : le type entier (en math, on dirait appartenant à l'ensemble de entiers naturels  $\mathbb{Z}$ ) désigné par le mot réservé java **int** et le type nombre à virgule flottante double précision (en math, on dirait appartenant à l'ensemble de réels  $\mathbb{R}$ ) désigné par le mot réservé java **double**.

Le nom des variables est choisi quasi librement par le programmeur mais doit commencer par une lettre. Le nom doit cependant être choisi de façon à faciliter la relecture du programme. On évitera donc `Variable1`, `variable2`, `Nombre`,.... Remarquez que le java est **case sensitive** c'est à dire qu'il fait la distinction entre minuscule et majuscule. Ainsi `dblprixunitaire` et `dblPrixUnitaire` sont deux variables distinctes.

Certains programmeurs apprécient de mettre un préfixe comme `dbl` ou `int` au nom de la variable de façon à identifier immédiatement le type de variable quand celle-ci est utilisée beaucoup plus loin que la déclaration.

Les premières lignes de codes ci-dessous où sont indiqués le nom des variables et leur type s'appellent **la déclaration des variables**.

```
int intQuantitConso = 0; //quantité de consommations commandées
double dblPrixUnitaire=0.0; //prix unitaire de la consommation
double dblPrixTotParConso=0.0; //prix total par type de consommation
```

La déclaration se fait généralement en début de programme même si Java autorise de déclarer une variable plus bas dans le programme pour autant que la variable n'ait pas dû être utilisée avant. Il arrive fréquemment qu'une variable soit déclarée juste avant son utilisation ainsi la mémoire est allouée uniquement si c'est nécessaire. Cependant, cela rend le code moins lisible car les déclarations ne se trouvent plus toutes au même endroit.

La variable `intQuantitConso` est suivie de `=0`. On dit que la variable est initialisée à sa déclaration, on lui donne une valeur de 0 ici en l'occurrence mais il est à noter que la valeur aurait pu être différente de 0. Dans un premier temps, on peut dire qu'en Java, il est obligatoire d'initialiser les variables qui sont utilisées au cours du programme.

L'exécution d'un programme est séquentielle c'est-à-dire que les instructions sont exécutées les unes après les autres, ligne par ligne. Les instructions sont délimitées par un point virgule (;) obligatoire. Une erreur classique du programmeur débutant est d'oublier de placer ce ; à la fin de chaque instruction.

Au moment de la déclaration, le programme va réserver un emplacement mémoire où le contenu numérique de la variable va être stocké.

Les lignes de code

```
intQuantitConso = 3;
dblPrixUnitaire = 2.5;
```

sont ce qu'on appelle des **affectations**. On affecte, c'est à dire on écrit les valeurs 3 et 2.5 respectivement dans les zones mémoires correspondantes aux variable `intQuantitConso` et `dblPrixUnitaire` qui contenaient avant les valeurs nulles donnée lors de l'initialisation.

Analysons la ligne de code :

```
dblPrixTotParConso = dblPrixUnitaire * intQuantitConso;
```

Le programme lit le contenu numérique de la mémoire des variables `dblPrixUnitaire`, `intQuantitConso` et les multiplie. Le résultat de la multiplication est stocké dans la zone mémoire allouée à la variable `dblPrixTotParConso`.

On affiche ensuite, à la console, le message "Prix total pour cette consommation : 7,5€" grâce à l'instruction.

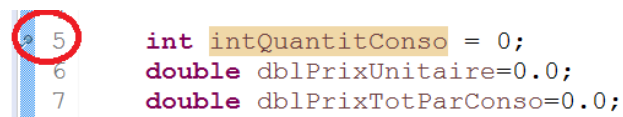
```
System.out.print("Prix total pour cette consommation: " + dblPrixTotParConso +
"€"); }
```

Remarquons que la variable `dblPrixTotParConso` pourtant numérique est ici considérée et même convertie en texte par la **méthode** `print()`. Aussi, les deux `+` ne sont pas du tout des opérateurs arithmétiques d'addition mais ce qu'on appelle des opérateurs de concaténation.

La **concaténation** consiste à assembler plusieurs parties de textes ensemble. Ces parties de textes sont appelées en programmation **chaînes de caractères** mais nous y reviendrons plus loin dans le cours.

#### 4.4.2. Séquencement des instructions - Debug

Il est très important de visualiser l'exécution séquentielle d'un programme pour pouvoir comprendre son fonctionnement, déceler et corriger (debugger) des anomalies éventuelles (bugs). L'exécution classique d'un programme (Run) permet de constater le résultat final mais pas l'évolution séquentielle nécessaire à un débogage efficace. Pour lancer le **mode debug**, l'exécution pas à pas du programme et la visualisation des variables, placez un point d'arrêt (**break point**) sur la première ligne de code. Un breakpoint peut être placé (ou retiré) en double cliquant à gauche du nombre désignant la ligne d'instruction, au niveau du liseré bleu.

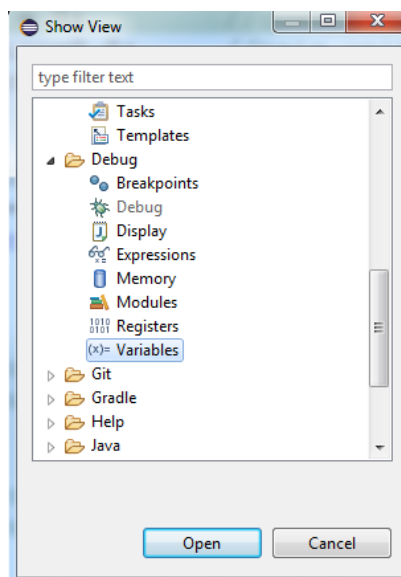


*Insertion d'un point d'arrêt breakpoint*

Appuyez ensuite sur **F11** pour lancer le **mode debug** ou via le menu *Run, Debug*.

Lorsqu'un breakpoint est placé, l'exécution des instructions se fait normalement (automatiquement en une seule fois) pour les lignes qui précèdent le breakpoint puis l'exécution s'arrête à la ligne où se trouve le breakpoint.

Si la fenêtre des variables n'est pas présente, ajoutez la manuellement via le menu *Window, Show View, Other*, puis sélectionnez le répertoire *Debug* puis *Variables*.



*Ajout de la fenêtre de visualisation des variables*


| Debug (x)= Variables   |                   |
|------------------------|-------------------|
| Name                   | Value             |
| no method return value |                   |
| args                   | String[0] (id=16) |
| intQuantitConso        | 0                 |
| dblPrixUnitaire        | 0.0               |
| dblPrixTotParConso     | 0.0               |

*Fenêtre de visualisation des variables*



Appuyez sur **F6** à chaque fois que vous voulez que la ligne d’instruction en surbrillance soit exécutée.

Pour chaque ligne de programme, estimez à l’avance la valeur future de chaque variable puis appuyez sur F6 pour constater les changements et l’adéquation avec ce que vous aviez prévu. Dans le futur, et pour vos prochains programmes personnels, cette façon de procéder sera la meilleure qui soit pour déceler un bug dans votre programme.

| 1. Placer un point d’arrêt                                                        | 2. Passer en mode debug | 3. Exécuter pas à pas |
|-----------------------------------------------------------------------------------|-------------------------|-----------------------|
|  | <b>F11</b>              | <b>F6</b>             |

*Méthode d’exécution manuelle, pas à pas*

#### 4.4.3. Importance des types de donnée

Remplacez l’instruction `dblPrixUnitaire = 2.5;` par `dblPrixUnitaire = 10/4;`

Quel problème observez-vous sur le résultat final, comment l’expliquer et le résoudre ?

Si nos connaissances en mathématiques nous permettent de savoir ce qu’est un nombre entier et un nombre à virgule, il existe également en Java les types `byte`, `short` et `long` en plus du type `int` comme nombres entiers. Il existe également le type `float` comme nombre à virgule flottante en plus du type `double`. Par ailleurs, on pourrait se demander l’intérêt d’utiliser des entiers puisque, à priori, qui peut le plus peut le moins et une variable à virgule qui peut contenir la valeur 3.2589 peut également contenir la valeur 3 ou 3.0. Il est donc nécessaire d’ouvrir une plus ou moins large parenthèse sur les types de données et le codage de l’information en informatique pour comprendre les spécificités de chaque type de donnée et pouvoir choisir le bon type de donnée pour chaque variable de nos programmes.

#### 4.5. Le codage de l’information

En tant qu’être humain, nous traitons quantité d’informations via nos sens et notre cerveau. Ces informations peuvent être une couleur, une photo, une vidéo, un son, de la musique, une lettre, un chiffre, un nombre, un mot, un texte, une température, une vitesse, ... Pour pouvoir représenter, stocker, transmettre toutes ces informations, nos appareils électroniques (PC, disques dur, modems, câbles) ont un système de codage c’est-à-dire un langage qui leur est propre. Ce langage s’appelle *le langage ou codage binaire*. Toute information dite numérique est codée sous forme de nombres binaires c’est-à-dire sous forme d’une succession de 0 ou 1 (ex : 100010100). Ce « langage » est le seul compréhensible par l’électronique, car, dans un circuit, il n’y a que deux possibilités : soit le courant passe (état 1) ou le courant ne passe pas (état 0), il y a de la tension (état 1) ou il n’y a pas de tension (état 0).



### 4.5.1. Les nombres binaires

On a dit qu'en informatique, toute information est stockée sous forme de nombres binaires.

**Un nombre binaire est une succession de 0 ou de 1 appelés bits (binary digit).**

**Le bit est la plus petite unité d'information manipulable par un système numérique. Il ne peut avoir que deux états : l'état logique 0 ou 1.**

Exemples :

- binaire 101001 représente le nombre décimal 41
- le nombre binaire 1110000 représente la lettre p

#### 4.5.1.1. Conversion binaire vers décimal de nombres entiers

Voyons comment un nombre décimal est représenté en codage binaire qu'on appelle aussi codage de base 2. Le nombre décimal correspondant au nombre binaire  $a_n \dots a_1 a_0$  est donné par la formule :

$$a_n \dots a_3 a_2 a_1 a_0 = a_n \cdot 2^n + \dots + a_3 \cdot 2^3 + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0$$

**Exemple** : soit le nombre binaire 101001 à convertir en décimal.

Après développement suivant la formule ci-dessus, on obtient un résultat de 41.

$$(101001)_2 = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 32 + 8 + 1 = 41$$

Le calcul d'un nombre binaire revient à additionner des puissances de 2.

Exercices : convertir en décimal les nombres binaires suivants :

- 1)  $(0111)_2 =$
- 2)  $(1010101)_2 =$
- 3)  $(10110011)_2 =$
- 4)  $(00000000)_2 =$
- 5)  $(10111011100000)_2 =$
- 6)  $(1010101001010000)_2 =$
- 7)  $(11111111)_2 =$

Vérifiez vos réponses en testant le code ci-dessous (creez une classe ExTypeDonnees1, copiez, collez le code). Le but de cet exercice est de comprendre les différences entre les types **byte**, **short**, **int** et **long** java. Le code prévoit seulement l'affichage (`System.out.println()`) des variables relatives à l'exercice 1), complétez le code pour afficher les variables jusqu'à l'exercice 7). Remarquez l'équivalence de l'affichage pour certaines variables et le comportement du compilateur pour d'autres. Lorsque le compilateur refuse de compiler une instruction, placez là en commentaire (`//`) et tentez d'en identifier la cause. Identifiez la plage des valeurs possibles pour un **byte**.

```
public class ExTypeDonnees1 {

 public static void main(String[] args) {

 int intEx1 = 0b0111;
 short shEx1 = 0b0111;
 byte byEx1 = 0b0111;
```

```

 int intEx2 = 0b1010101;
 short shEx2 = 0b1010101;
 byte byEx2 = 0b1010101;

 int intEx3 = 0b10110011;
 short shEx3 = 0b10110011;
 byte byEx3 = 0b10110011;
 byte by2Ex3 = (byte)0b10110011;

 int intEx4 = 0b000000000;
 short shEx4 = 0b000000000;
 byte byEx4 = 0b000000000;

 int intEx5 = 0b101110111000000;
 short shEx5 = 0b101110111000000;
 byte byEx5 = 0b101110111000000;

 long lngEx6 = 0b1010101001010000;
 int intEx6 = 0b1010101001010000;
 short shEx6 = 0b1010101001010000;
 byte byEx6 = 0b1010101001010000;

 int intEx8 = 2147483647;
 int intEx8 = 2147483648;
 long lngEx8 = 2147483648L;

 byte intEx9a = 126;
 byte intEx9b = 127;
 byte byEx9c = 128;

 byte intEx9d = -129;
 byte intEx9e = -128;
 byte lngEx9f = -127;

 System.out.println ("intEx1 " + intEx1);
 System.out.println ("shEx1 " + shEx1);
 System.out.println ("byEx1 " + byEx1);
 //compléter ici la suite de l'affichage

} //fin main

} //fin class

```

#### 4.5.1.2. Nombre maximum et nombre de combinaisons

Dans l'ex 7) du §4.5.1.1, on a constaté que le plus grand nombre binaire sur 8 bits était 1111 1111 = 255 et qu'il y a donc 256 valeurs possibles (de 0 à 255).

De façon générale, on a :

**Le nombre entier positif maximum d'un nombre binaire codé sur N bits =  $2^N - 1$**

**Un nombre binaire à N bits possède  $2^N$  combinaisons possibles**

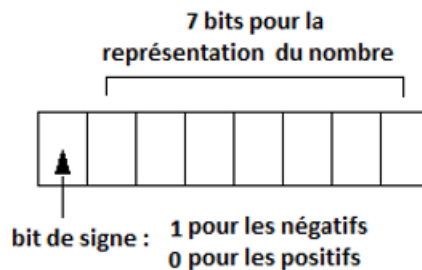
Exemples :

- Pour 8 bits, on a comme maximum  $2^8 - 1 = 255$  ou 256 valeurs possibles
- Pour 16 bits, on a comme maximum  $2^{16} - 1 = 65535$  ou 65536 valeurs possibles
- Pour 32 bits, on a comme maximum  $2^{32} - 1 = 4\,294\,967\,295$  ou 4294967296 possibilités
- Pour 64 bits, on a comme maximum  $2^{64} - 1$  ou  $2^{64}$  possibilités

#### 4.5.1.3. Nombres signés

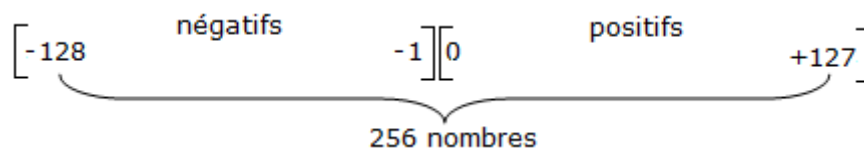
Sachant qu'un byte occupe 8 bits en mémoire, comment comprendre le fait que sa plage de valeurs possibles va de -128 à 127 alors que le nombre maximum en 8 bits est 255 ?

Jusqu'à présent, on a considéré uniquement des nombres non signés, c'est-à-dire des entiers positifs. En réalité, les variables java peuvent contenir des entiers positifs ou négatifs. Or, pour stocker un entier positif ou négatif, on a besoin de réserver le bit de poids fort (le plus à gauche, appelé MSB) comme bit de signe. Il sera à 0 si l'entier est positif, à 1 si l'entier est négatif. Les 7 autres bits servent pour représenter la valeur absolue du nombre entier.



Représentation d'un entier signé dans un octet.

Pour un octet non signé, on peut aller de 0 à +255 ( $2^8 - 1$ ) et on a 256 ( $2^8$ ) combinaisons possibles. En nombres signés, il n'y a plus que 7 bits pour le nombre et on a donc  $2^7 = 128$  combinaisons possibles de 0 à 127 en positif et de -1 à -128 en négatif.



**La plage des nombres possibles d'un nombre signé codé sur N bits, s'étend de**

**$-2^{(N-1)}$  à  $+2^{(N-1)} - 1$**

En 8 bits, on va de  $-2^{8-1}$  à  $+2^{8-1} - 1 = -128$  à  $+127$

En 16 bits, on va de           à           =           à

En 32 bits, on va de           à           =           à

En Java, les entiers de type int sont codés sur 32 bits alors qu'ils l'étaient sur 16 bits en C.

Il faut garder à l'esprit qu'on ne peut dépasser ces limites lors de l'exécution du programme sans quoi cela crée un bug informatique, un arrêt de celui-ci. Pour s'en rendre compte, testez le programme suivant en ayant estimé auparavant, la valeur correcte de x2. Proposez une modification du code.

```
public class Test {
 public static void main(String[] args) {
 int x1 = 2147000000;
 int x2=0;

 x2 = x1*2;
 System.out.println(x1);
 System.out.println(x2);
 }
}
```

### 4.5.2. Synthèse sur les nombres et types entiers en java

En Java, les types de données entiers primitifs ont les caractéristiques reprises dans le tableau ci-dessous. Il est à noter qu'ils sont tous signés (alors qu'il existe des types non signés dans d'autres langages comme le C).

| Type de donnée Java | Signification     | Taille (en octets) | Plage de valeurs acceptées                                  |
|---------------------|-------------------|--------------------|-------------------------------------------------------------|
| <code>byte</code>   | Entier très court | 1 (8 bits)         | -128 à 127                                                  |
| <code>short</code>  | Entier court      | 2 (16 bits)        | -32768 à 32767                                              |
| <code>int</code>    | Entier            | 4 (32 bits)        | -2 147 483 648 à +2 147 483 647                             |
| <code>long</code>   | Entier long       | 8 (64 bits)        | - 9 223 372 036 854 775 808 à - 9 223 372 036 854 775 807 à |

Ce tableau est uniquement valable pour le Java. Pour d'autres langages de programmation (C, C++), le type `int` stocke les valeurs sur 16 bits.

La tendance de Java est d'utiliser le `int`. En effet, lorsqu'on écrit une valeur numérique, elle se trouve par défaut codée sur un `int` 32 bits. Economiser de la mémoire et utiliser le `short`, le `byte` s'ils sont suffisants n'est donc plus en vogue avec Java puisqu'il vous impose alors des contraintes de conversions (voir §4.7). Il est à remarquer aussi qu'on utilise le `int` même si on doit représenter uniquement des entiers positifs. Ce n'est pas forcément le cas dans d'autres langages comme le C++ où le type (unsigned int) existe par exemple et permet de garder le bit de signe pour la valeur numérique, donc d'augmenter la capacité de représentation du 16 bits (-32768 à +32767 pour un entier signé 16 bits, 0 à 65535 pour un non signé 16 bits).

### 4.5.3. Représentation des nombres à virgule flottante (norme IEEE75)

#### 4.5.3.1. Conversion binaire vers décimal de nombres fractionnaires

Les nombres fractionnaires sont écrits en binaire sous la forme :

$$a_n a_{n-1} \dots a_2 a_1 a_0, a_{-1} a_{-2} \dots a_{-m} = a_n \cdot 2^n + a_{n-1} \cdot 2^{n-1} + \dots + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0 + a_{-1} \cdot 2^{-1} + \dots + a_{-m} \cdot 2^{-m}$$

**Exemple :** soit  $(1001101,100101)_2$  à convertir en nombre décimal.

Après développement, le résultat est 77,578125

**Exercices :** convertir en décimal les nombres binaires suivants :

1)  $(1001,011)_2 =$

2)  $(10110,010011)_2 =$

#### 4.5.3.2. Conversion décimal vers binaire de nombres fractionnaires

La méthode consiste en une suite de multiplication par 2 de la partie fractionnaire.

- Quand le résultat du produit est plus petit que 1, on le garde à la ligne suivant pour la multiplication suivante.
- Quand le résultat du produit est plus grand que 1, on garde uniquement la partie fractionnaire pour la multiplication suivante.
- Un produit plus petit que 1 donne un bit 0, plus grand ou égal à 1 donne un bit 1.

Les multiplications s'arrêtent quand on arrive à 1 ou quand la précision désirée est suffisante.

**Exemple :** soit le nombre 77,578125 dont on veut donc transformer la partie fractionnaire 0,578125 en nombres binaires.

$$0,578125 \times 2 = 1,15625 \quad \rightarrow \quad 1$$

$$0,15625 \times 2 = 0,3125 \quad \rightarrow \quad 0$$

$$0,3125 \times 2 = 0,625 \quad \rightarrow \quad 0$$

$$0,625 \times 2 = 1,25 \quad \rightarrow \quad 1$$

$$0,25 \times 2 = 0,5 \quad \rightarrow \quad 0$$

$$0,5 \times 2 = 1 \quad \rightarrow \quad 1$$

$$\Rightarrow 0,578125 = 0,100101$$

**Exercices :** convertir en binaire les nombres décimaux non entiers suivants :

$$1) (7,265625)_{10} =$$

$$2) (17,641)_{10} =$$

$$3) (9,9999999...)_{10} =$$

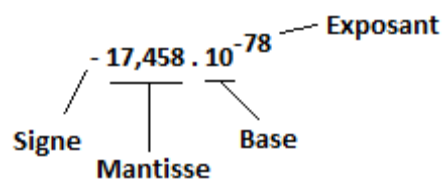
**Contrairement aux entiers, les nombres fractionnaires ne peuvent pas tous être représentés de manière exacte en codage binaire.**

#### 4.5.3.3. Codage des nombre réels

Un nombre réel  $R$  comportant des chiffres décimaux (derrière la virgule) peut être représenté en notation scientifique sous la forme :

|                                |                                                                                                        |
|--------------------------------|--------------------------------------------------------------------------------------------------------|
| $R = (-1)^S \cdot M \cdot b^E$ | Avec $R$ : nombre réel<br>$S$ : 1 si le nombre est $<0$ , 0 si le nombre est $>0$<br>$M$ : la mantisse |
|--------------------------------|--------------------------------------------------------------------------------------------------------|

**Exemple :**



Le nombre décimal  $\pi$  peut être représenté sous la forme :

$31,4159 \cdot 10^{-1}$  ou  $3,14159 \cdot 10^0$  ou  $0,31415 \cdot 10^1$  ou  $0,03141 \cdot 10^2$  ou  $0,00314 \cdot 10^3, \dots$

On voit que la position de la virgule est "flottante", elle varie en fonction de l'exposant de la base 10. Il y a donc plusieurs façons de représenter le nombre  $\pi$  suivant la position de cette virgule flottante.

L'**IEEE 754** est une norme pour la représentation des nombres à virgule flottante en binaires. Elle est la norme la plus employée actuellement pour le calcul des nombres à virgule flottante dans le domaine informatique.

La **norme IEEE754** a fixé un format d'écriture normalisé pour un nombre réel :

|                                                                                  |
|----------------------------------------------------------------------------------|
| $R = (-1)^S \times 1, M_{\text{codé}} \times 2^{E_{\text{codé}} - \text{Biais}}$ |
|----------------------------------------------------------------------------------|

Avec :

- $S$  est le bit de signe  $s_i=0 \Rightarrow R \text{ est } >0$ ,  $s_i=1 \Rightarrow R < 0$
- La base  $b$  utilisée est **2**.
- la mantisse  $M$  composée de  $n+1$  chiffres (binaires puisqu'on est en base 2) est représentée sous une forme normalisée (unique).

$$\text{Mantisse} = 1, M_1 M_2 \dots M_n$$

avec un seul chiffre avant la virgule, toujours le bit 1, qui ne sera donc pas codé (stocké) en machine, on l'appelle bit caché (hidden bit).

- L'**exposant** =  $E_{\text{codé en machine}} - \text{Biais}$ . L'exposant codé en machine sera un nombre entier positif sur 8 ou 11 bits suivant la précision. Le biais est une valeur constante 127 ou 1023

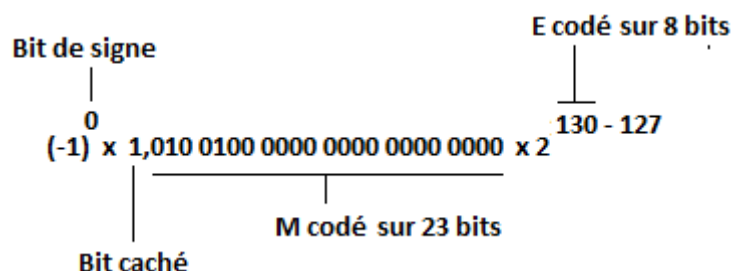
suivant la précision. Le biais permet d'avoir des exposants négatifs sinon on pourrait représenter uniquement de grands nombres.

La norme IEEE754 définit 3 formats de base suivant le nombre de bits mémoires qu'ils occupent :

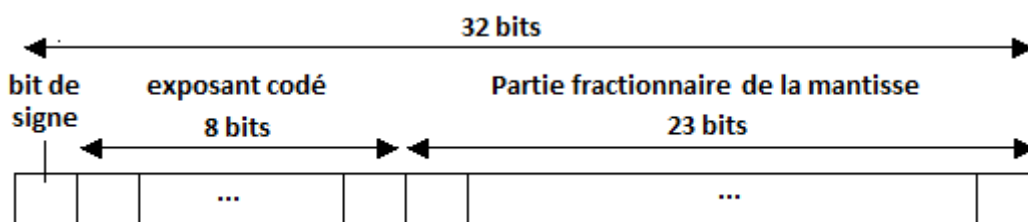
**Le simple précision (32 bits)**  
**Le double précision (64 bits)**  
**Le quadruple précision (128 bits)**

#### 4.5.3.4. Nombre réel simple précision

**Exemple :** Le nombre réel simple précision 10,25, écrit en norme IEEE754 sous la forme  $1,28125 \times 2^3$ , sera représenté de la façon suivante :



En mémoire, on stocke le bit de signe, l'exposant sur 8 bits et la partie fractionnaire de la mantisse sur 23 bits soit 32 bits au total.



**Exemple :**

A quelle valeur réelle correspond le nombre stocké en machine par :

0 0111 1101 110 1100 0000 0000 0000 0000

□ Solution :

Le bit de gauche est 0 ce qui signifie que le nombre est positif

L'exposant codé est 0111 1101 = 125

E réel = E codé - Biais = 125 - 127 = -2

La mantisse avec le bit caché est 1.110 1100 0000 0000 0000 0000

$= 1.2^0 + 1.2^{-1} + 1.2^{-2} + 0.2^{-3} + 1.2^{-4} + 1.2^{-5} + \dots + 0.2^{-23} = 1 + 1/2 + 1/4 + 1/16 + 1/32 = 1,84375$

Le nombre réel recherché =  $+1,84375.2^{-2}$

#### Exercice

Testez les programmes suivants mettant en jeu des variables de type flottant simple précision.



Interprétez le résultat de l'exécution. Remarquez la présence du f derrière la valeur numérique. Elle est indispensable pour x1, x2, x3 car une valeur comportant un point décimal est par défaut un double précision en java et non un simple.

```
public class Test {

 public static void main(String[] args) {
 float x1,x2,x3;
 x1=3331720f;
 x2=0.625f;
 x3=x1+x2;
 System.out.println(x3);

 x1=123456789f;
 x3=5+x1;
 x3=x3-x1;
 System.out.println(x3);

 x1=853000000f;
 x2= 7400000f;
 x3=x1+x2;
 System.out.println(x3);

 x1=0.000000000000853f;
 x2=0.000000000000074f;
 x3=x1+x2;

 System.out.println(x3);
 x1=0.000000000000853f;
 x2=0.000000000000074f;
 x3=10.0f+x1+x2;
 System.out.println(x3);
 }
}
```

**Le format simple précision garde (au moins) 6 chiffres significatifs.**

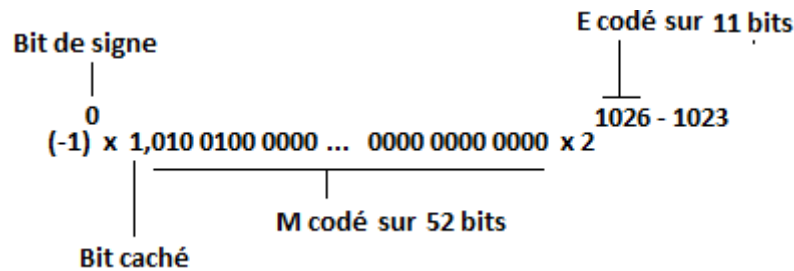
**Remarque :**

Il ne faut pas confondre chiffres derrière la virgule et chiffres significatifs !

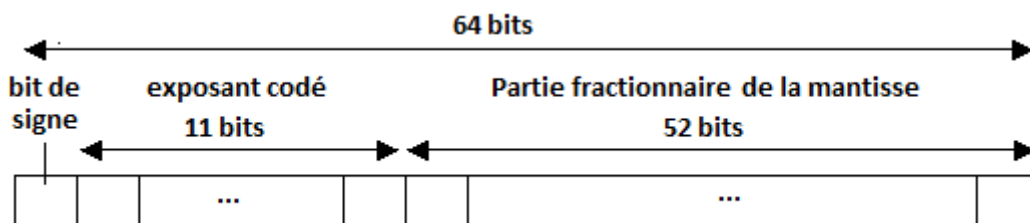
0,000000256 et 354 sont deux nombres comprenant 3 chiffres significatifs.

#### 4.5.3.5. Nombre réel double précision

Exemple : le nombre réel double précision 10,25, écrit en norme IEEE754  $1,28125 \times 2^3$ , sera stocké en mémoire de la façon suivante :



En mémoire, on stocke le bit de signe, l'exposant sur 11 bits et la partie fractionnaire de la mantisse sur 52 bits soit 64 bits au total.



Reprenons les programmes testés lors de l'étude du réel simple précision et comparons le résultat de l'exécution

```
double x1, x2, x3;
x1=3331720;
x2=0.625;
x3=x1+x2;
System.out.println(x3);
```

```
x1=123456789;
x3=5+x1;
x3=x3-x1;
System.out.println(x3);
```

```
//ne pas confondre chiffres significatifs et chiffres derrière la virgule
x1=853000000;
x2= 7400000;
x3=x1+x2;
System.out.println(x3);
```

```
x1=0.0000000000000853;
x2=0.0000000000000074;
x3=x1+x2;
System.out.println(x3);
x1=0.0000000000000853;
x2=0.0000000000000074;
x3=10.0+x1+x2;
System.out.println(x3);
```

**Le format double précision garde au moins 15 chiffres significatifs.**

#### 4.5.3.6. Format d’affichage des nombres réels

Il peut être nécessaire de présenter les nombres à virgules flottantes sous un format habituel aux yeux de l’utilisateur et non au format scientifique de type 5.2948 E13 ou avec un nombre de décimales trop important comme 0.17142858. En java, l’objet `DecimalFormat` permet de présenter les nombres à virgules flottantes au format désiré, spécifiant le nombre de décimales ainsi que le type d’arrondi à utiliser.

Testez et interprétez le code ci-dessous.

Quel serait le format à choisir pour les nombres réels inscrits sur le ticket de caisse sachant qu’on veut toujours deux décimales quel que soit le prix (exemple : 36 doit être noté 36.00).

```
import java.math.RoundingMode;
import java.text.DecimalFormat;

public class TestDecimalFormat {
 public static void main(String[] args) {

 float x1,x2,x3,x4,x5;
 DecimalFormat DfFormat1 = new DecimalFormat("0.0000");
 DecimalFormat DfFormat2 = new DecimalFormat("#.####");
 DecimalFormat DfFormat3 = new DecimalFormat("0.00");

 x1=52945000000000.0f;
 x2=30000000000.0f;
 x3=x1+x2;
 x4=15.0f/4.0f;
 x5=1.2f/7.0f;

 System.out.println("x3 sans format : " + x3);
 System.out.println("x3 Format 1 0.0000 : " + DfFormat1.format(x3));
 System.out.println("x3 Format 2 #.#### : " + DfFormat2.format(x3));

 System.out.println("x4 sans format : " + x4);
 System.out.println("x4 Format 1 0.0000 : " + DfFormat1.format(x4));
 System.out.println("x4 Format 2 #.#### : " + DfFormat2.format(x4));

 System.out.println("x5 sans format : " + x5);
 System.out.println("x5 Format 1 0.0000 : " + DfFormat1.format(x5));
 System.out.println("x5 Format 2 #.#### : " + DfFormat2.format(x5));
 System.out.println("x5 Format 3 0.00 : " + DfFormat3.format(x5));
 //Pour identifier les arrondis effectués par DECimalFormat
 DfFormat3.setRoundingMode(RoundingMode.HALF_UP);
 System.out.println("37.624 Format 3 0.00 half up : " +
DfFormat3.format(37.624));
 System.out.println("37.625 Format 3 0.00 half up : " +
DfFormat3.format(37.625));
 System.out.println("37.626 Format 3 0.00 half up : " +
DfFormat3.format(37.626));
 System.out.println("37.6 Format 3 0.00 half up : " +
DfFormat3.format(37.6));

 DfFormat3.setRoundingMode(RoundingMode.HALF_DOWN);
 System.out.println("37.624 Format 3 0.00 half down : " +
DfFormat3.format(37.624));
 System.out.println("37.625 Format 3 0.00 half down : " +
DfFormat3.format(37.625));
 System.out.println("37.626 Format 3 0.00 half down : " +
DfFormat3.format(37.626));
 }
}
```

#### 4.5.4. Les conversions numériques

Il arrive fréquemment que l'on doive manipuler des variables de types différents. L'exemple ci-dessous illustre l'un des dangers à manipuler des entiers et des réels dans un même calcul.

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

 int intPctTVA = 21;
 double dblPrixNet, dblPrixBrut;

 dblPrixBrut= 100;
 dblPrixNet = dblPrixBrut * (1+ (intPctTVA/100));

 System.out.println("Pour un prix brut de " + dblPrixBrut + " € le prix
net est de " + dblPrixNet + " €");

}
```

donne le résultat erroné : Pour un prix brut de 100.0€ le prix net est de 100.0€  
Les conversions numériques sont alors nécessaires pour arriver à un résultat correct.

##### 4.5.4.1. Les conversions implicites

L'expression d'affectation est parfaitement définie, si la variable à gauche du signe d'affectation et la valeur à droite du signe d'affectation sont de **même type**. Dans les autres cas, une **conversion implicite** peut intervenir, s'il n'y a pas de dégradation de la valeur lors de la copie.

Les conversions permises doivent suivre une des 2 hiérarchies suivantes. Elles sont tout à fait logiques si on se rappelle qu'on peut mettre une petite boîte dans un grande mais non l'inverse.

**byte → short → int → long → float → double**  
**char → int → long → float → double**

Interprétez le comportement du compilateur à ces instructions :

```
short shMaVariableShort=0;
int intMaVariableInt=0;
double dblMaVariableDouble=0;

shMaVariableShort = intMaVariableInt;
intMaVariableInt = shMaVariableShort;
intMaVariableInt = dblMaVariableDouble;
dblMaVariableDouble = intMaVariableInt;
```

#### 4.5.4.2. Conversions explicites

Le programmeur peut forcer **explicitement** les conversions au moyen de l'**opérateur de trans-typage**. Si la conversion ne respecte pas les hiérarchies définies précédemment, cette conversion provoquera une perte de précision. En effet, il est possible de placer une boîte dans une plus petite mais à condition d'en arracher un morceau.

L'exemple suivant convertit le contenu d'un entier dans une variable de type `short`

```
shMaVariableShort = (short)intMaVariableInt;
```

De façon, générale, en java on peut écrire :

VariableDunType1 = (NomRéservéDuType1) VariableDunAutreType;

Cette conversion dégradante peut provoquer une altération de la valeur (perte des chiffres de la partie fractionnaire, perte de chiffres en partie entière) suivant la valeur numérique contenue dans la variable du membre de droite.

Interprétez le résultat du compilateur et de l'exécution à ces instructions :

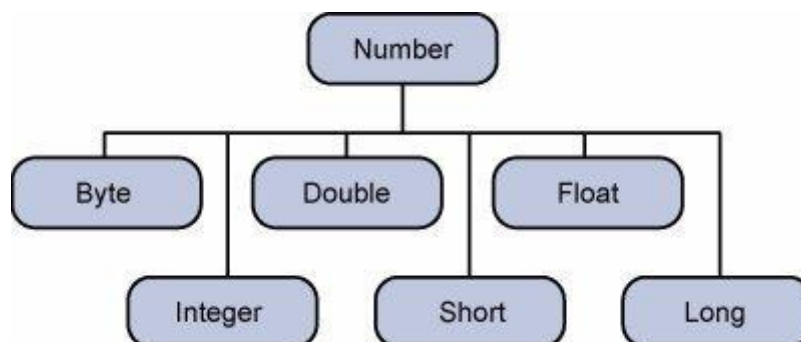
```
short shMaVariableShort;
int intMaVariableInt = 156000;
double dblMaVariableDouble=3.5;

shMaVariableShort = (short)intMaVariableInt;
intMaVariableInt = (int)dblMaVariableDouble;

System.out.println(shMaVariableShort);
System.out.println(intMaVariableInt);
```

#### 4.5.5. Les objets nombres en Java

En plus des types primitifs issus du langage C, Java permet d'utiliser les nombres sous forme d'objets. L'avantage réside dans le fait qu'il existe une série de fonctionnalités (méthodes) directement intégrées, notamment les conversions de type. On a d'ailleurs utilisé une de ces méthodes au §4.5.3.2 pour afficher des nombres entiers au format hexadécimal.



Pour passer du type primitif à la classe nombre, il suffit de remplacer `int` par `Integer`, `double` par `Double`, ...

On ne s'étendra pas davantage sur les objets nombres puisque, comme leur nom l'indique, ils relèvent de la programmation objet. On préférera donc pour l'instant utiliser les types primitifs.

Testez l'équivalence des variables suivantes :

```
int intMonEntier = 157;
Integer intMonAutreEntier = 157;

System.out.println("intMonEntier contient " + intMonEntier);
System.out.println("intMonAutreEntier contient " + intMonAutreEntier);
```

### 4.5.6. Le codage hexadécimal

Le codage hexadécimal ou codage de base 16 utilise 16 codes : 0,1,2,4,5,6,7,8,9,A,B,C,D,E,F correspondant à 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

#### 4.5.6.1. Conversion hexadécimal - décimal

□ Donné à titre indicatif, pas vu au cours

Le nombre décimal correspondant au nombre hexa  $a_n \dots a_1 a_0$  est donné par la formule

$$a_n \dots a_3 a_2 a_1 a_0 = a_n.16^n + \dots + a_3.16^3 + a_2.16^2 + a_1.16^1 + a_0.16^0$$

ex :  $(2A0F)_{16}$  à convertir en décimal

Après développement, on trouve  $(2A0F)_{16} = 10767$

$$a_0 = F, a_1 = 0, a_2 = A, a_3 = 2 \Rightarrow (2A0F)_{16} = 2.16^3 + A.16^2 + 0.16^1 + 4.16^0 = 10767$$

**Exercices :** convertir en décimal

- 1)  $(0111)_{16} =$
- 2)  $(ABC)_{16} =$
- 3)  $(F5C3)_{16} =$
- 4)  $(ED9D)_{16} =$

Vérifiez vos réponses en testant le code ci-dessous. Complétez le code pour afficher les autres variables. Remarquez les ressemblances, les différences de code et d’affichage pour l’exercice 1) et interprétez les.

```
public class ExTypeDonnees1 {

 public static void main(String[] args) {

 int intHexaEx1 = 0x0111;
 int intHexaBinEx1 = 0b0111;
 int intDecEx1 = 273;
 int intBinEx1 = 0b100010001;

 int intEx2 = 0xABC;
 int intEx3 = 0xF5C3;
 int intEx4 = 0xED9D;

 System.out.println ("intHexaEx1 " + intHexaEx1);
 System.out.println ("intHexaBinEx1 " + intHexaBinEx1);
 System.out.println ("intDecEx1 " + intDecEx1);
 System.out.println ("intBinEx1 " + intBinEx1);
 System.out.println ("intEx2 " + intEx2);
 System.out.println ("intEx3 " + intEx3);
 System.out.println ("intEx4 " + intEx4);

 } //fin main

} //fin class
```

**4.5.6.2. Conversion décimal -> hexadécimal**

- On utilise la méthode de la division entière par 16

Exemple : soit  $(1258)_{10}$  à convertir en hexadécimal.

En appliquant la même méthode qu'en binaire mais en divisant par 16 cette fois, on obtient

$(1258)_{10} = (4EA)_{16}$  en hexadécimal

|      | Reste de la division entière par 16              |                                        |
|------|--------------------------------------------------|----------------------------------------|
| 1258 |                                                  |                                        |
| 78   | A car $16 \cdot 78 = 1248$ et $1258 - 1248 = 10$ | $\Rightarrow (1258)_{10} = (4EA)_{16}$ |
| 4    | E car $16 \cdot 4 = 64$ et $78 - 64 = 14$        |                                        |
| 0    | 4                                                |                                        |
|      |                                                  |                                        |

**Exercices** : convertir en hexadécimal.

- 1)  $(8)_{10} = (8)_{16}$
- 2)  $(10)_{10} = (A)_{16}$
- 3)  $(32)_{10} = (20)_{16}$
- 4)  $(755)_{10} = (2F3)_{16}$

Vérifiez vos réponses en utilisant le code suivant :

```
Integer intEntier = 0;
System.out.println(intEntier.toHexString(8));
System.out.println(intEntier.toHexString(10));
System.out.println(intEntier.toHexString(32));
System.out.println(intEntier.toHexString(755));
```

8  
a  
20  
2f3

Remarque : le mot `Integer` désigne non pas un type dit primitif d'entier comme celui qu'on a rencontré jusque maintenant (`int`) mais un objet de type entier. `toHexString` est une **méthode** c'est-à-dire une fonctionnalité de l'objet `intEntier`. Il s'agit de notions de programmation objet.



### 4.5.7. Le codage ASCII

Les informations informatiques ne comportent pas que des nombres, nous utilisons également des caractères comme les lettres de l'alphabet minuscules et majuscules, les symboles de ponctuation, les chiffres (0 à 9), les symboles mathématiques,...

Le code ASCII (American Standard Code for Information Interchange) est un système de codage universel et permet de coder chaque caractère par un nombre.

Le code ASCII de base représente les caractères sur 7 bits (c'est-à-dire 128 caractères possibles, de 0 à 127).

#### ASCII CODE TABLE

| DEC | HEX  | OCT | CHAR                               | DEC | HEX  | OCT | CHAR              | DEC | HEX  | OCT | CHAR     | DEC | HEX  | OCT | CHAR       |
|-----|------|-----|------------------------------------|-----|------|-----|-------------------|-----|------|-----|----------|-----|------|-----|------------|
| 0   | 0x00 | 000 | <b>NUL</b> (null)                  | 32  | 0x20 | 040 | <b>SP</b> (space) | 64  | 0x40 | 100 | <b>@</b> | 96  | 0x60 | 140 | <b>`</b>   |
| 1   | 0x01 | 001 | <b>SOH</b> (start of heading)      | 33  | 0x21 | 041 | <b>!</b>          | 65  | 0x41 | 101 | <b>A</b> | 97  | 0x61 | 141 | <b>a</b>   |
| 2   | 0x02 | 002 | <b>STX</b> (start of text)         | 34  | 0x22 | 042 | <b>"</b>          | 66  | 0x42 | 102 | <b>B</b> | 98  | 0x62 | 142 | <b>b</b>   |
| 3   | 0x03 | 003 | <b>ETX</b> (end of text)           | 35  | 0x23 | 043 | <b>#</b>          | 67  | 0x43 | 103 | <b>C</b> | 99  | 0x63 | 143 | <b>c</b>   |
| 4   | 0x04 | 004 | <b>EOT</b> (end of transmission)   | 36  | 0x24 | 044 | <b>\$</b>         | 68  | 0x44 | 104 | <b>D</b> | 100 | 0x64 | 144 | <b>d</b>   |
| 5   | 0x05 | 005 | <b>ENQ</b> (enquiry)               | 37  | 0x25 | 045 | <b>%</b>          | 69  | 0x45 | 105 | <b>E</b> | 101 | 0x65 | 145 | <b>e</b>   |
| 6   | 0x06 | 006 | <b>ACK</b> (acknowledge)           | 38  | 0x26 | 046 | <b>&amp;</b>      | 70  | 0x46 | 106 | <b>F</b> | 102 | 0x66 | 146 | <b>f</b>   |
| 7   | 0x07 | 007 | <b>BEL</b> (bell)                  | 39  | 0x27 | 047 | <b>'</b>          | 71  | 0x47 | 107 | <b>G</b> | 103 | 0x67 | 147 | <b>g</b>   |
| 8   | 0x08 | 010 | <b>BS</b> (backspace)              | 40  | 0x28 | 050 | <b>(</b>          | 72  | 0x48 | 110 | <b>H</b> | 104 | 0x68 | 150 | <b>h</b>   |
| 9   | 0x09 | 011 | <b>HT</b> (horizontal tab)         | 41  | 0x29 | 051 | <b>)</b>          | 73  | 0x49 | 111 | <b>I</b> | 105 | 0x69 | 151 | <b>i</b>   |
| 10  | 0x0A | 012 | <b>LF</b> (NL line feed, new line) | 42  | 0x2A | 052 | <b>*</b>          | 74  | 0x4A | 112 | <b>J</b> | 106 | 0x6A | 152 | <b>j</b>   |
| 11  | 0x0B | 013 | <b>VT</b> (vertical tab)           | 43  | 0x2B | 053 | <b>+</b>          | 75  | 0x4B | 113 | <b>K</b> | 107 | 0x6B | 153 | <b>k</b>   |
| 12  | 0x0C | 014 | <b>FF</b> (NP from feed, new page) | 44  | 0x2C | 054 | <b>,</b>          | 76  | 0x4C | 114 | <b>L</b> | 108 | 0x6C | 154 | <b>l</b>   |
| 13  | 0x0D | 015 | <b>CR</b> (carriage return)        | 45  | 0x2D | 055 | <b>-</b>          | 77  | 0x4D | 115 | <b>M</b> | 109 | 0x6D | 155 | <b>m</b>   |
| 14  | 0x0E | 016 | <b>SO</b> (shift out)              | 46  | 0x2E | 056 | <b>.</b>          | 78  | 0x4E | 116 | <b>N</b> | 110 | 0x6E | 156 | <b>n</b>   |
| 15  | 0x0F | 017 | <b>SI</b> (shift in)               | 47  | 0x2F | 057 | <b>/</b>          | 79  | 0x4F | 117 | <b>O</b> | 111 | 0x6F | 157 | <b>o</b>   |
| 16  | 0x10 | 020 | <b>DLE</b> (data link escape)      | 48  | 0x30 | 060 | <b>0</b>          | 80  | 0x50 | 120 | <b>P</b> | 112 | 0x70 | 160 | <b>p</b>   |
| 17  | 0x11 | 021 | <b>DC1</b> (device control 1)      | 49  | 0x31 | 061 | <b>1</b>          | 81  | 0x51 | 121 | <b>Q</b> | 113 | 0x71 | 161 | <b>q</b>   |
| 18  | 0x12 | 022 | <b>DC2</b> (device control 2)      | 50  | 0x32 | 062 | <b>2</b>          | 82  | 0x52 | 122 | <b>R</b> | 114 | 0x72 | 162 | <b>r</b>   |
| 19  | 0x13 | 023 | <b>DC3</b> (device control 3)      | 51  | 0x33 | 063 | <b>3</b>          | 83  | 0x53 | 123 | <b>S</b> | 115 | 0x73 | 163 | <b>s</b>   |
| 20  | 0x14 | 024 | <b>DC4</b> (device control 4)      | 52  | 0x34 | 064 | <b>4</b>          | 84  | 0x54 | 124 | <b>T</b> | 116 | 0x74 | 164 | <b>t</b>   |
| 21  | 0x15 | 025 | <b>NAK</b> (negative acknowledge)  | 53  | 0x35 | 065 | <b>5</b>          | 85  | 0x55 | 125 | <b>U</b> | 117 | 0x75 | 165 | <b>u</b>   |
| 22  | 0x16 | 026 | <b>SYN</b> (synchronous idle)      | 54  | 0x36 | 066 | <b>6</b>          | 86  | 0x56 | 126 | <b>V</b> | 118 | 0x76 | 166 | <b>v</b>   |
| 23  | 0x17 | 027 | <b>ETB</b> (end of trans. block)   | 55  | 0x37 | 067 | <b>7</b>          | 87  | 0x57 | 127 | <b>W</b> | 119 | 0x77 | 167 | <b>w</b>   |
| 24  | 0x18 | 030 | <b>CAN</b> (cancel)                | 56  | 0x38 | 070 | <b>8</b>          | 88  | 0x58 | 130 | <b>X</b> | 120 | 0x78 | 170 | <b>x</b>   |
| 25  | 0x19 | 031 | <b>EM</b> (end of medium)          | 57  | 0x39 | 071 | <b>9</b>          | 89  | 0x59 | 131 | <b>Y</b> | 121 | 0x79 | 171 | <b>y</b>   |
| 26  | 0x1A | 032 | <b>SUB</b> (substitute)            | 58  | 0x3A | 072 | <b>:</b>          | 90  | 0x5A | 132 | <b>Z</b> | 122 | 0x7A | 172 | <b>z</b>   |
| 27  | 0x1B | 033 | <b>ESC</b> (escape)                | 59  | 0x3B | 073 | <b>;</b>          | 91  | 0x5B | 133 | <b>[</b> | 123 | 0x7B | 173 | <b>{</b>   |
| 28  | 0x1C | 034 | <b>FS</b> (file separator)         | 60  | 0x3C | 074 | <b>&lt;</b>       | 92  | 0x5C | 134 | <b>\</b> | 124 | 0x7C | 174 | <b> </b>   |
| 29  | 0x1D | 035 | <b>GS</b> (group separator)        | 61  | 0x3D | 075 | <b>=</b>          | 93  | 0x5D | 135 | <b>]</b> | 125 | 0x7D | 175 | <b>}</b>   |
| 30  | 0x1E | 036 | <b>RS</b> (record separator)       | 62  | 0x3E | 076 | <b>&gt;</b>       | 94  | 0x5E | 136 | <b>^</b> | 126 | 0x7E | 176 | <b>~</b>   |
| 31  | 0x1F | 037 | <b>US</b> (unit separator)         | 63  | 0x3F | 077 | <b>?</b>          | 95  | 0x5F | 137 | <b>_</b> | 127 | 0x7F | 177 | <b>DEL</b> |

*codage ASCII*

### 4.5.8. L'Unicode

En observant la table du codage ASCII, on remarque son avantage : il contient les caractères que nous utilisons le plus souvent et il n'utilise que 7 bits (1 octet) pour les représenter en mémoire. L'inconvénient est qu'il ne reprend pas bon nombre de caractères spécifiques à une langue comme les lettres accentuées de la langue française, les lettres grecques, les symboles monétaires, les symboles mathématiques,... Il n'est donc pas suffisant pour coder l'intégralité des caractères présents dans les textes que nous utilisons.

L'Unicode permet d'inventorier les caractères utiles et utilisés dans toutes les langues et tous les domaines (monétaires, mathématique...). Il est mis à jour régulièrement et présente donc des versions. L'Unicode associe un caractère à un nombre hexadécimal de 0000 à FFFF (pour les premières tables) appelé point de code (code point). Ce n'est pas cette valeur qui est stockée en machine. La façon de stocker (coder) un caractère unicode en machine dépend du type d'encodage (UTF-8, UTF16). Java utilise le codage UTF16, stocke donc les caractères sur 2 octets et peut représenter les 65536 premiers caractères Unicode.

|   |     |      |     |     |     |     |     |     |
|---|-----|------|-----|-----|-----|-----|-----|-----|
|   | 000 | 001  | 002 | 003 | 004 | 005 | 006 | 007 |
| 0 | NUL | SOH  | SP  | 0   | @   | P   |     | p   |
| 1 | STX | DC1  | !   | 1   | A   | Q   | a   | q   |
| 2 | ETX | DC2  | "   | 2   | B   | R   | b   | r   |
| 3 | END | DC3  | #   | 3   | C   | S   | c   | s   |
| 4 | SOB | DC4  | \$  | 4   | D   | T   | d   | t   |
| 5 | ENQ | NAK  | %   | 5   | E   | U   | e   | u   |
| 6 | ACK | SYN  | &   | 6   | F   | V   | f   | v   |
| 7 | BEL | CTRL | '   | 7   | G   | W   | g   | w   |
| 8 | BS  | CAN  | (   | 8   | H   | X   | h   | x   |
| 9 | HT  | EM   | )   | 9   | I   | Y   | i   | y   |
| A | LF  | BUR  | *   | :   | J   | Z   | j   | z   |
| B | VT  | ESC  | +   | ;   | K   | [   | {   |     |
| C | FF  | FS   | ,   | <   | L   | \   |     |     |
| D | CR  | GS   | -   | =   | M   | ]   | m   | }   |
| E | SO  | RS   | >   | N   | ^   | n   | ~   |     |
| F | SI  | US   | /   | ?   | O   | _   | o   | DEL |

Caractères Unicode  
0000 à 007F (0 à 127)  
(caractères latins)

|   |      |      |     |     |     |     |     |     |
|---|------|------|-----|-----|-----|-----|-----|-----|
|   | 008  | 009  | 00A | 00B | 00C | 00D | 00E | 00F |
| 0 | CTRL | CTRL | SP  | °   | À   | Ä   | à   | ä   |
| 1 | CTRL | CTRL | !   | ±   | Á   | Å   | á   | å   |
| 2 | CTRL | CTRL | ¢   | 2   | Â   | Ö   | â   | ö   |
| 3 | CTRL | CTRL | £   | 3   | Ã   | Ó   | ã   | ó   |
| 4 | CTRL | CTRL | ¤   | 4   | Ä   | Ô   | ä   | ö   |
| 5 | CTRL | CTRL | ¥   | µ   | Å   | Ö   | å   | ö   |
| 6 | CTRL | CTRL | ¦   | ¶   | Æ   | Ø   | æ   | ø   |
| 7 | CTRL | CTRL | §   | ·   | Ç   | ×   | ç   | ÷   |
| 8 | CTRL | CTRL | ¨   | ¸   | È   | Ø   | è   | ø   |
| 9 | CTRL | CTRL | ©   | 1   | É   | Û   | é   | ù   |
| A | CTRL | CTRL | ª   | º   | Ê   | Ú   | ê   | ú   |
| B | CTRL | CTRL | «   | »   | Ë   | Û   | ë   | û   |
| C | CTRL | CTRL | ¼   | ¼   | Ì   | Ü   | ì   | ü   |
| D | CTRL | CTRL | ½   | ½   | Í   | Ý   | í   | ý   |
| E | CTRL | CTRL | ¾   | ¾   | Î   | Þ   | î   | þ   |
| F | CTRL | CTRL | ¿   | ¿   | Ï   | ß   | ï   | ÿ   |

Caractères Unicode  
0080 à 00FF (128 à 255)  
(caractères latins, dont  
accentués)

|   |     |     |     |     |     |     |     |     |
|---|-----|-----|-----|-----|-----|-----|-----|-----|
|   | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 |
| 0 | ㄱ   | ㅋ   | ㆁ   | ㆁ   | ㆁ   | ㆁ   | ㆁ   | ㆁ   |
| 1 | ㄴ   | ㄷ   | ㄹ   | ㄴ   | ㄴ   | ㄴ   | ㄴ   | ㄴ   |
| 2 | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   |
| 3 | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   |
| 4 | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   |
| 5 | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   |
| 6 | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   |
| 7 | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   |
| 8 | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   |
| 9 | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   |
| A | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   |
| B | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   |
| C | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   |
| D | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   |
| E | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   |
| F | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   | ㄷ   |

Caractères Unicode  
1100 à 117F (4352 à 4479)  
(caractères hangul jamo)

La notation \uxxxx (avec xxxx étant le code point Unicode au format hexadécimal) permet d'afficher le caractère correspondant en Java.

#### 4.5.8.1. Exercices

- 1) Quel est l'affichage produit par cette ligne de code ? Déterminez avant de l'exécuter.

```
System.out.print("\u0073\u0075\u0072\u0070\u0072\u0069\u0073\u0065");
```

- 2) Afficher 4.2.1 de la même manière que précédemment, via les code point Unicode.

### 4.5.9. Les caractères

Les caractères peuvent également être stockés dans des variables. En Java, le type de donnée caractère est défini par le mot réservé **char**. Il est possible d'utiliser le caractère par sa représentation ou sa valeur numérique (son point de code Unicode).

#### 4.5.9.1. Exercices

1) Testez et interprétez les quatre programmes suivants :

a)

```
char chCharTest='a';

System.out.println(chCharTest + "h d'accord!");
```

b)

```
char chCharTest=90;

System.out.println("Le caractère " +chCharTest + " a pour point de code " +
(int) chCharTest);
```

c)

```
char chCharTest=50;

System.out.println("Le caractère " +chCharTest + " a pour point de code " +
(int) chCharTest);
```

d)

```
char chPremierCharTest=50;
char chDeuxiemeCharTest=54;
int intTroisiemeCharTest;

intTroisiemeCharTest = (int)chPremierCharTest + (int)chDeuxiemeCharTest;
System.out.println(chPremierCharTest + "+" + chDeuxiemeCharTest + "="
+(char)intTroisiemeCharTest);
```

## 4.6. Algorithmes - Pseudo Code

Lors des exercices précédents, on a écrit du code source directement sans analyse préalable. Si on compare la programmation à la maçonnerie, ou à l'électricité, cela reviendrait à construire une maison sans plan d'architecte en posant les briques directement ou en câblant une armoire électrique au fur et à mesure que les fils nous tombent sous la main. Même si cette façon de procéder pourrait donner l'illusion qu'elle nous fait gagner du temps lors des premiers instants de la réalisation, en réalité, cela conduit à une construction brouillon, souvent erronée où l'on ne sait rapidement plus où on va ni d'où on vient. Cela aboutit à des temps d'arrêts pour se questionner, vérifier, revérifier, corriger et adapter. Pour réaliser un programme structuré, il est donc essentiel qu'une analyse soit réalisée au préalable. La programmation se réalise d'abord sur une feuille de papier à l'aide d'un crayon et d'une gomme et dans un formalisme indépendant du langage de programmation. Ce formalisme s'appelle **l'algorithme ou pseudo code**, il décrit le "comment on va faire" pour arriver à réaliser les fonctionnalités demandées par le programme.

Il n'existe pas de formalisme tout à fait standard pour l'écriture du pseudo code. Il suffit de consulter la littérature à ce sujet pour se rendre compte qu'il existe différentes formes d'écriture de pseudocode. Néanmoins celles-ci sont très semblables et facilement compréhensibles.

A partir du pseudocode, le programmeur, qui peut être différent de celui qui a réalisé l'analyse, va traduire le programme en code source dans un langage spécifique (Java, Python, C#, ...)

Reprenons le programme précédent et voyons comment on aurait dû, au préalable, écrire le pseudo code de celui-ci.

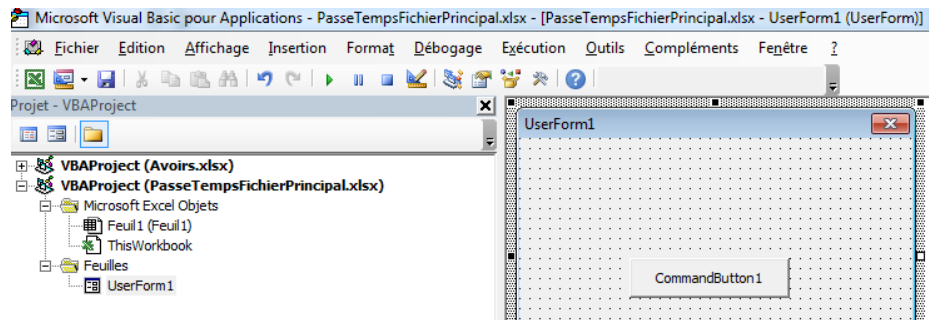
```
public static void main(String[] args) {

 int intQuantitConso; //quantité de consommations commandées
 double dblPrixUnitaire; //prix unitaire de la consommation
 double dblPrixTotParConso ;//prix total par type de consommation

 intQuantitConso = 3 ;
 dblPrixUnitaire = 2.5;
 dblPrixTotParConso = dblPrixUnitaire * intQuantitConso;

 System.out.print("Prix total pour cette consommation: " + dblPrixTotParConso +
 "€"); } //fin main
```

Pour bien comprendre l'utilité du pseudo code, réécrivons ce même programme en Visual Basic à partir d'Excel par exemple. Depuis un fichier excel acceptant les macros (.xlsx), cliquez sur le bouton droit sur une page, *visualiser le code*. Ensuite, depuis le menu, *insérer, UserForm*, placez un bouton sur le formulaire puis double cliquez sur celui-ci.



Introduisez le code suivant :

```
Private Sub CommandButton1_Click()
Dim intQuantitConso As Integer
Dim dblPrixUnitaire, dblPrixTotParConso As Double
 intQuantitConso = 3
 dblPrixUnitaire = 2.5
 dblPrixTotParConso = dblPrixUnitaire * intQuantitConso
 MsgBox "Prix total pour cette consommation " & dblPrixTotParConso & "€"
End Sub
```

Quelles sont les similitudes et différences avec le Java ?

On comprend maintenant aisément que ce qui fait la complexité de la programmation c'est bien plus l'analyse elle-même que sa traduction en code source dans un langage ou un autre.

**A partir de maintenant, il est strictement interdit de réaliser un code sans avoir réalisé un pseudocode au préalable.**

#### 4.6.1. Exercice

Réalisez l'exercice PJ1Ex1-4 mais cette fois dans l'ordre logique des choses, en réalisant d'abord un pseudocode puis le code Java.

PJ1Ex1-4 Le programme calcule le montant du prix net à partir d'un prix brut donné et de la TVA (fixée à 21%). Le programme affiche. « Pour un prix brut de xxx €, le prix net est de yyy € ». On donne, au prix brut, une valeur quelconque fixée dans le code.

## 4.7. Programmation structurée – les fonctions et procédures

Nous avons vu qu'il est possible d'obtenir un programme fonctionnel en se servant uniquement de `public static void main()` et en y entassant toutes nos lignes de code. Dans le même esprit, imaginez que, chez vous, vous aillez une seule immense armoire où vous stocker vos vêtements, votre vaisselle, votre outillage et vos classeurs de cours ! Même si c'est fonctionnel, ce n'est pas pour autant ordonné, pratique, efficace à l'utilisation ni joli à regarder quand on ouvre la porte de l'armoire. Il en va exactement de même pour notre code ; nous ne pouvons décemment pas nous servir de la méthode `main()` comme seul et unique fourre-tout pour l'ensemble de nos instructions.

Avant d'envisager la POO, il est possible de créer des fonctions et procédures qui permettent de rendre le code plus structuré, efficace, concis et lisible. Quand elles sont bien construites, ces procédures et fonctions, peuvent aussi être réutilisées indépendamment dans d'autres projets. Elles sont la base de la programmation dite structurée.

### 4.7.1. Procédures et fonctions

**Une fonction est un sous-programme qui renvoie une valeur d'un type défini au programme appelant**

**Une procédure est un sous-programme qui ne renvoie pas de valeur au programme appelant.**

La particularité du C++ et du Java, c'est qu'ils n'intègrent pas explicitement la différence entre les deux, ils considèrent simplement qu'une procédure est une fonction qui ne renvoie rien, d'où la présence du `void` (traduction : *vide* ou *rien* en français) dans ce cas. Dans d'autres langages, il existe un nom spécifique différent pour la fonction et la procédure.

Voici un exemple de programme Java comportant une procédure. Testez en mode debug pas à pas pour visualiser le séquençement des instructions exécutées. Remarquez le passage des deux paramètres `PrixAffiche` et `PctRemise` de la procédure `main()` à la procédure `MaProcedurePrixSolde()`.

```
public class TestProcedure {
 public static void main(String[] args) {

 MaProcedurePrixSolde(100.0,15.0); //appel procédure
 MaProcedurePrixSolde(200.0,40.0); //appel procédure
 MaProcedurePrixSolde(50.0,10.0); //appel procédure

 } //fin main
 static void MaProcedurePrixSolde(double PrixAffiche, double PctRemise) {

 double PrixAPayer=0;

 PrixAPayer = PrixAffiche - (PctRemise/100)*PrixAffiche;
 System.out.println("A Payer " + PrixAPayer + "€");
 } //fin MaProcedurePrixSolde
} //fin class
```



Montrons tout de suite la différence entre une *procédure* et une *fonction* en produisant le même résultat que le programme précédent mais en utilisant cette fois des *fonctions*.

On a dit qu'une fonction renvoyait une valeur au programme appelant. Voici donc une fonction `CalculerPrixSolde()` qui renvoie le prix à payer au programme appelant. Remarquez, de nouveau, le passage des deux paramètres `PrixAffiche` et `PctRemise` de la procédure `main()` à la fonction `CalculerPrixSolde()` et la valeur renvoyée (**return** `PrixAPayer`) par la fonction au programme appelant.

```
public class Test {
public static void main(String[] args) {

 System.out.println("A Payer " + CalculerPrixSolde(100.0,15.0) + "€");
} //fin main
static double CalculerPrixSolde(double PrixAffiche, double PctRemise) {
 double PrixAPayer;

 PrixAPayer= PrixAffiche - (PctRemise/100)*PrixAffiche;
 return PrixAPayer;

} //fin CalculerPrixSolde
} //fin class
```

Contrairement à la procédure qui ne renvoyait rien (**void**), la fonction renvoie une valeur de type **double** pour l'exemple ci-dessus. Le type de donnée **double** devant le nom de la fonction est donc obligatoire ainsi que la présence d'au moins un **return**. Remarquez que les types bien que tous **double** dans l'exemple ci-dessous peuvent être tous différents.

Pour d'autres langages la différence est plus évidente car il y a des mots réservés explicites pour la procédure et la fonction. Le Visual Basic utilise le mot **Sub** pour la procédure et **Function** pour la fonction



## 4.7.2. Pseudo code

### 4.7.2.1. Appel de procédure

```
ident_procedure (param1, param2, ...)
```

### 4.7.2.1. Appel de fonction

```
ident_var ← ident_fonction (param1, param2, ...)
```

### 4.7.2.2. Déclaration de procédure

```
PROCEDURE ident_procedure(ident_type_var nomParam1, ident_type_var
 paramètre 2 nomParam2, ...)
 [<declarations locales>]

DEBUT

 <instructions>

FIN PROCEDURE
```

### 4.7.2.3. Déclaration de fonction

```
FONCTION ident_fonction (ident_type_var nomParam1, ident_type_var
 nomParam2, ...) : ident_type_var_retourné
 [<declarations locales>]

DEBUT

 <instructions>
 ...
RETOURNE <résultat>

FIN FONCTION
```

#### Remarques importantes :

- il n'est pas obligatoire d'avoir des paramètres. Dans ce cas, la parenthèse qui suit le nom de la fonction ou la procédure est vide.
- Les noms des paramètres dans la fonction ou la procédure appelante ne doivent pas être forcément le même que ceux dans la déclaration de la procédure ou fonction.
- Dans une fonction, il peut y avoir plusieurs *RETOURNE* (**return**), ce peut être le cas par exemple dans une structure alternative.

### 4.7.3. Exercices

Réalisez les exercices PJ1Ex1-4-b et PJ1Ex1-4-c. Le but est identique à l'exercice PJ1Ex1-4 mais cette fois en utilisant les procédures et les fonctions.

PJ1Ex1-4-b Le programme calcule le montant du prix net à partir d'un prix brut donné et de la TVA qui, cette fois, n'est pas forcément 21%. En utilisant une procédure, le programme affiche : « Pour un prix brut de x €, le prix net est de y € ». Pour le test du programme, appelez votre procédure au moins deux fois, une première fois pour un prix brut de 100€ et une TVA à 6%, une deuxième fois pour un prix brut de 125.5€ et une TVA à 21%. En premier lieu, déterminez les paramètres de la procédure.

PJ1Ex1-4-c Le programme calcule le montant du prix net à partir d'un prix brut donné et de la TVA qui, cette fois, n'est pas forcément 21%. En utilisant une fonction qui calcule le prix net, le programme affiche : « Pour un prix brut de x €, le prix net est de y € ». Pour le test du programme, appelez votre fonction au moins deux fois, une première fois pour un prix brut de 100€ et une TVA à 6%, une deuxième fois pour un prix brut de 125.5€ et une TVA à 21%.

Réalisez les exercices PJ1Ex1-5-a, b et c.

PJ1Ex1-5-a : Le programme calcule le montant de la TVA à partir d'un prix net (prix net = prix TVAC avec un taux de TVA fixé à 21%). Le programme affiche. « Pour un net de x €, le coût de la TVA s'élève à y € ». Une analyse préalable au pseudocode est nécessaire ! Cet exercice ne prévoit pas d'utiliser les procédures et fonctions. A ce stade, si les procédures et fonctions ne vous posent aucun problème, réalisez le directement avec procédures et fonctions (énoncés suivants b et c). Si ces notions vous sont paraissent encore floues, il est alors intéressant d'écrire d'abord le programme uniquement dans le `main()`.

PJ1Ex1-5-b : Créez une procédure qui calcule et affiche le montant de la TVA à partir d'un prix net et un taux de TVA variables. Prendre 121€ et 21,0 %, 85,5€ et 12.5% pour les essais. Le programme affiche : « Pour un prix net de x € et une TVA de y %, le coût de la TVA s'élève à z € ».

PJ1Ex1-5-c : Créez une fonction qui calcule le montant de la TVA à partir d'un prix net et un taux de TVA variables. Prendre 121€ et 21,0 %, 85,5€ et 12.5% pour les essais. Le programme affiche : « Pour un prix net de x € et une TVA de y %, le coût de la TVA s'élève à z € ».

## 4.8. Saisies utilisateur

Lors des exercices précédents, les données de base pour les calculs étaient fixées dans le code source (TVA à 21%, Prix brut de 100€, ...) ce qui présentait l'inconvénient majeur de faire fonctionner le programme pour certaines valeurs uniquement et sans aucune interaction avec l'utilisateur du programme. On aimerait pouvoir effectuer les calculs en fonction de données choisies par l'utilisateur et sans devoir changer le code source. Pour cela, il faut recourir aux saisies de l'utilisateur.

### 4.8.1. Exemple

Un moyen préconisé par Java pour les saisies utilisateur est de recourir à la classe `Scanner`. Testez et interprétez le code suivant pour en comprendre l'utilisation. Il s'agit là de programmation orientée objet qu'on peut introduire ici même si elle fera partie d'une autre UE. Testez et interprétez le code suivant :

```
import java.util.Scanner;
public class TestSaisie {

 public static void main(String[] args) {

 int intSaisieUtilisateur;
 byte bytSaisieUtilisateur;
 double dblSaisieUtilisateur;

 Scanner sc = new Scanner(System.in);
 System.out.println("Entrez un nombre entier");
 intSaisieUtilisateur = sc.nextInt();
 System.out.println("Le nombre entier saisi est " +
 intSaisieUtilisateur);

 System.out.println("Entrez un byte");
 bytSaisieUtilisateur = sc.nextByte();
 System.out.println("Le byte saisi est " + bytSaisieUtilisateur);

 System.out.println("Entrez un nombre réel");
 dblSaisieUtilisateur = sc.nextDouble();
 System.out.print("Le réel saisi est " + dblSaisieUtilisateur);

 }
}
```

### 4.8.2. Pseudocode

En pseudocode, la saisie utilisateur se note :

*Lire `NomDeVariable`*

### 4.8.3. Exercices

Réalisez les exercices PJ1Ex1-6 et PJ1Ex1-7

PJ1Ex1-6-a L'exercice est identique au précédent c'est-à-dire qu'il permet d'afficher le coût de la TVA à partir d'un prix net mais, cette fois, l'utilisateur entre le prix net au clavier répondant à l'invitation « *Entrez le prix net* ». Tout le code est dans le `main()` pour cet énoncé.

PJ1Ex1-6-b Améliorez l'exercice précédent en utilisant une première fonction de calcul du coût de la TVA `VATCostCompute(..., ...)`, et une deuxième `getUserNetPriceInput( )` qui renvoie la saisie de l'utilisateur (le prix Net) au format réel double précision.

PJ1Ex1-7 L'exercice est identique à l'exercice PJ1Ex1-4 c'est-à-dire qu'il permet de calculer le prix net à partir de la TVA et du prix brut mais, cette fois, l'utilisateur entre le prix brut et le taux de TVA répondant aux invitations « *Entrez le prix brut* » et « *Entrez le taux de TVA* ». Tout le code est dans le `main()` pour cet énoncé.

PJ1Ex1-7-b Creez, testez et utilisez une fonction `computeNetPrice()`, une fonction `getUserGrossPriceInput( )` qui renvoie le prix brut saisi par l'utilisateur et une autre `getUserVATInput( )` qui renvoie le taux de TVA lui aussi saisi par l'utilisateur.

## 4.9. Chaînes de caractères.

Nous aurions pu parler plus tôt des chaînes caractères mais nous aurions manqué d'outils pour pouvoir les exploiter plus en profondeur. On entend par chaîne de caractères, une suite de caractères qui forment un texte ou simplement un mot. Nous avons déjà utilisé à plusieurs reprises le format "des mots, du texte !" ou "Le Passe-temps" qui sont en réalité des chaînes de caractères. En java, une chaîne de caractères doit être délimitée par des guillemets "".

### 4.9.1. La classe String

Java fournit, entre autres, une classe d'objet nommée `String` qui permet de manipuler des chaînes de caractères. Ainsi, le programme suivant affecte le nom de la brasserie dans la variable `strNomEtablissement` et l'affiche à la console :

```
String strNomEtablissement="Le Passe Temps";

System.out.println(strNomEtablissement);
```

Que ce soit pour afficher du texte variable à l'utilisateur, vérifier les saisies de l'utilisateur, mettre en forme du texte, l'utilisation des chaînes de caractères est omniprésente.

Il est nécessaire de connaître les fonctionnalités liées aux chaînes de caractères, de savoir qu'elles existent et de pouvoir y recourir au moment où l'on en a besoin.

Testez le code ci-dessous et déduisez-en les fonctionnalités des méthodes suivantes :

```
String strNom="Le Passe Temps";

System.out.println("1 lenght : " + strNom.length());

System.out.println("2 CompareTo 'Le Passe Temps' : " + strNom.compareTo("Le Passe Temps"));

System.out.println("3 CompareTo 'Le Passe Temps ' : " + strNom.compareTo("Le Passe Temps "));

System.out.println("4 CompareTo 'le passe temps': " + strNom.compareTo("le passe temps"));

System.out.println("5 CompareToIgnoreCase 'le passe temps': " + strNom.compareToIgnoreCase("le passe temps"));

strNom=" Le Passe Temps ";
System.out.println("6 :"+ strNom);

System.out.println("7:" + strNom.trim());

System.out.println("8:"+ strNom);

strNom=strNom.trim();
System.out.println("9 :"+ strNom);

strNom="Le Passe Temps";
System.out.println("10:"+ strNom.substring(10, 12));

System.out.println("11:"+ strNom.charAt(3));

System.out.println("12:"+ strNom.concat("\nBienvenu"));
```

```
System.out.println("13 Contenu de la variable : "+ strNom);

strNom = strNom + "\nBienvenu";
System.out.println("14 Contenu de la variable : " + strNom);

strNom="Le Passe Temps";
System.out.println("15 endsWith: " + strNom.endsWith("Temps"));

System.out.println("16 endsWith: " + strNom.endsWith("mps"));

System.out.println("17 endsWith: " + strNom.endsWith("mas"));

System.out.println("18 replace: " + strNom.replace("temps", "Partout"));

System.out.println("19 replace: " + strNom.replace("Temps", "Partout"));

System.out.println("20 replace All: " + strNom.replaceAll("e", "EEE"));

System.out.println("21 to lower case: " + strNom.toLowerCase());

System.out.println("22 to upper case: " + strNom.toUpperCase());

System.out.println("23 Index of: " + strNom.indexOf("s", 0));

System.out.println("24 Index of: " + strNom.indexOf("s", 2));
```

```
System.out.println("25 Index of: " + strNom.indexOf("s", 7));

System.out.println("26 Index of: " + strNom.indexOf("x", 0));

System.out.println("27 Last Index of e : " + strNom.lastIndexOf("e",10));

System.out.println("28 Last Index of e: " + strNom.lastIndexOf("e",9));

System.out.println("29 Last Index of e : " + strNom.lastIndexOf("e",0));

System.out.println("30 Last Index of e : " + strNom.lastIndexOf("e"));

strNom = Integer.toString(38);
System.out.println("31 string ou entier ?? " + strNom);
```

#### 4.9.2. Exercice

PJ1Ex1-7-c Améliorez l'exercice PJ1Ex1-7-b qui consistait à calculer le prix net en fonction d'un prix brut et d'une TVA saisis par l'utilisateur. Remplacez les deux fonctions de saisie ( `getUserGrossPriceInput( )` et `getUserVATInput( )` ) par une seule `getUser_doubleInput( )` avec cette fois, le texte d'invitation (« *Entrez le prix brut* », « *Entrez la TVA* ») passé en paramètre de la fonction.



## 4.10. Structures alternatives

### 4.10.1. Structure alternative if

#### 4.10.1.1. Premier exemple

Examinons l'énoncé de l'exercice PJ1Ex1-8-a.

L'utilisateur saisit un entier correspondant au N° de table répondant à l'invitation « Entrez le N° de table ». Le programme vérifie si le N° de table entré est correct. Si le N° est correct, le programme affiche « *Vous avez entré le numéro de table X* » (avec X le N° de table entré). Si le nombre est supérieur à 20 ou inférieur à 1, on affiche « *Numéro de table incorrect* ».

A l'analyse de l'énoncé, on peut dire qu'il y a deux alternatives possibles :

- Soit le N° de table est correct et on doit afficher le N° saisi, c'est celui qui sera utilisé dans l'impression du ticket.
- Soit le N° de table est erroné et on doit afficher "N° de table incorrect"

On parle alors de structure alternative à deux branches. Le pseudo code s'écrit de la manière suivante :

Classe PJ1Ex1-8-a

Algorithme Main() :

**DEBUT**

Variables locales :

Entier : intNumTable ;

Instructions :

Afficher "Entrez le N° de la table"

Lire intNumTable

SI intNumTable < 1 OU intNumTable > 20 ALORS

Afficher "N° de table incorrect"

SINON

Afficher "Vous avez entré le N° de table" + intNumTable

FIN SI

**FIN**

En java, SI intNumTable < 1 OU intNumTable > 20 ALORS s'écrit :

```
if (intNumTable < 1 || intNumTable > 20) {
 ...
 série d'instructions
 ...
}
```

SINON s'écrit

```
else {
 ...
 série d'instructions
 ...
}
```

#### 4.10.1.2. Exercice

En tenant compte de ces informations de syntaxe java, écrivez et testez le code pour l'exercice PJ1Ex1-8.

**4.10.1.3. Opérateurs booléens et de comparaison**

Dans l'expression logique des conditions, on retrouvera, en Java, les opérateurs de comparaison et opérateurs booléens repris dans les tableaux suivants :

| Opérateur de comparaison | Syntaxe en Java |
|--------------------------|-----------------|
| Strictement plus grand   | >               |
| Strictement plus petit   | <               |
| Plus grand ou égal       | >=              |
| Plus petit ou égal       | <=              |
| Différent                | !=              |
| Egal                     | = =             |

Remarque : l'opérateur de comparaison = = est souvent confondu avec le signe d'affectation = en Java. Même si en français, on dit "égal" indifféremment pour les deux signes, le premier est bien un opérateur de comparaison donnant un résultat booléen alors que l'autre modifie une valeur stockée en mémoire. Chez les débutants, on retrouve fréquemment le code erroné suivant :

```
if (intValue = 1)
```

| Opérateur booléen | Syntaxe en Java |
|-------------------|-----------------|
| ET                | &&              |
| OU                |                 |
| OU Exclusif       | ^               |
| NOT               | !               |

#### 4.10.1.4. Pseudocode et code Java des structures alternatives

Le tableau ci-dessous présente les différentes structures alternatives qu'on peut rencontrer.

| Condition simple                                         | Alternative à 2 branches                                                          | Alternative à N+1 branche                                                                                                                                                                                                                                                                        |
|----------------------------------------------------------|-----------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SI expression booléenne<br>ALORS<br>Traitement<br>FIN SI | SI expression booléenne 1<br>ALORS<br>Traitement<br>SINON<br>Traitement<br>FIN SI | SI expression booléenne 1<br>ALORS<br>Traitement<br>SINON SI expression booléenne 2<br>Traitement<br>SINON SI expression booléenne 3<br>Traitement<br>SINON SI expression booléenne 4<br>Traitement<br>...<br>SINON SI expression booléenne N<br>Traitement<br><br>SINON<br>Traitement<br>FIN SI |
| Java                                                     |                                                                                   |                                                                                                                                                                                                                                                                                                  |
| <pre>if(Boolean_expression) {     // Statements }</pre>  | <pre>if(Boolean_expression) {     // Statements }else {     // Statements }</pre> | <pre>if(Boolean_expres 1) {     // Statements }else if(Boolean_expres 2) {     // Statements }else if(Boolean_expres 3) {     // Statements }else {     // Statements }</pre>                                                                                                                    |

#### 4.10.1.5. Exercices

Réalisez les exercices PJ1Ex1-8-b, PJ1Ex1-9 et PJ1Ex1-10

PJ1Ex1-8-b. Créez une fonction `getTableNumber()` qui permet de renvoyer le N° de table saisi par l'utilisateur répondant à l'invitation « *Entrez le numéro de table* ». La fonction vérifie si le N° de table entré est correct. Le N° doit être compris entre 1 et le N° de la dernière table qui est le *NumMax* passé en paramètre. Normalement, ce nombre est de 20 mais le programme doit avoir prévu une extension éventuelle de l'établissement ; c'est pourquoi le nombre 20 doit être une constante du programme. Si le nombre est supérieur au N° de table maximum ou inférieur à 1, la fonction affiche « *Numéro de table incorrect, le N° doit être compris entre 1 et N° max* ». Si le N° de table est incorrect, la fonction doit renvoyer -1 comme code d'erreur. Prévoyez plusieurs appels de la fonction de façon à la valider en testant tous les cas de figure.

PJ1Ex1-9 Créez une fonction `getAndCheckTableNumber()` qui permet de renvoyer le N° de table saisi par l'utilisateur répondant à l'invitation « *Entrez le numéro de table* ». La fonction vérifie si le N° de table entré est correct. La différence avec l'exercice précédent réside dans le fait que le message d'erreur doit être spécifique à l'erreur de saisie : si le N° est inférieur à 1, le programme affiche « *Saisie incorrecte ! Le N° de table doit être supérieur ou égal à 1* », si le

nombre est supérieur à 20 (constante du programme), on affiche « *Saisie incorrecte ! Le N° table le plus élevé est NumMax. Contactez l'administrateur du programme si vous souhaitez augmenter ce nombre* », si le N° est correct, le programme main() affiche « *Vous avez entré le numéro de table X* » avec X le N° de table entré. Si le N° de table est incorrect, la fonction doit renvoyer -1 comme code d'erreur.

Remarque : après avoir terminé votre exercice, enlevez les mots **else** de votre programme, laissez seulement le **if** et testez l'exécution du programme. Comparez le résultat de l'exécution puis le déroulement de celle-ci en exécutant le programme pas à pas à l'aide du mode *Debug*. Donnez votre avis sur l'optimisation du programme pour l'une et l'autre solution.

PJ1Ex1-10 Le programme calcule le prix net à partir de la TVA et du prix brut saisis par l'utilisateur. Le taux de TVA est choisi par l'utilisateur répondant à l'invitation "*Pour un taux de TVA de 6% tapez 1, 2 pour 12% et 3 pour 21%*". On affiche alors le texte : "*Pour un prix brut de X€, le prix net est de Y€ (TVA Z%)*". Si l'entrée de l'utilisateur n'est pas dans les choix possibles, le programme s'arrête en indiquant « *Saisie du taux incorrecte* » et le prix brut n'est pas demandé. Utilisez les fonctions déjà développées pour récupérer le prix brut et calculer le prix net. Ajoutez une fonction `getVATChoice()` qui renvoie le taux de TVA choisi en %, -1.0 en cas de choix incorrect. Prévoyez tous les tests qui permettent de valider les fonctionnalités.

## 4.10.2. Structure alternative - Switch Case

### 4.10.2.1. Exemple

Il existe en java comme dans d'autres langages, une autre structure alternative qui gère les conditions sous forme de différents cas possibles en fonction de la valeur d'une variable. Voyons comment traiter l'exemple précédent PJ1Ex1-10 en remplaçant les "if else" habituels par le "switch case". Essayez le programme suivant et déduisez-en le fonctionnement du **switch** ...**case**, du **default**, du **break** et le rôle de la variable `bFlagSaisieOK`.

```
import java.util.Scanner;

public class Ex10_AvecSwitchCase {
public static void main(String[] args) {

final double dblPCT_TVA1 = 6.0, dblPCT_TVA2 = 12.0, dblPCT_TVA3=21.0;
double dblPctTVA=0.0, dblPrixNet, dblPrixBrut;
int intChoixTVA;
boolean bFlagSaisieOK = false;

Scanner SaisieUtilisateur = new Scanner(System.in);
System.out.println("Pour un taux de TVA de 6% tapez 1, 2 pour 12% et 3 pour 21%\n.");
intChoixTVA = SaisieUtilisateur.nextInt();

switch (intChoixTVA)
{
case 1 :
 dblPctTVA =dblPCT_TVA1 ;
 bFlagSaisieOK = true;
 break;
case 2 :
 dblPctTVA =dblPCT_TVA2 ;
 bFlagSaisieOK = true;
 break;
case 3 :
 dblPctTVA =dblPCT_TVA3 ;
 bFlagSaisieOK = true;
 break;
default :
 System.out.println("Saisie du taux incorrecte");
}

if (bFlagSaisieOK) {
 System.out.println("Entrez le prix brut");
 dblPrixBrut= SaisieUtilisateur.nextDouble();
 dblPrixNet = dblPrixBrut + dblPrixBrut * (dblPctTVA/100);
 System.out.println("Pour un prix brut de " + dblPrixBrut + " € le prix net est de " + dblPrixNet + " € (TVA " + dblPctTVA + "%) ");
}

SaisieUtilisateur.close();
}
}
```

**4.10.2.2. Pseudocode et code Java****SELON** *identificateur de variable*

Valeur 1 : instructions

Valeur 2 : instructions

... : ...

Autrement : instructions

**FIN\_SELON****4.10.2.3. Exercice**

Réalisez l'exercice PJ1Ex1-11.

PJ1Ex1-11. L'utilisateur entre le N° de consommation répondant à l'invitation « *Entrez le N° de consommation* ». Selon le N° choisi, le programme affiche le nom de la consommation et son prix de vente qui peut être éventuellement réduit. Dans le cas où le N° n'existe pas, le programme affiche "*N° non référencé*" et se termine. Créez une fonction *getUser\_intInput()* qui permet de récupérer un entier saisi par l'utilisateur avec comme paramètres : le message d'invitation, une valeur minimum et une valeur maximum admises (de 1 à 4 dans ce cas). Cette fonction renvoie -1 si le N° n'est pas correct. Après avoir choisi le N° de consommation, l'utilisateur doit répondre à la question "*Happy Hour ? Y/N*". Créez une fonction *checkAnswer()* ayant comme paramètres une question et une réponse attendue, qui renvoie vrai ou faux selon que la réponse de l'utilisateur correspond (ou pas) à la réponse attendue ("Y" dans ce cas ci). Si l'utilisateur a répondu Y, le prix affiché est réduit de 50% par rapport au prix normal, autrement le prix indiqué est le prix normal. Parmi les N° de consommations, on considère seulement la N°1 Eau plate 3,00€, N°2 Coca Cola 3,00€, N°3 Bière pression 2,80€, N°4 Café long 2,90€. On doit obligatoirement recourir à la structure "switch case" pour la sélection de la consommation, la structure "if" pour les autres conditions rencontrées. Le programme affiche "*Prix de la consommation X : Y €*" avec X, le nom de la consommation et Y le prix de celle-ci.

Quels sont les tests à réaliser pour valider le programme ?

Après réalisation du programme, critiquez la manière de stocker, d'exploiter les données des consommations et son implication dans la manière de devoir coder.

### 4.10.3. L'opérateur ternaire ? :

Cet opérateur permet d'alléger le nombre d'instructions lorsque l'on a deux choix possibles d'affectation d'une variable suivant une condition logique (booléenne). Le code suivant illustre différentes manières d'afficher le prix d'un billet d'entrée selon que la personne est majeure ou mineure.

```
import java.util.Scanner;

public class TestOperateurTernaire {

 public static void main(String[] args) {

 int age = 0;
 double prixBillet = 0.0;

 Scanner sc = new Scanner(System.in);
 System.out.println("Entrez votre âge");
 age = sc.nextInt();

 //version avec structure classique if..else
 if (age < 18) {
 prixBillet = 15.0;
 } else {
 prixBillet = 22.0;
 } //end if
 System.out.println("Prixbillet :" + prixBillet + "€");

 //version avec opérateur ternaire
 prixBillet = age<18 ? 15.0 : 22.0;
 System.out.println("Prixbillet :" + prixBillet + "€");

 //version encore plus concise avec opérateur ternaire
 System.out.println("Prixbillet :" + (age<18 ? 15.0 : 22.0) + "€");

 } //end main

} //end class
```

L'opérateur ternaire s'exprime sous la forme :

$$\text{NomDeVariable} \leftarrow (\text{expression booléenne}) ? \text{valeur 1} : \text{valeur 2}$$

Si l'expression booléenne est vraie, la *valeur 1* définie à gauche du **:** est affectée à la variable définie à gauche du signe d'affectation. Si l'expression est fausse, c'est la *valeur 2* à droite du **:** qui lui est affectée.



## 4.11. Tableaux (Arrays)

Il est possible de stocker des informations de même type sous forme de tableaux. Pour illustrer cela, traitons d'abord le cas d'un tableau ne comprenant qu'une seule dimension.

### 4.11.1. Tableau unidimensionnel

L'exercice PJ1EX1-12 est une évolution de l'exercice PJ1Ex1-10 en utilisant un tableau pour stocker les taux de TVA. Reprenons donc l'exercice PJ1Ex1-10 et voyons comment diminuer le nombre d'instructions en utilisant un tableau. Pour ce faire, on va stocker dans un tableau à une dimension (1 colonne et 3 lignes) les taux de TVA existants. La déclaration et initialisation peuvent se faire indifféremment de deux façons :

```
double[] VAT = new double[3];
VAT[0]=6.0 ;
VAT[1]=12.0 ;
VAT[2]=21.0 ;
```

ou plus rapidement :

```
VAT = {6.0, 12.0, 21.0};
```

Dans le premier cas, on instancie (**new**) un objet tableau à 3 lignes, puis on affecte des valeurs dans chaque case de celui-ci. On est clairement dans l'esprit de la programmation objet où un tableau est un objet.

Dans le 2<sup>ème</sup> cas, on crée le tableau et on le remplit en une seule ligne de code. Cette solution est envisageable étant donné qu'on connaît, dès le départ, le contenu du tableau (ce ne sera pas toujours le cas).

Remarquez que la taille d'un tableau ne peut plus changer pendant l'exécution du programme et c'est un des principaux inconvénients de celui-ci.

La visualisation du tableau créé précédemment est la suivante :

|          |      |
|----------|------|
| VAT[0] : | 6.0  |
| VAT[1] : | 12.0 |
| VAT[2] : | 21.0 |

**Attention, la première case porte l'index 0 et non 1 !**

Pour modifier et afficher le contenu de la première case du tableau on utilise les instructions

```
VAT[0] = 8.5 ;
System.out.println (VAT[0]) ;
```

Grâce à ce tableau, on remplace avantageusement les lignes de codes surlignées en jaune (de l'exercice PJ1Ex1-10) servant à la sélection de la TVA choisie.

```
static double getVATChoice() {
 final double VAT1 = 6.0, VAT2 = 12.0, VAT3=21.0;
 int userChoice=0;

 System.out.println("Pour une TVA à " + VAT1 + "% tapez 1, "
```

```

 + "pour " + VAT2 + "% tapez 2 et pour " + VAT3 + "%, tapez
3");
Scanner sc = new Scanner (System.in);
intUserChoice = sc.nextInt();
if (userChoice == 1) {
 return VAT1;
} else if (userChoice == 2) {
 return VAT2;
} else if (userChoice == 3) {
 return VAT3;
} else {
 return -1.0;
} //end if
} //end getVATChoice

```

*Solution getVATChoice() Ex PJ1Ex1-10*

```

final double VAT[] = {6.0, 12.0, 21.0};
int intUserChoice=0;

System.out.println("Pour une TVA à " + VAT[0] + "% tapez 1, "
 + "pour " + VAT[1] + "% tapez 2 et pour " + VAT[2] + "%,
tapez 3");
Scanner sc = new Scanner (System.in);
intUserChoice = sc.nextInt();

if (userChoice >= 1 && userChoice <= VAT.length) {
 return VAT[userChoice-1];
} else {
 return -1.0;
} //end if
} //end getVATChoice

```

*Soluion getVATChoice() Ex PJ1Ex1-12 utilisation d'un tableau*

On appelle cette façon de procéder `VAT[userChoice-1]` : adressage indirect. On adresse indirectement l'index d'une case du tableau via une variable `userChoice-1`. Remarquez que si on ajoutait encore 10 autres taux de TVA, il faudrait ajouter 10 conditions dans la première version du programme alors que, dans la 2<sup>ème</sup>, il suffirait d'ajouter les nouveaux taux de TVA à l'initialisation du tableau sans devoir ajouter une seule ligne de code.

#### 4.11.1.1. Pseudocode

La déclaration et initialisation d'un tableau à n lignes s'exprime par :

|                                                                                                              |
|--------------------------------------------------------------------------------------------------------------|
| Identificateur de type : NomTableau [valeur <sub>1</sub> , valeur <sub>2</sub> , ... , valeur <sub>n</sub> ] |
|--------------------------------------------------------------------------------------------------------------|

Exemple : Réel double précision : VAT[6.0, 12.0, 21.0]

Pour accéder à une case d'un tableau en écriture, on indique :

|                          |
|--------------------------|
| NomTableau [index] ← ... |
|--------------------------|

**4.11.1.2. Exercices**

Réalisez les exercices PJ1Ex1-12 et PJ1Ex1-13.

PJ1Ex1-12 Le programme est identique à l'exercice PJ1Ex1-10 mais on stocke les trois taux de TVA prédéfinis dans un tableau. Pour ce faire, déclarez un tableau `VAT[]` et modifiez la fonction `getVATChoice()` développée à l'exercice PJ1Ex1-10.

PJ1Ex1-13 Le programme est identique à l'exercice PJ1Ex1-11. mais on doit stocker les informations des consommations (dénominations et prix) dans deux tableaux `Names[]` et `Prices[]`. L'utilisation du "switch case" n'est, par contre, plus du tout requise.

Rappel PJ1Ex1-11. L'utilisateur entre le N° de consommation répondant à l'invitation « *Entrez le N° de consommation* ». Selon le N° choisi, le programme affiche le nom de la consommation et son prix de vente qui peut être éventuellement réduit. Dans le cas où le N° n'existe pas, le programme affiche "*N° non référencé*" et se termine. Créez une fonction `getUser_intInput()` qui permet de récupérer un entier saisi par l'utilisateur avec comme paramètres : le message d'invitation, une valeur minimum et une valeur maximum admises (de 1 à 4 dans ce cas). Cette fonction renvoie -1 si le N° n'est pas correct. Après avoir choisi le N° de consommation, l'utilisateur doit répondre à la question "*Happy Hour ? Y/N*". Créez une fonction `checkAnswer()` ayant comme paramètres une question et une réponse attendue, qui renvoie vrai ou faux selon que la réponse de l'utilisateur correspond (ou pas) à la réponse attendue ("Y" dans ce cas ci). Si l'utilisateur a répondu Y, le prix affiché est réduit de 50% par rapport au prix normal, autrement le prix indiqué est le prix normal. Parmi les N° de consommations, on considère seulement la N°1 Eau plate 3,00€, N°2 Coca Cola 3,00€, N°3 Bière pression 2,80€, N°4 Café long 2,90€.

### 4.11.2. Tableaux multidimensionnels

Dans les cas précédents, on a envisagé uniquement des tableaux à une dimension constitués de plusieurs lignes mais une seule colonne. Considérons maintenant un tableau contenant  $i$  lignes et  $j$  colonnes. L'instruction suivante permet de déclarer un tableau à 6 lignes et 4 colonnes destiné à contenir des réels doubles :

```
double [][] myTab = new double[6][4];
```

La représentation mentale de ce tableau doit être celle-ci :

|  |       |  |  |
|--|-------|--|--|
|  |       |  |  |
|  |       |  |  |
|  |       |  |  |
|  | 16.72 |  |  |
|  |       |  |  |
|  |       |  |  |

Pour placer une valeur dans une des cases, il faut garder à l'esprit que les index de ligne et de colonne commencent à 0. Pour placer la valeur 16.72 à la 4<sup>ème</sup> ligne et 2<sup>ème</sup> colonne, on utilise donc l'instruction suivante :

```
myTab[3][1]=16.72 ;
```

Lorsqu'on connaît le contenu du tableau à l'avance, la déclaration peut se faire de cette manière :

```
int [][] intTab = {{0,2,4,8}, {1,10,2,7},{8,9,0,-1},{12,1,14,-4},{15,14,16,1}};
```

On a un tableau à deux dimensions, 5 lignes et 4 colonnes dont les valeurs sont prédéfinies :

|    |    |    |    |
|----|----|----|----|
| 0  | 2  | 4  | 8  |
| 1  | 10 | 2  | 7  |
| 8  | 9  | 0  | -1 |
| 12 | 1  | 14 | -4 |
| 15 | 14 | 16 | 1  |

Remarquez que les contenus des éléments d'un tableau sont tous de même type ce qui ajoute un second inconvénient à la structure tableau. Le premier était le fait que, comme les tableaux unidimensionnels, les tableaux multidimensionnels ne peuvent changer de taille après leur déclaration. Dans certains cas, on peut néanmoins choisir la taille du tableau en fonction du contexte et du déroulement du programme. On a dit que la déclaration des variables devait se faire en tête de programme et c'est ce qu'un programmeur consciencieux doit continuer de faire. Mais, en réalité, java accepte qu'une déclaration soit faite n'importe où dans le programme ! Il est donc possible d'utiliser une (plusieurs) variables qui détermine(nt) la taille du tableau et de déclarer le tableau, non au début, mais dans la partie instructions du programme.

Le programme suivant illustre un tableau dont la taille n'est pas fixée avant exécution mais définie par l'utilisateur.

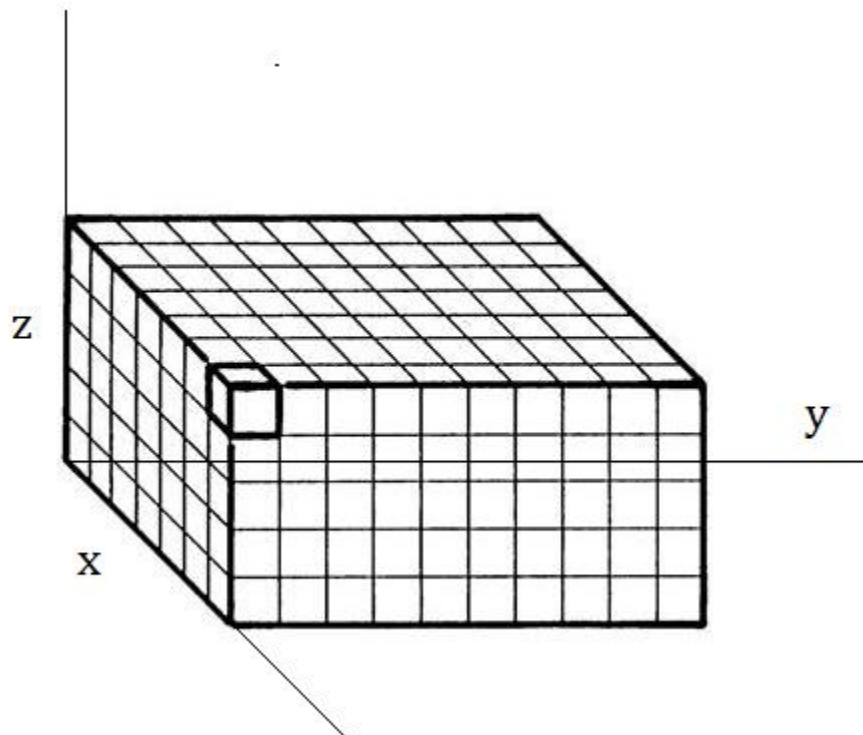
```
int i = 0;
int j = 0;
Scanner sc = new Scanner(System.in);
System.out.println("Entrez le nombre de lignes du tableau");
i = sc.nextInt();
System.out.println("Entrez le nombre de colonnes du tableau");
j = sc.nextInt();

double [][] myTab = new double[i][j];
```

Il est à remarquer qu'on n'a pas résolu le problème de taille des tableaux puisque le tableau, même si la taille peut être paramétrable à la déclaration, ne peut toujours pas évoluer après celle-ci.

Le nombre de crochets indique le nombre de dimensions. Même si les tableaux à deux dimensions sont les plus utilisés puisque nous avons l'habitude de synthétiser nos informations sur des tableaux écrits sur une feuille de papier à deux dimensions, il est possible de déclarer des tableaux dont la dimension est supérieure à 2. L'exemple ci-dessous illustre la représentation d'un tableau à 3 dimensions dont chaque cube serait la référence à une case mémoire du tableau. Dans chacun de ces cubes, on peut stocker une valeur dépendante du type de donnée.

```
int x=7 ;
int y=10 ;
int z=5 ;
double [][][] myTab = new double[x][y][z];
```



**4.11.2.1. Pseudocode**

La déclaration et initialisation d'un tableau à deux dimensions comportant  $i$  lignes et  $j$  colonnes dont les valeurs sont connues, s'exprime par :

Identificateur de type :  $\text{NomTableau}[(\text{val}_{11}, \text{val}_{12}, \dots, \text{val}_{1j}), (\text{val}_{21}, \text{val}_{22}, \dots, \text{val}_{2j}), \dots, (\text{valeur}_{i1}, \text{val}_{i2}, \dots, \text{valeur}_{ij})]$

Exemple : Entier :  $\text{MonTableau}[(1,32),(30,-2),(16,2),(5,18)]$

Pour la déclaration d'un tableau à deux dimensions comportant  $i$  lignes,  $j$  colonnes et dont les valeurs ne sont pas connues dès le départ, on notera :

Identificateur de type :  $\text{NomTableau } [i] [j]$

Exemple : Entier :  $\text{MonTableau}[4][3]$

Pour accéder à une case d'un tableau en écriture, on indique :

$\text{NomTableau}[\textit{index ligne}][\textit{index colonne}] \leftarrow \dots$

Exemple :  $\text{MonTableau}[1][2] \leftarrow 28$

#### 4.11.2.2. Exercices

Appliquons cette notion à notre projet brasserie et examinons l'énoncé de l'exercice PJ1Ex1-14.

PJ1Ex1-14 : Le programme contient un tableau à deux dimensions avec les N° d'emplacements des consommables dans la 1<sup>ère</sup> colonne et le nombre d'unités en stock dans la 2<sup>ème</sup>. Un autre tableau contient les dénominations des consommables.

Les tableaux sont initialisés comme suit :

| index | N°emplacement | Unités en stock |
|-------|---------------|-----------------|
| 0     | 19            | 92              |
| 1     | 6             | 16              |
| 2     | 14            | 27              |
| 3     | 4             | 72              |

| index | Dénomination   |
|-------|----------------|
| 0     | Eau plate      |
| 1     | Coca Cola      |
| 2     | Bière pression |
| 3     | Café long      |

L'utilisateur entre le N° d'identifiant (1 pour l'eau plate, 2 pour le coca, ...) du consommable répondant à l'invitation *"Entrez le N° d'identifiant"*. Si le N° d'identifiant est non référencé dans les tables, le programme indique *"N° non référencé"* puis s'arrête. (Pour cette entrée, utilisez la fonction déjà développée). Ensuite, l'utilisateur entre le nombre d'unités consommées répondant à l'invitation *"Entrez le nombre d'unités consommées pour X. Quantité actuellement en stock Y, emplacement Z"* (avec X le nom du consommable, Y la quantité en stock avant la modification et Z le N° d'emplacement). Le stock s'en trouve diminué de la valeur saisie. Si le nombre saisi dépasse la quantité actuelle en stock, le programme affiche *"Erreur de saisie ou de quantité en stock pour X ! Annuler ou Mettre à Zero le stock. A/Z"*. Pour ce faire, réutilisez la fonction *checkAnswer()* développée pour l'exercice PJ1Ex1-13. Dans le cas où l'utilisateur n'a pas demandé la mise à zéro du stock, le programme n'impacte pas le stock et affiche le message *"Modification annulée"* ; en cas de remise à zéro, le stock pour ce produit est mis à 0 et le programme affiche *"Stock 0 pour X"*. Le programme affiche, au début et à la fin du programme, un récapitulatif, ligne par ligne, avec l'identifiant, le nom de toutes les consommations avec leurs quantités respectives en stock (en tenant compte de la modification effectuée à la fin). Pour ce faire, créez une procédure *showStock()*. Les tableaux peuvent être déclarés en variables globales ; dans ce cas, ils ne doivent pas être passés en paramètre à la procédure.

Le programme affiche, au début et à la fin du programme, un récapitulatif, ligne par ligne, avec l'identifiant, le nom de la consommation, l'emplacement et la quantité en stock (en tenant compte de la modification effectuée à la fin). Pour ce faire, créez une procédure *showStock()*. Les tableaux peuvent être déclarés en variables globales ; dans ce cas, ils ne doivent pas être passés en paramètres à la procédure.

On a donc, en plus du tableau des noms de consommation, un tableau à deux dimensions contenant 4 lignes et 2 colonnes : le N° d'emplacement du consommable dans la première colonne et le nombre d'unités en stock dans la deuxième :

```
19 92
6 16
14 27
4 72
```

En java, ce tableau peut se déclarer et s'initialiser comme suit :

```
int Stock [][] = {{19,92}, {6,16},{14,27} ,{4,72} };
```

L'accès à une case du tableau se réalise en faisant référence à un index de ligne et de colonne.

Ainsi, `intStock [2][1]` par exemple, fait référence à la case située à la 3<sup>ème</sup> ligne, 2<sup>ème</sup> colonne (puisque l'index commence à 0 et non à 1) et contient la valeur 27.

Réalisez l'exercice PJ1Ex1-14 en tenant compte de ces informations.



## 4.12. Les structures répétitives

Jusqu'à présent, chaque ligne d'instruction de nos codes n'a été exécutée au plus qu'une seule fois. Lorsqu'un bloc de lignes de code doit être exécuté plusieurs fois, en boucle, alors on parle de structures répétitives (ou boucles) dont il en existe trois types.

### 4.12.1. La boucle REPETER (faire) ... TANT QUE (jusqu'à ce que)

#### 4.12.1.1. Exemple

Pour illustrer la boucle REPETER (ou faire) TANT QUE, prenons l'exercice PJ1Ex1-15. Ce programme est semblable à l'exercice PJ1 Ex1-9 mais pour l'instant sans utiliser les fonctions qui avaient été développées.

L'utilisateur saisit un entier correspondant au N° de table où les consommations ont été prises répondant à l'invitation « *Entrez le N° de table* ». Le programme vérifie si le N° de table entré est correct. Si le N° est inférieur à 1, le programme affiche « *Saisie incorrecte ! Le N° de table doit être supérieur ou égale à 1* », si le nombre est supérieur à 20, on affiche « *Saisie incorrecte ! Le N° table le plus élevé est 20. Contactez l'administrateur du programme si vous souhaitez augmenter ce nombre* », si le N° est correct, le programme affiche « *Vous avez entré le numéro de table X* » avec X le N° de table entré. La différence avec l'exercice PJ1 EX1-9 réside dans le fait que, aussi longtemps que le N° de table entré n'est pas valide, le programme repose la question « *Entrez le N° de table* ».

Le code ci-dessous constitue une solution possible de l'exercice PJ1 Ex1-9 (sans fonction).

```
public class Ex9 {

 public static void main(String[] args) {

 int numTable=0;
 final int intNUM_TABLE_MAX = 20;
 Scanner sc = new Scanner(System.in);

 System.out.println("Entrez le N° de table");
 numTable= sc.nextInt();

 if (numTable < 1) {

 System.out.println("Saisie incorrecte ! Le N° de table doit être supérieur
ou égal à 1");

 System.out.println("Vous avez entré le N° de table " + numTable);

 }else if (numTable < 1 && numTable <= intNUM_TABLE_MAX) {

 System.out.println("Saisie incorrecte ! Le N° table plus élevé est " +
intNUM_TABLE_MAX + "\n" Contactez l'administrateur du programme si vous
souhaitez augmenter ce nombre ");

 } else {
 System.out.println("Vous avez entré le N° de table " + numTable + " et le
programme peut continuer");
 }
 } //end main
} //end class
```

Dans notre nouvel exercice, la partie d'instructions dans le cadre vert doit se répéter tant que le N° de table n'est pas valide puisque c'est ce que demande le cahier des charges.

On va donc ajouter l'instruction *faire* (**do** { }) **au début** de ce bloc et *tant que le N° n'est pas valide* (**while** (numTable < 1 || numTable > intNUM\_TABLE\_MAX); ) **à la fin** de celui-ci.

Modifiez le programme ci-dessus (tel quel, en faisant un copier-coller, avec tout le code dans le `main()`) en tenant compte de ces informations et testez-le de différentes manières (en entrant volontairement des N° de table erronés), pas à pas, en mode debug.

En analysant la structure d'une boucle REPETER...TANT QUE, on peut dire que le bloc d'instructions à l'intérieur de la boucle est exécuté **au moins une fois** étant donné que la condition de poursuite de boucle se trouve à la fin de celle-ci.

**C'est une caractéristique essentielle et qui guidera notre choix vers l'utilisation ou pas de ce type de boucle.**

Les instructions à l'intérieur de la boucle se répètent tant que l'expression booléenne est **vraie**.

#### 4.12.1.2. Pseudocode

En pseudocode, la boucle REPETER (faire) TANT QUE s'exprime par :

##### REPETER

instructions ...

**TANT QUE** (expression booléenne)

#### 4.12.1.3. Exercices

Réalisez les exercices PJ1Ex1-15, PJ1Ex1-16 et PJ1Ex1-17.

PJ1Ex1-15. Modifiez la fonction `getUser_intInput()` pour en créer une nouvelle `get_intInput()` (de façon à ne pas fausser tous les programmes précédents qui utilisaient cette fonction) qui oblige l'utilisateur et donc la fonction à retourner un entier valide (dans ce cas, un N° de table) compris dans l'intervalle minimum et maximum passés en paramètres.

PJ1Ex1-16 Le programme indique que le stock, pour une consommation précise, est constant et fixé à 100 unités initialement. Au fur et à mesure des validations de ticket, le stock va diminuer. Réalisons donc un programme semblable à cela. L'utilisateur entre un nombre d'unités consommées répondant à l'invitation « *Entrez le nombre d'unités consommées* ». Le stock se trouve diminué de la quantité encodée, il est affiché le message. « *Nombre d'unités en stock XX* ». L'opération se répète jusqu'à ce que le stock soit à 0, alors le programme affiche « *Stock à 0 d'unités* ». Si jamais l'utilisateur entre un nombre plus grand que le stock actuel, le message d'erreur « *saisie incorrecte* » s'affiche mais les invitations « *Entrez le nombre d'unités consommées* » se poursuivent.

PJ1Ex1-17. Lors de l'élaboration du ticket de caisse, le serveur aura l'occasion de saisir des réponses aux invitations du programme telles que "Voici une question... / A pour Annuler, V pour

*valider, Q pour quitter le programme".* Créez une fonction *getUserSpecificInput(..., ...)* qui pose une question à l'utilisateur, et renvoie une réponse valide si elle correspond bien à la demande (A, V ou Q dans l'exemple) sans ambiguïté (donc ni "VQ" ni "AQ" ni "AVQ", ...une seule lettre est attendue). Dans le cas d'une réponse non valide, le message d'erreur s'affiche *"Erreur de saisie, votre choix doit être parmi XX..X, une seule lettre seulement"* (XX\_X est AVQ dans notre exemple) puis la question est reposée. Les possibilités de réponse ne sont pas limitées à trois, les caractères attendus doivent se trouver en paramètre (*expectedAnswers*) de la fonction rassemblés dans une chaîne de caractères "AVQ" pour notre exemple. La fonction renvoie une chaîne de caractères même si elle ne comprend qu'un seul caractère (donc soit "A", "V" ou "Q" pour l'exemple). L'appel de la fonction peut être :

```
getUserSpecificInput("Voici une question / A pour Annuler, V pour valider, Q
pour quitter le programme", "AVQ")
```

### 4.12.2. La boucle TANT QUE

La championne des boucles est sûrement la boucle TANT QUE car elle peut être utilisée dans tous les cas de figure, en lieu et place des deux autres boucles existantes sans pour autant que ce soit le meilleur choix ou disons le choix le plus judicieux si on veut être scolaire.

#### 4.12.2.1. Exemple

Pour illustrer la boucle TANT QUE et préciser sa différence avec la boucle REPETER TANT QUE, réalisons l'exercice PJ1Ex1-18 qui est une évolution de l'exercice PJ1Ex1-16. Ce programme permettait à l'utilisateur de diminuer, en boucle, un stock initial fixé à 100 unités en répondant à l'invitation « *Entrez le nombre d'unités consommées* ». Le programme s'arrêtait quand le stock était à 0. La boucle REPETER TANT QUE se justifiait puisque forcément, il fallait poser la question au moins une fois à l'utilisateur. Pour l'exercice PJ1Ex1-18, le stock initial n'est pas fixé à 100, c'est l'utilisateur qui encode le stock initial avec une valeur entière supérieure ou égale à 0.

Imaginons le cas particulier où le stock est initialisé à 0. Dans ce cas, la question "*Entrez le nombre d'unités*" n'a aucun sens et le bloc d'instructions à l'intérieur de la boucle ne doit même pas être exécuté une fois. On comprend qu'il faut vérifier dès le début de la boucle si le stock n'est pas nul. La condition de poursuite de boucle ne doit donc plus se situer à la fin des instructions de boucle mais au tout début de celles-ci. C'est là toute la différence et la seule différence avec la boucle REPETER TANT QUE. Pour cet exercice c'est la boucle TANT QUE qui doit être utilisée.

Pour la boucle **TANT QUE**, le test de poursuite de boucle se faisant au début de celle-ci, le bloc d'instructions à l'intérieur de la boucle peut être exécuté de **0** à N fois tandis que pour une boucle REPETER TANT QUE, il peut être exécuté de **1** à N fois. **C'est une caractéristique essentielle et qui guidera notre choix vers l'utilisation d'une boucle REPETER...TANT QUE ou d'une boucle TANT QUE**

En Java, les boucles TANT QUE et REPETER TANT QUE s'écrivent :

| Boucle TANT QUE                                | Boucle REPETER TANT QUE                         |
|------------------------------------------------|-------------------------------------------------|
| <pre>while (condition booléenne) { ... }</pre> | <pre>do{ ... }while(condition booléenne);</pre> |

#### 4.12.2.2. Pseudocode

En pseudocode, la boucle TANT QUE s'écrit :

**TANT QUE** (*expression booléenne*)

instructions  
...

**FIN TANT QUE**

### 4.12.2.3. Exercices

Réalisez les exercices PJ1Ex1-18 et PJ1Ex1-19.

PJ1Ex1-18 Le programme est semblable à l'exercice PJ1Ex1-16 mais, cette fois, le stock initial est saisi par l'utilisateur (au lieu d'être fixé à 100) répondant à l'invitation « *Entrez le stock initial* ». L'utilisateur entre ensuite le nombre d'unités consommées répondant à l'invitation « *Entrez le nombre d'unités consommées* ». Le stock se trouve diminué de la quantité encodée, il est affiché le message. « *Nombre d'unités en stock XX* ». L'opération se répète jusqu'à ce que le stock soit à 0. Dans ce cas le message « *Stock nul* » s'affiche. Si jamais l'utilisateur entre un nombre plus grand que le stock actuel, le message d'erreur « *saisie incorrecte* » s'affiche mais les invitations « *Entrez le nombre d'unités consommées* » se poursuivent. Remarque : il se peut que l'utilisateur entre un stock initial de 0, dans ce cas le message « *Stock nul* » s'affiche directement.

PJ1Ex1-19. Ce programme sert à pré visualiser toute la commande d'une table avant l'impression du ticket de caisse. La commande est déjà stockée dans un tableau d'entiers Order[][] à deux colonnes. La première colonne correspond au N° de la consommation qui est à une unité près l'index dans le tableau Names[] (la consommation N°1 porte l'index 0 dans le tableau Names[]). La deuxième colonne contient le nombre de consommations. Voici un exemple de contenu du tableau Order[][] :

|    |   |
|----|---|
| 3  | 2 |
| 1  | 3 |
| 12 | 4 |
| 37 | 1 |
| 36 | 3 |
| 0  | 0 |
|    |   |
|    |   |

```
int Order[][]={{3,2},{1,3},{12,4},{37,1},{36,3},{0,0},{0,0},{0,0},{0,0}};
```

Par convention, le tableau ne contient plus de données utiles quand on rencontre 0 et 0 sur une ligne. Créez une procédure *previewOrder()* utilisant la boucle TANT QUE, qui affiche le détail de la commande en indiquant le nom des consommations, le nombre d'unités et le prix net total par consommation. Le tableau Order[][] sera passé en paramètre à la procédure. Utilisez en variables globales les tableaux Names[] et NetPrices[] de l'exercice PJ1Ex1-27.

```
final static String Names[]= {"Spa reine 25 ","Bru plate 50","Bru pét 50","Pepsi","Spa orange",
"Schweppes Tonic","Schweppes Agr","Ice Tea","Ice Tea Pêche","Jus d'orange Looza", "Cécémel", "Red
Bull","Petit Espresso","Grand Espresso","Café décaféiné ","Lait Russe ","Thé et infusions","Irish
Coffee ","French Coffee ","Cappuccino","Cécémel chaud","Passione Italiano","Amour Intense",
"Rhumba Caliente ","Irish Kisses ","Cuvée Trolls 25","Cuvee Trolls 50","Ambrasse-Temps
25","Ambrasse-Temps 50 ","Brasse-Temps Cerises 25","Brasse-Temps Cerises 50","La Blanche Ste
Waudru 25","Blanche Ste Waudru 50","Brasse-Temps citr 25","Brasse-Temps citr 50","Spaghetti Bolo
","Tagl Carbonara","Penne poulet baslc ","Tagl American","Tagl saum"};
```

```
final static double NetPrices[]=
{2.2,2.3,3.9,2.2,2.2,2.6,2.6,2.6,2.6,2.6,2.6,4.5,2.2,2.2,2.2,2.5,2.5,7.0,7.0,2.8,2.8,6.2,6.2,6.2,6
.2,2.9,5.5,2.7,5.1,3.1,5.8,2.6,4.9,2.6,4.9,10.8,11.2,12.2,14.5,16.9};
```

Pour l'exemple, l'affichage est tel que celui-ci :

```
Bru pét 50 3.9 2 7,80
Spa reine 25 2.2 3 6,60
Red Bull 4.5 4 18,00
Tagl Carbonara 11.2 1 11,20
Spaghetti Bolo 10.8 3 32,40
```

### 4.12.3. La boucle FOR

La boucle FOR est normalement utilisée lorsque le nombre de passages dans la boucle est connu à l'avance. Elle gère obligatoirement un compteur de boucle qui se trouve dans l'en-tête de celle-ci et non à l'intérieur du bloc d'instructions comme c'est le cas dans les autres boucles (REPETER et TANT QUE).

Dans les exercices précédents sur les boucles REPETER ... TANT QUE et TANT QUE, le nombre de passages dans la boucle n'était pas connu avant l'exécution du programme car il dépendait des saisies de l'utilisateur. La boucle FOR n'était donc pas un choix indiqué suivant la définition qui précède. Par rapport à d'autres langages, Java (C++) a cependant inséré la possibilité et d'ailleurs la nécessité d'intégrer une condition de sortie de boucle ce qui la rend utilisable même si le nombre de passages n'est pas forcément connu.

#### 4.12.3.1. Exemple

Pour illustrer un exemple d'utilisation de la boucle FOR, examinons l'exercice PJ1-EX1-20. PJ1Ex1-20. Les noms des consommations sont stockés dans un tableau *Names*[] et leur prix net dans un autre tableau *NetPrices*[] (Voir Ex PJ1Ex1-27). En utilisant une boucle FOR, réalisez une procédure *showMenu()* qui affiche l'entièreté du menu. L'affichage comprend la liste des consommations avec leur prix net sous la forme ci-dessous :

```
Spa reine 25 cl 2,20 €
Bru plate 50 cl 2,3 €
Bru légèrement pétillante 50 cl 3,9 €
Pepsi, Pepsi max 2,20 €
Spa orange 2,20€
....
```

L'analyse de l'exercice permet de dire qu'on connaît à l'avance le nombre de lignes puisque les données de la carte sont connues, fixées dans un tableau. Nous sommes typiquement dans le cadre d'utilisation d'une boucle FOR.

En java, le programme demandé peut s'écrire comme suit :

```
final static String strNames[]= {"Spa reine 25 ", "Bru plate 50", "Bru pét 50",
"Pepsi", "Spa orange", "Schweppes Tonic", "Schweppes Agr", "Ice Tea", "Ice Tea Pêche",
"Jus d'orange Looza", "Cécémel", ...};
double static NetPrices[]= {2.2, 2.3, 3.9, 2.2, 2.2, ...};

for(int i=0; i < NetPrices.length; i=i+1) {
 System.out.println(Names[i] + " " + NetPrices[i]+"€");
} //fin for
```

Testez le programme tel quel. Remplacez ensuite successivement (pas en même temps) les termes

- a) `int i=0` par `int i=1`
- b) `i < db1PrixNet.length` par `i < 2`
- c) `i=i+1` par `i=i+2`

et en déduire leurs rôles et fonctionnement respectifs en comparant le résultat de l'exécution par rapport à l'exécution du programme initial.

#### 4.12.3.2. Pseudocode

En pseudocode, la boucle FOR s'écrit :

**POUR** i ALLANT DE n A m PAR PAS DE x

instructions

...

**FIN POUR**

#### 4.12.3.3. Remplacement d'une boucle FOR par TANT QUE

On a dit que la boucle TANT QUE pouvait être utilisée dans tous les cas. En effet, en gérant l'initialisation et l'incrément du compteur, la boucle TANT QUE réalise la même fonctionnalité qu'une boucle FOR. L'exercice précédent devient :

```

public class Ex21_Avec_TANT_QUE {

String static Names[]= {"Spa reine 25 cl","Bru plate 50 cl","Bru légèrement
pétillante 50 cl","Pepsi, Pepsi max","Spa orange",...};
Double static NetPrices[]= {2.2, 2.3, 3.9, 2.2, 2.2,...};

public static void main(String[] args) {

int i=0;//initialisation du compteur de boucle

while(i < dblPrixNet.length) {

 System.out.println(Names[i] + " " + NetPrices[i]+"€");
 i=i+1; //incrément du compteur de boucle

} //fin while

} //fin main
} //fin class

```

#### 4.12.3.4. Boucle FOR EACH - POUR CHAQUE...DANS

Il existe une boucle POUR CHAQUE...DANS, intégrée dans la boucle FOR en Java. C'est aussi cette façon de faire qui permet de gérer le parcours des collections d'objets (voir POO). Elle doit être utilisée en lecture des éléments à parcourir et non en écriture.

Testez le code suivant et interprétez le fonctionnement et l'écriture du **for**.

```

public class Test2 {

 public static void main(String[] args) {

String Names[]= {"Spa reine 25 cl","Bru plate 50 cl","Bru légèrement
pétillante 50 cl","Pepsi, Pepsi max", "Spa orange"};

for(String str : Names) {

 System.out.println(str);

} //fin for
} //fin main
} //fin class

```

En pseudo code, la boucle FOR EACH peut s'écrire :

**POUR CHAQUE ELEMENT** *TypeDeDonne : NomElement* **DANS** *Collection*

Instructions

**FIN POUR**

Exemple :

**POUR CHAQUE** Chaîne : str **DANS** Names[]

Afficher (str)

**FIN POUR**

#### 4.12.3.5. Exercice sur les boucles for

Réalisez l'exercice PJ1Ex1-21.

PJ1Ex1-21 Les nom des consommations sont stockés dans un tableau identique à celui de l'exercice précédent. L'utilisateur doit encoder le stock pour toutes les consommations répondant à l'invitation « *Entrez le stock pour X* » avec *X* le nom de la consommation.

Le programme affiche ensuite un récapitulatif avec ligne par ligne, le nom des consommations avec le stock encodé précédemment. Réalisez le programme d'abord uniquement avec des boucles FOR, puis uniquement avec des boucles TANT QUE. Limitez les tableaux à 5 éléments de façon à raccourcir le temps de saisie des stocks.

--Récapitulatif--

```
Spa reine 25 cl Stock : 45
Bru plate 50 cl Stock : 120
Bru légèrement pétillante 50 cl Stock : 200
Pepsi, Pepsi max Stock : 240
Spa orange Stock : 90
```



#### 4.12.4. Les boucles imbriquées

##### 4.12.4.1. Exemple

Il arrive fréquemment que l'on trouve une boucle (ou plusieurs) à l'intérieur d'un autre boucle. On parle alors de boucles imbriquées. Prenons l'exemple de l'exercice PJ1 Ex1-22

PJ1 Ex1-22. On veut parcourir et afficher le contenu du tableau ci-dessous. Celui-ci contient le N° d'identifiant, les quantités actuellement en stock, les quantités à prévoir et les quantités vendues la semaine cours, ceci pour chaque identifiant de consommation. Un autre tableau contient les noms des consommations. Le programme doit afficher ligne par ligne, le N° d'identifiant, le nom de la consommation et les différentes quantités en stock correspondantes séparées par un espace. L'affichage doit donc être de ce type :

N° 1 Spa reine 25 cl 56 200 55

N° 2 Bru Plate 50 cl 42 200 60

...

| Index<br>tableau | Identifiant | Quantité en<br>stock | Quantité à<br>prévoir | Quantités vendues<br>semaine en cours | Index<br>tableau | Consommable                     |
|------------------|-------------|----------------------|-----------------------|---------------------------------------|------------------|---------------------------------|
| 0                | 1           | 56                   | 200                   | 55                                    | 0                | Spa reine 25 cl                 |
| 1                | 2           | 42                   | 200                   | 60                                    | 1                | Bru plate 50 cl                 |
| 2                | 3           | 62                   | 200                   | 125                                   | 2                | Bru légèrement pétillante 50 cl |
| 3                | 4           | 45                   | 200                   | 150                                   | 3                | Pepsi, Pepsi max                |
| 4                | 5           | 25                   | 200                   | 140                                   | 4                | Spa orange                      |
| 5                | 6           | 72                   | 200                   | 86                                    | 5                | Schweppes Tonic                 |
| 6                | 7           | 40                   | 200                   | 47                                    | 6                | Schweppes Agrumes               |
| 7                | 8           | 48                   | 200                   | 80                                    | 7                | Lipton Ice Tea                  |
| 8                | 9           | 24                   | 150                   | 75                                    | 8                | Lipton Ice Tea Pêche            |
| 9                | 10          | 36                   | 200                   | 90                                    | 9                | Jus d'orange Looza              |
| 10               | 11          | 15                   | 100                   | 55                                    | 10               | Cécémel                         |
| 11               | 12          | 25                   | 80                    | 23                                    | 11               | Red Bull                        |

Si l'on veut afficher ligne par ligne toutes les quantités (en stock, à prévoir et vendues) pour chaque consommation, il suffit de parcourir les lignes et pour chaque ligne de parcourir toutes les colonnes.

L'algorithme va s'écrire comme suit :

**POUR i ALLANT DE 0 à 11 PAR PAS DE 1**

Afficher "N° identifiant " + Stock[i,0] + " " + Names[i]

**POUR j ALLANT DE 1 à 3 PAR PAS DE 1**

Afficher (" " + Stock[i,j])

**FIN POUR**

**FIN POUR**

On a donc deux boucles FOR : une dont le compteur i est l'indice de ligne et une autre boucle FOR imbriquée dont le compteur j est l'indice de colonne.

#### 4.12.4.2. Exercices

Réalisez l'exercice PJ1 Ex1-22 en terminant l'algorithme et le code initiés au point précédent. Testez en mode debug pas à pas pour visualiser la séquence des instructions exécutées dans les boucles imbriquées.

Réalisez ensuite l'exercice PJ1 Ex1-23.

PJ1 Ex1-23 Le programme est semblable à l'exercice PJ1 Ex1-22 mais l'affichage à la console est amélioré pour obtenir un alignement correct des données. Ce programme servira également de base pour l'affichage des données du ticket de caisse. Les unités de la première colonne de nombres sont alignées au 35<sup>ème</sup> caractère ; au 40<sup>ème</sup> et 45<sup>ème</sup> caractère pour les deux autres colonnes d'entiers. Les textes sont donc alignés à gauche et les nombres à droite. Le N° d'identifiant ne doit plus être affiché.

Créez une fonction `placeNumberRank(..., ..., ...)` qui ajoute un nombre à la fin d'une chaîne de caractères existante en spécifiant la position du chiffre de droite (paramètre *rank*, 35 pour les nombres de la première colonne par exemple). Cette fonction renvoie une chaîne de caractères contenant la chaîne originale à laquelle on a ajouté le nombre à la bonne position. Pour rendre la fonction utilisable pour des entiers, et des réels, le nombre est passé en paramètre sous forme de chaîne de caractères. Conseil : un espace est un caractère, invisible mais un caractère tout de même !

Exemple : la chaîne initiale "Spa Reine" doit devenir

"Spa Reine 56" après un appel de la fonction  
`placeNumberRank(..., ..., ...)`

L'affichage complet doit donc être celui-ci dessous. A la première ligne, le 6 de 56 se trouve au 35<sup>ème</sup> caractère, le 0 de 200 au 40<sup>ème</sup> caractère et le 5 de 45 au 45<sup>ème</sup> caractère.

|                           |    |     |     |
|---------------------------|----|-----|-----|
| Spa Reine                 | 56 | 200 | 55  |
| Bru Plate                 | 42 | 200 | 60  |
| Bru légèrement pétillante | 62 | 200 | 125 |
| Pepsi                     | 45 | 200 | 150 |
| Spa Orange                | 25 | 200 | 140 |
| Schweppes Tonic           | 72 | 200 | 86  |
| Schweppes Agrumes         | 40 | 200 | 47  |
| Lipton Ice Tea            | 48 | 200 | 80  |
| Lipton Ice Tea Pêche      | 24 | 150 | 126 |
| Jus d'orange Looza        | 36 | 200 | 164 |
| Cécémel                   | 15 | 100 | 85  |
| Red Bull                  | 25 | 80  | 23  |

### 4.13. Gestion des exceptions

Les programmes doivent prévoir les opérations risquées qui potentiellement peuvent échouer. Généralement, toutes les saisies de l'utilisateur peuvent ne pas correspondre aux attentes du programme. Pour l'instant, une saisie incorrecte d'un utilisateur conduit à un plantage et un arrêt prématuré de l'application. Voyons comment le programme suivant traite une erreur éventuelle qui devient alors non plus un bug mais une *Exception java* puisqu'elle est prévue et gérée par le programme. Testez le programme en entrant volontairement des saisies incorrectes. Interprétez la signification et le rôle du `try` et du `catch`.

```
import java.util.Scanner;

public class TestTryCatch {

 public static void main(String[] args) {

 try {
 System.out.println("Entrez le N° du consommable");
 Scanner sc = new Scanner(System.in);
 int intSaisieNumConso = sc.nextInt();
 sc.close();

 System.out.println("valeur saisie du N° de consommable : " +
 intSaisieNumConso);

 } catch (Exception e) {
 System.out.println("Exception récupérée " + e);
 }

 } //fin catch
} //fin main
} //fin class
```

#### 4.13.1. Pseudo code

En pseudocode, cette gestion d'erreur peut se traduire par :

##### ESSAYER

instructions  
...

##### FIN ESSAYER

##### EN CAS D'ERREUR FAIRE

instructions  
...

##### FIN EN CAS D'ERREUR

#### 4.13.2. Exercices

Modifiez la fonction de saisie `get_intInput(...)` de l'exercice PJ1 Ex21 de façon à ce qu'une erreur de type de donnée conduise maintenant à redemander une nouvelle saisie correcte et non à un plantage du programme.

#### 4.14. Les Arrays List

Lors de l'étude sur les tableaux, on a dit que l'inconvénient majeur de ceux-ci réside dans le fait qu'ils ne sont pas redimensionnables après leur déclaration. Pourquoi cet aspect est-il préjudiciable ? Reprenons l'exemple de l'exercice PJ1\_EX19 où toute la commande d'une table était stockée dans un tableau à deux dimensions. A priori, le nombre de consommations n'est pas connu à l'avance ; la seule issue est donc de réserver un tableau de très grande taille en espérant qu'on n'arrivera jamais en dépassement de sa capacité. Si on envisage un repas complet entrée, plat, dessert d'une société de 30 personnes, on peut arriver très facilement à 150 consommations (ou plus ?).

En déclarant un tableau de taille 150 pour toute commande, on gaspillera de la mémoire la plupart du temps tout en ayant aucune assurance de pouvoir contenir toute la commande d'une occasionnelle grande table. Au final, stocker la commande dans un tableau dont la taille est figée n'est certainement pas une solution idéale. L'idée serait de posséder une sorte de tableau dont la taille peut évoluer dynamiquement au cours de l'exécution. Les ArrayList offrent cette possibilité !

Un ArrayList est un tableau indexé de références à des objets de classe qui peut grandir ou rétrécir à la demande. Un ArrayList ne peut donc contenir une liste de nombres de type primitifs (`int`, `double`, ...) qui ne désignent pas des références mais des contenus (valeurs).

Le code suivant permet d'établir la liste du personnel qui travaillera ce jour à la brasserie. Le nombre de personnes diffère chaque jour c'est pourquoi un ArrayList est mieux indiqué qu'un tableau classique. Testez et interprétez le rôle des méthodes `.add()`, `.size()`, `.get()`

```
import java.util.ArrayList;

public class TestArrayList1 {

 public static void main(String[] args) {

 ArrayList<String> a = new ArrayList<>();

 a.add("Paul Lemaire");
 a.add("Stephane Dupont");
 a.add("Isabelle Deramaix");
 a.add("Emilie Maggi");

 for (int i=0;i<a.size();i++) {
 //prénom et nom
 System.out.println(a.get(i));
 //prénom
 System.out.println(a.get(i).substring(0,a.get(i).indexOf(" ")));}

 }
} //end main

} //end class
```

Il est aussi possible facilement d'insérer et décaler un élément dans la liste. Testez et interprétez le code ci-dessous en comparant avec le résultat de l'exécution précédente.

```
Public static void main(String[] args) {

 ArrayList<String> a = new ArrayList<>();

 a.add("Paul Lemaire");
 a.add("Stephane Dupont");
 a.add("Isabelle Deramaix");
 a.add(" Emilie Maggi");

 a.add(2, "Jean Deneyer") ;

 for (int i =0;i<a.size();i++) {
 System.out.println(a.get(i));
 }
}
```

La suppression d'un élément s'effectue par la méthode `remove(index)`.

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

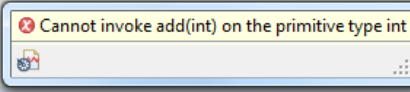
 ArrayList<String> a = new ArrayList<>();

 a.add("Paul Lemaire");
 a.add("Stephane Dupont");
 a.add("Isabelle Deramaix");
 a.add("Emilie Maggi");

 a.remove(2);
 for (int i=0;i<a.size();i++) {
 System.out.println(a.get(i));
 }
}
```

Les ArrayList ne peuvent pas contenir les types primitifs car les variables de type primitif désignent des valeurs et non des références. Le code suivant ne compile pas.

```
45 public void test() {
46
47 int a=1,b=2,c=3, d=4;
48 ArrayList at = new ArrayList ();
49 a.add(a);
50 a.add(b);
51 a.add(c);
52
53 }
```



#### 4.14.1. Autres possibilités de parcours et modifications des ArrayList

Outre la boucle for, il existe d'autres solutions pour parcourir et modifier les éléments d'un ArrayList. La boucle for... each permet de désigner une variable String comme référence de chaque élément de la liste. Le code suivant utilisant la boucle for ... each permet également d'afficher toute la liste de noms. Ajoutez le code ci-dessous au code précédent et constatez l'équivalence du résultat.

```
for (String strName : a) {
 System.out.println(strName);
}
```

Par contre, la boucle for each n'autorise pas l'écriture, c'est-à-dire qu'elle ne permet pas de modifier le contenu de la liste parcourue. Testez le code suivant qui est censé modifier chaque élément de la liste puis réafficher le contenu de celle-ci.

```

for (String strName : a) {
 strName="autre nom";
}
for (String strName : a) {
 System.out.println(s);
}

```

Pour modifier un élément de la liste, il faut utiliser la méthode `set(index, valeur)` et spécifier l'index de l'élément à modifier.

```

for (int i=0;i<a.size();i++) {
 a.set(i, "autre nom "+i);
}
for (String strName : a) {
 System.out.println(s);
}

```

Il est aussi possible d'utiliser un *itérateur* de liste pour parcourir l'ArrayList. Il ne faut alors pas gérer d'index `i` pour le parcourir.

```

ListIterator<String> it = a.listIterator();
while (it.hasNext()) {
 System.out.println(it.next());
}

```

La méthode `next()` renvoie la référence suivante contenue dans la liste et déplace l'index de parcours à la référence suivante. La première fois qu'on appelle `next()`, c'est bien la première référence de la liste qu'on désigne.

La méthode `hasNext()` renvoie `True` s'il y a encore une référence *après*, par rapport à la position de l'index courant. Elle est utilisée comme condition de poursuite de la boucle `while`. Pour la modification de l'élément référencé par l'itérateur, on utilise :

```

it.set("autre nom");

```

#### 4.14.2. Exercice

PJ1Ex24. Le but de cet exercice est de récupérer toute la commande d'une table dans un ArrayList. Le serveur a, sous forme de tableau imprimé à côté de l'ordinateur, la liste de tous les consommables avec leur N° identifiant. Il entre le N° de la consommation par son N° d'identifiant répondant à l'invitation "*Entrez le N° de consommable ou Q(Quitter)*".

Il précise ensuite la quantité de consommations pour ce type de consommation en répondant à l'invitation du programme "*Nombre de consommations pour X ? /A(Annuler) /Q (Quitter)*" X étant le nom de la consommation dont il vient de saisir le N° d'identifiant.

Après une consommation, le serveur a le droit de valider la commande et le message d'invitation devient : "*Entrez le N° de consommable ou Q(Quitter) V (Valider le ticket)*". Ces opérations d'encodage se succèdent jusqu'au moment où le serveur valide le ticket en entrant la lettre V (majuscule ou minuscule). Le but est de créer une procédure `getOrder(...)` dont la signature est la suivante :

```

public static void getOrder(ArrayList<int[]> ord) { }

```

et qui va modifier le contenu de l'ArrayList vierge passé en paramètre. Il s'agit d'un ArrayList de tableaux à une dimension et à 2 cases () contenant, dans la 1<sup>ère</sup> case, l'index de la consommation (à 1 unité près étant donné que la consommation N°1 porte l'index 0 dans les tables de prix et de consommation) et, dans la deuxième, le nombre de consommations.

```
ArrayList<int[]>
```

**ord**

|            |           |                    |   |
|------------|-----------|--------------------|---|
| 366712642  | → [1, 3]  | Spa Reine 25 Cl    | 3 |
| 257897541  | → [10, 2] | Jus D'orange Looza | 2 |
| 4878915841 | → [36, 2] | Spagh Bolo         | 2 |
| 9851478966 | → [37, 1] | Spagh Carbo        | 1 |

La commande est donc saisie dans un tableau dynamique (ArrayList) et non dans un tableau de taille figée comme c'était le cas pour l'exercice PJ1 EX19.

## 4.15. Création de fichier texte et écriture

Le ticket de caisse doit être simulé par la création d'un fichier texte. Voyons comment créer un fichier et y écrire du texte. On doit inévitablement recourir à un minimum de POO.

Le code ci-dessous donne un exemple de création de fichier texte et écriture dans celui-ci. Testez le code suivant, interprétez le rôle et le fonctionnement des méthodes de classes `File` et `FileWriter`. Entrez un chemin non valide pour comprendre le rôle de la structure `try {} catch {}` dans ce contexte.

```
import java.io.File;
import java.io.FileWriter;
import java.util.Scanner;

public class TestFichierTexte {

 public static void main(String[] args) {

 String strPathDirectory = "C:\\\\Test";

 try {
 Scanner sc = new Scanner(System.in);
 System.out.println("Entrez le chemin du fichier comme C:\\\\Temp par exemple");
 strPathDirectory = sc.nextLine();

 File f=new File(strPathDirectory + "\\MonFichierTexteIRAM_PS.txt"); //

 if (f.exists()) {
 System.out.println("Le fichier existe déjà et va être modifié");
 }else {
 System.out.println("Le fichier n'existe pas encore et va être créé");

 f.createNewFile();
 }

 FileWriter fw = new FileWriter(f, false);

 System.out.println("Entrez votre nom");

 fw.write("Ca y est ! Voilà que je sais écrire mon nom " +
 sc.nextLine() + " dans un fichier texte \r\n" +
 " Youhouuuuuu !!!!!\r\n");

 fw.close();
 System.out.println("Allez voir votre fichier " + f.getName() + " dans le " +
 " répertoire " + f.getParent());
 sc.close();

 }catch (Exception e) {

 System.out.println("Erreur survenue : " + e.getMessage());
 } //end catch
 } //end main
}
```



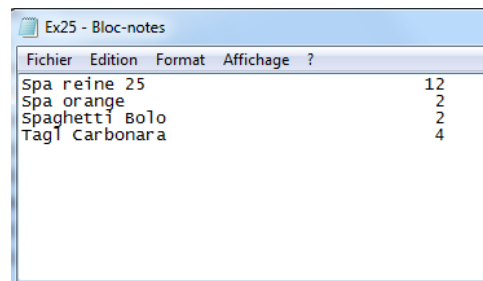
### 4.15.1. Pseudo code

Voici quelques exemples d'utilisation des fichiers en pseudo code.

```
Fichier : f
f.creer (Chemin)
f.ouvrir (Chemin)
Booleen : b ← f.existe(Chemin)
f.ecrire (str)
f.fermer ()
```

### 4.15.2. Exercice

PJ1 Ex1-25 Ce programme doit créer un fichier texte contenant le détail de l'ArrayList créé à l'exercice PJ1 Ex1-24 soit par exemple :



Les nombres de consommations sont alignés à droite et les unités sont au 30<sup>ème</sup> rang. Réutilisez avantageusement les fonctions déjà développées en les déclarant **public** et en les appelant de cette manière (dans la classe de l'exercice 25) :

```
Ex24.getOrder(order) ;
Ex23.placeNumberToRank (...) ;
```

## 4.16. Date et Heure

Ce chapitre nous permet de comprendre comment il est possible de gérer la date et l'heure dans un programme écrit en Java. Le moyen présenté ci-dessous recourt à la classe `Calendar` pour la récupération de la date et heure, la classe `SimpleDateFormat` pour présenter celle-ci au format désiré.

Pour la création du ticket de caisse, il sera nécessaire de récupérer la date et heure au format "year-month-day hour:minute second - millisecond" et l'utiliser pour nommer le fichier. Dans le contenu du ticket de caisse, il sera nécessaire d'y indiquer la date et heure mais cette fois au format "day/month/year hour:minute".

En Java le code minimum pour récupérer la date heure et l'afficher dans un format voulu est :

```
import java.text.SimpleDateFormat;
import java.util.Calendar;

public class TestCalendar1 {

 public static void main(String[] args) {

SimpleDateFormat Smp1DateFrmt = new SimpleDateFormat ("dd/MM/yyyy hh:mm:ss");
Calendar cldr = Calendar.getInstance(); //instanciation d'une classe Calendar
System.out.println("Actual Date & Time: " +
Smp1DateFrmt1.format(cldr1.getTime()));
 }
}
```

Testez et interprétez cet autre programme mettant en évidence d'autres types de format d'affichage.

```
import java.text.SimpleDateFormat;
import java.util.Calendar;

public class TestCalendar {

 public static void main(String[] args) {

int intHour; //récupération de l'heure du jour

//5 formats de date différents
SimpleDateFormat Smp1DateFrmt1 = new SimpleDateFormat ("dd/MM/yyyy hh:mm:ss");
SimpleDateFormat Smp1DateFrmt2 = new SimpleDateFormat ("dd/MM/yy HH:mm:ss");
SimpleDateFormat Smp1DateFrmt3 = new SimpleDateFormat ("dd/MM/yy hh:mm a");
SimpleDateFormat Smp1DateFrmt4 = new SimpleDateFormat ("dd/MM/yy HH:mm:ss.SSS ");
SimpleDateFormat Smp1DateFrmt5 = new SimpleDateFormat ("yyyy-MM-dd HH:mm:ss");
SimpleDateFormat Smp1DateFrmt6 = new SimpleDateFormat ("yyyy-MM-dd HH'h'mm:ss");

Calendar cldr1 = Calendar.getInstance(); //instanciation d'une classe Calendar
Calendar cldr2, cldr3 ;

System.out.println("Current Date & Time without format: " + cldr1.getTime());
System.out.println("Current Date & Time format 1 " + Smp1DateFrmt1.format(cldr1.getTime()));
System.out.println("Current Date & Time format 2: " + Smp1DateFrmt2.format(cldr1.getTime()));
System.out.println("Current Date & Time format 3: " + Smp1DateFrmt3.format(cldr1.getTime()));
System.out.println("Current Date & Time format 4: " + Smp1DateFrmt4.format(cldr1.getTime()));
System.out.println("Current Date & Time format 5: " + Smp1DateFrmt5.format(cldr1.getTime()));
System.out.println("Current Date & Time format 6: " + Smp1DateFrmt6.format(cldr1.getTime()));

System.out.println("Année : " + cldr1.get(Calendar.YEAR));
System.out.println("Année : " + cldr1.get(1));
System.out.println("Mois : " + cldr1.get(Calendar.MONTH));
System.out.println("Mois : " + cldr1.get(2));
System.out.println("Jour du mois : " + cldr1.get(Calendar.DAY_OF_MONTH));
System.out.println("Heure du jour : " + cldr1.get(Calendar.HOUR_OF_DAY));
System.out.println("Minute : " + cldr1.get(Calendar.MINUTE));
System.out.println("Seconde : " + cldr1.get(Calendar.SECOND));
```

```

System.out.println("Milliseconde : " + cldr1.get(Calendar.MILLISECOND));

intHour = cldr1.get(Calendar.HOUR_OF_DAY);
System.out.println("On a récupéré l'heure du jour dans une variable : " + intHour);

cldr2 = cldr1; //copie de cldr1 dans cldr2 avant le add
cldr3 = (Calendar)cldr1.clone(); //copie de cldr1 dans cldr3 avant le add

cldr1.add(Calendar.HOUR_OF_DAY, 3);
System.out.println("\nIn 3 hours Date & Time will be : " + Smp1DateFrmt2.format(cldr1.getTime()));

//illustration de la différence entre cldr2 = cldr1 et cldr3 = (Calendar) cldr1.clone()
System.out.println("Calendrier 2 : Current Date & Time format 2 : " +
Smp1DateFrmt2.format(cldr2.getTime()) + " Oupssssss cldr2 impacté par le add de cldr1 !!!!!");
System.out.println("Calendrier 3 Current Date & Time format 2 : " +
Smp1DateFrmt2.format(cldr3.getTime()) + " cldr3 pas impacté par le add de cldr1 !");

}
}

```

#### 4.16.1.Pseudo Code

Date : maDateHeureActuelle  $\leftarrow$  getTime()

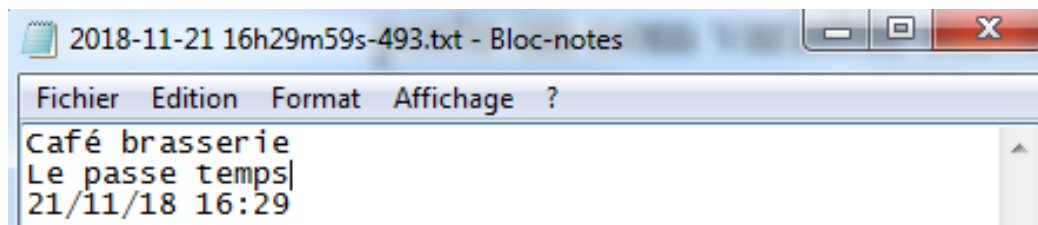
Chaine : str  $\leftarrow$  format(maDateHeureActuelle, "dd/MM/yyyy hh:mm:ss")

#### 4.16.2. Exercice

Réalisez l'exercice PJ1 Ex1-26

PJ1 Ex1-26 Le programme crée un fichier texte dans un répertoire choisi par l'utilisateur répondant à l'invitation « *Entrez le chemin du répertoire de destination* ». Dans le cas où le chemin n'est pas valide, le message d'erreur « *Chemin non valide* » survient, et le programme repose ensuite la question. Le fichier créé porte un nom variable suivant la date et l'heure de l'exécution du programme. Le nom du fichier doit être au format suivant : 2018-11-21 16h29m56s-493.txt

Le contenu du fichier est celui-ci (avec la date heure au moment de la création) :



En fin de programme, on affiche le message « *Fichier créé avec succès!* ».

Créer les fonctions

-*getActualFormattedDateTime(...)* qui renvoie la date heure actuelle dans un format choisi et passé en paramètre.

-*getUserPathDir(...)* qui renvoie un chemin valide saisi par l'utilisateur

-*fileCreation(file)* qui crée le fichier passé en paramètre et qui renvoie true si la création a réussi.

## 4.17. Variables globales et variables locales

Voyons comment définir, différencier et utiliser les variables locales et globales en testant l'exercice suivant :

```
import java.util.Scanner;

public class TestPorteeVariables {

 static int intNombreA = 0;

 public static void main(String[] args) {

 int intNombreB = 5;

 Scanner sc = new Scanner(System.in);
 System.out.println("Entrez un nombre entier");
 intNombreA = sc.nextInt();

 sc.close();
 MaProcedure();
 intNombreA = intNombreA + intNombreB;
 MaProcedure();

 } //fin main

 public static void MaProcedure() {
 System.out.println("Valeur de intNombreA : " + intNombreA);

 //System.out.println("Valeur de intNombreB : " + intNombreB);

 } //fin MaProcedure
} //fin class
```

### 4.17.1.1. Variable globale

Remarquez l'endroit où a été déclaré la variable `int intNombreA` : elle a été déclarée en dehors du `main()` juste en dessous de `public class TestPorteeVariables()`.

Déclarer une variable à cet endroit, (en dehors d'une fonction ou d'une procédure) rend la variable utilisable par toutes les procédures (`main()` et `MaProcédure()`).

`intNombreA` désigne donc un emplacement mémoire accessible par tous les sous programmes on parle alors de variable globale.

## Variable globale

Lorsqu'une variable est déclarée directement dans la classe, c'est-à-dire à l'extérieur de toute fonction ou procédure (*méthode*), on a alors une zone mémoire qui sera accessible en lecture écriture dans toute la classe, par toutes les procédures, fonctions déclarées dans cette classe. On parle alors de variable globale (*champ* de la classe (*fields*) en POO).

Remarque : En java on ajoutera systématiquement le mot `static` dans la déclaration. On ne pourra l'expliquer qu'en POO.

### 4.17.1.2. Variable locale

Penchons-nous maintenant sur la variable `int intNombreB` qui a été déclarée à l'intérieur du `main()`. Enlevez le double `//` de l'instruction suivante dans la procédure `MaProcédure()` :  
`//System.out.println("Valeur de intNombreB : " + intNombreB );`  
 pour tenter d'afficher le contenu de la variable `intNombreB` à l'intérieur de la procédure `MaProcédure()`.

Constatez le comportement du compilateur et interprétez-le.

## Variable locale

Lorsqu'une variable est déclarée à l'intérieur d'une fonction ou d'une procédure (*méthode en POO*), on a alors une zone mémoire qui sera accessible et utilisable uniquement par cette fonction ou procédure. Son emplacement mémoire sera réservé seulement pendant la durée de vie de la fonction ou de la procédure.

On pourrait être tenté de se dire « *qui peut le plus peut le moins* » et de déclarer toutes les variables en mode global. Elles seraient ainsi accessibles tout le temps et par tous les sous-programmes sans devoir se soucier de quelle procédure utilise quelle variable.

**Cette façon de procéder est à bannir absolument !** Premièrement parce qu'elle déstructure le programme en allant à l'encontre de la programmation modulaire, deuxièmement car la mémoire est mal gérée. En effet, un avantage des fonctions et procédures est de pouvoir les utiliser et réutiliser dans l'un ou l'autre programme. Si dans le code de ces fonctions, on utilise des variables globales alors ces fonctions ou procédures ne pourront être réutilisées telles quelles dans d'autres projets. Aussi, il faut garder à l'esprit que toutes les variables globales prennent de l'espace mémoire et consomment des ressources tant que la classe est ouverte.

## 4.18. Passage de paramètres par adresse ou valeur

Commençons par dire que Java est un mauvais exemple pour expliquer la différence entre un passage de paramètres par valeur ou par adresse étant donné que Java ne nous laisse pas le choix entre l'un et l'autre contrairement à d'autres langages ou même son ancêtre le C qui intégrait la notion de pointeur.

Le **passage de paramètres par valeur** indique que le programme appelant passe le contenu mémoire de sa variable à la variable locale de la routine. Il y a donc deux zones mémoires et le contenu de la mémoire du programme appelant est simplement copié dans la zone mémoire de la

routine. Les deux zones mémoires bien distinctes de sorte qu'une modification éventuelle de la variable en paramètre à l'intérieur de la routine ne sera pas répercutée sur la variable du programme appelant.

A l'inverse **un passage de paramètre par adresse (référence)** indique que le programme appelant passe l'adresse mémoire de la variable et non son contenu. Le programme appelant et la routine appelée utilisent alors la même zone mémoire. Une modification de la valeur dans la variable déclarée dans la routine affectera également la valeur de la variable déclarée dans le programme appelant.

Testez le programme Java suivant qui permet de déterminer si Java travaille en passage de paramètre par valeur ou par adresse pour le type `int`, `Integer`, `String`. Quel est le type de passage pour chacun d'eux ?

```
import java.util.Scanner;

public class TestPassageParametres {

 public static void main(String[] args) {

 int intNombreA = 5;
 Integer intNombreB = new Integer(5);
 String strChaine = "Bonjour";

 System.out.println("Valeur de intNombreA avant exécution de la procédure : " +
intNombreA);
 AjouterDixAvecParametre_TypeInteger(intNombreA);
 System.out.println("Valeur de intNombreA après exécution de la procédure : " +
intNombreA);

 System.out.println("\nValeur de intNombreB avant exécution de la procédure : " +
intNombreB);
 AjouterDixAvecParametre_TypeInteger(intNombreB);
 System.out.println("Valeur de intNombreB après exécution de la procédure : " +
intNombreB);

 System.out.println("\nValeur de strChaine avant exécution de ModifieString : " +
strChaine);
 ModifieString(strChaine);
 System.out.println("Valeur de strChaine après exécution de ModifieString : " +
strChaine);

 } //fin main

 public static void AjouterDixAvecParametre_Typeint(int intNombre){

 intNombre = intNombre +10;
 //System.out.println("Valeur de intNombre : " + intNombreB);

 } //fin AjouterDixAvecParametre_Typeint

 public static void AjouterDixAvecParametre_TypeInteger(Integer intNombre){
 intNombre = intNombre + 10;
 System.out.println("Valeur de intNombre à l'intérieur de
AjouterDixAvecParametre_TypeInteger: " + intNombre);
 } //fin AjouterDixAvecParametre_TypeInteger

 public static void ModifieString(String str){
 str= "Aurevoir";
 System.out.println("Valeur de str à l'intérieur de ModifieString: " + str);
 } //fin ModifieString
} //fin class
```

L'exemple suivant illustre les deux modes de passage de paramètres dans un autre langage comme le C#.

```
C#
class Program
{
 static void Main(string[] args)
 {
 int arg;

 // Passing by value.
 // The value of arg in Main is not changed.
 arg = 4;
 squareVal(arg);
 Console.WriteLine(arg);
 // Output: 4

 // Passing by reference.
 // The value of arg in Main is changed.
 arg = 4;
 squareRef(ref arg);
 Console.WriteLine(arg);
 // Output: 16
 }

 static void squareVal(int valParameter)
 {
 valParameter *= valParameter;
 }

 // Passing by reference
 static void squareRef(ref int refParameter)
 {
 refParameter *= refParameter;
 }
}
```

L'exemple suivant illustre les deux modes de passage de paramètres dans le VBA d'Excel :

```
Option Explicit
Private Sub CommandButton1_Click()
Dim intNombre As Integer

intNombre = 5

Call Ajouter10ParValeur(intNombre)
MsgBox "intNombre après passage dans la procédure Ajouter10ParValeur : " &
intNombre

intNombre = 5
Call Ajouter10ParAdresse(intNombre)
MsgBox "intNombre après passage dans la procédure Ajouter10ParAdresse : " &
intNombre

End Sub

Public Sub Ajouter10ParValeur(ByVal Nombre As Integer)
 Nombre = Nombre + 10
End Sub
Public Sub Ajouter10ParAdresse(ByRef Nombre As Integer)
 Nombre = Nombre + 10
End Sub
```

#### 4.19. Exercice récapitulatif – création du ticket de caisse.

Nous disposons maintenant de tous les outils pour pouvoir réaliser l'exercice d'impression du ticket de caisse. Voici le cahier des charges initial :

PJ1 Ex1-27 A partir d'une interface console, le serveur encode d'abord le N° de table entre 1 et 20, répondant à l'invitation "Entrez le numéro de table (de 1 à 20)/ Q (Quitter)". Si le N° est inférieur à 1 ou ne correspond pas à un entier, le programme affiche « saisie incorrecte », si le N° est supérieur à 20, le programme affiche « N° de table maximum 20, contactez votre développeur pour augmenter ce nombre ».

Il entre ensuite le N° de la consommation par son N° d'identifiant répondant à l'invitation "Entrez le N° de consommable ou Q(Quitter)". Il a sous les yeux, sous forme d'un tableau imprimé, la liste de tous les consommables avec leur N° identifiant.

Il précise ensuite la quantité de consommations pour ce type de consommation en répondant à l'invitation du programme "Nombre de consommations pour XXX ? /A(Annuler) Q (Quitter)" XXX étant le nom de la consommation dont il vient de saisir le N° d'identifiant.

Le message d'invitation devient ensuite "Entrez le N° de consommable ou Q(Quitter) V (Valider le ticket)". Ces opérations d'encodage se succèdent jusqu'au moment où le serveur valide le ticket en entrant la lettre V (majuscule ou minuscule). Pour simuler la sortie du ticket, on enregistre un nouveau fichier texte sur le disque dur. Le nom de ce fichier est la date et l'heure du moment où le fichier est créé. Le format est de type 2018-09-18 13h41m15s-380.txt de façon à garantir qu'il n'y aura jamais de doublon même si un autre serveur encode un autre ticket quasi au même moment. Le contenu du ticket est tel qu'affiché ci-dessous

| Fichier Edition Format Affichage ? |    |       |        |
|------------------------------------|----|-------|--------|
| Café-Brasserie                     |    |       |        |
| Le passe temps                     |    |       |        |
| 26/11/18 17:47                     |    |       |        |
| Table:2                            |    |       |        |
| Spa reine 25                       | 2  | 2,20  | 4,40   |
| Pepsi                              | 2  | 2,20  | 4,40   |
| Spaghetti Bolo                     | 2  | 10,80 | 21,60  |
| Tag1 saum                          | 4  | 16,90 | 67,60  |
| Jus d'orange Looza                 | 1  | 2,60  | 2,60   |
| -----                              |    |       |        |
| TOTAL CONSUMMATIONS                | 11 |       |        |
| -----                              |    |       |        |
| TOTAL TVA 21,00%                   |    | 1,98  | 11,40  |
| TOTAL TVA 12,00%                   |    | 9,56  | 89,20  |
| TOTAL TVA 6,00%                    |    | 0,00  | 0,00   |
| -----                              |    |       |        |
| TOTAL                              |    |       | 100,60 |

Le programme se réfère, pour les N° de consommations, dénominations, prix TVAC et TVA à des tableaux internes au programme (voir les tableaux à la fin de l'énoncé)

- L'appui sur Q(Quitter) à n'importe quel message d'invitation arrête le programme sans enregistrement du ticket.
- Il ne doit pas être possible d'entrer un N° de table plus grand que 20. Si le N° de table entré est supérieur à 20, le programme indique "contactez votre développeur pour augmenter le nombre de tables".
- Après la première consommation entrée, il est possible de valider le ticket ; le message d'invitation devient alors "Entrez le N° de consommable. V (Valider), Q(Quitter)".

Pour l'amélioration de l'affichage :



