Programmation Java 1A - TP1
Environnement de développement
Première application en Java

Exercice 1. Présentation

1. Afin d'organiser proprement vos fichiers, créez dans votre compte (par exemple dans le répertoire *prive*) un répertoire nommé ***java***, dans lequel vous allez créer le répertoire ***tp1***.

2. A l'aide de l'éditeur de texte *nedit*, tapez le programme Java suivant permettant d'écrire "Bonjour tout le monde" à l'écran. Sous quel nom devez-vous le sauvegarder ?

```
/* Bonjour.java : ce programme affiche un message de bienvenue */

class Bonjour {

    public static void main(String[] args) {

        System.out.println("Bonjour");

    }
}
```

Remarques :

- Vous pouvez lancer *nedit* en tapant ***nedit &*** dans le terminal de façon à avoir toujours en permanence à l'écran la fenêtre *nedit* pour **taper le code** et la fenêtre *xterm* pour **compiler et exécuter** votre programme.
- Vérifiez bien que vous avez configuré *nedit* pour qu'il affiche en couleur la syntaxe (dès que vous aurez donné un nom à votre fichier, *nedit* saura qu'il est écrit en java), pour qu'il ne passe pas à la ligne automatiquement, et pour qu'il vous affiche les numéros de ligne.
- Sauvez ***très régulièrement*** votre programme : cela vous évitera de le perdre en cas de mauvaise manipulation ou de plantage du système.

3. Regardez les fichiers présents dans le répertoire courant (avec la commande *ls -l*). Pour compiler votre premier programme, tapez sur la ligne de commande (dans le *terminal*) la commande : *javac Bonjour.java*
Regardez à nouveau les fichiers dans votre répertoire courant. Le fichier *Bonjour.class* a été créé par le compilateur Java, et c'est lui que l'on peut exécuter avec l'interpréteur Java.

4. Exécutez votre premier programme Java en tapant sur la ligne de commande : *java Bonjour*

Exercice 2. println et print

1. Recopiez le source de votre premier programme dans un autre fichier nommé *Bonjour2.java* (commande *cp*).

2. Modifiez le fichier *Bonjour2.java* pour utiliser la méthode *print* à la place de *println :* remplacez la ligne contenant la méthode *println* par les deux lignes suivantes :

```
System.out.println("Bonjour ");
System.out.println("tout le monde");
```

Que se passe-t-il lorsque vous exécutez le programme *Bonjour2* ?
Remplacez maintenant sur la première ligne *println* par *print*. Que se passe-t-il alors ?

Exercice 3. Corriger le programme suivant

1. Copiez le programme suivant dans un nouveau fichier (sauvez-le souvent, par exemple en utilisant le raccourci clavier *Ctrl-S*) :

```
class ArithmeticTest {
// Classe qui permet de tester les opérations arithmétiques
    public static void main (String args[]) {
        // déclarations de deux entiers et de deux réels
        int x
        int y
        double a
        double b
        // initialisation des quatre variables
        x = 12;
        y = 5;
        a = 12.5;
        b = 7;
        // affichage des deux entiers et de leur somme
        System.out.println("x vaut " + x + ", y vaut " + y);
        System.out.println("x + y = " + x + y);
        // affichage des deux réels et du résultat de leur division
        System.out.println("a vaut " + a + ", b vaut " + b);
        System.out.println("a / b = " + (a / b));
    }
}
```

2. Essayez de compiler le programme.
Regardez bien comment le compilateur vous indique les erreurs de compilation, puis corrigez-les et exécutez le programme.

3. Corrigez le programme de façon à voir le résultat correct pour x+y.

4. Complétez le programme pour qu'il affiche en plus de l'addition de deux entiers le résultat des quatre autres opérations usuelles sur les entiers : -, *, /, et %.

Exercice 4. Séquences d'échappement

1. Recopiez le fichier *Bonjour2.java* dans un nouveau fichier nommé *Bonjour3.java*

2. Modifiez *Bonjour3* pour qu'il affiche à l'écran : *"Bonjour tout le monde"* (avec les guillemets).

3. Pour afficher des guillemets avec un *println* (ou *print*), on est obligé d'utiliser ce qu'on appelle une séquence d'échappement ; en effet, si on écrit *System.out.println(""Bonjour"")* pour que *"Bonjour"* s'affiche, ça ne marche pas : le compilateur croit qu'on ouvre les guillemets du *println* et qu'on les referme tout de suite... Une séquence d'échappement est composée du caractère *backslash* (\) suivi d'un autre caractère, et sert à afficher des caractères particuliers.

Du coup, pour afficher un \ à l'écran, on ne peut pas se contenter d'écrire *System.out.println("\")* puisque c'est le caractère réservé pour les séquences d'échappement !

Voici les principales séquences d'échappement dont vous pouvez vous servir : **\n** pour passer à la ligne, **\t** pour placer une tabulation, **\b** pour effacer le caractère précédent, \" pour afficher des guillemets, \\ pour afficher un antislash.

Modifiez le source de *Bonjour3.java* pour utiliser ces différentes séquences et voir leur fonctionnement.

Programmation Java 1A - TP2
Instructions conditionnelles - Lecture au clavier

Pour faire les exercices 1, 3 et 4 de ce TP, créez un dossier *tp2* dans le dossier *Java* situé sur votre compte. Vous pourrez créer également dans le dossier *tp2* des sous-dossiers différents pour chacun des exercices.

Exercice 1. Valeur absolue d'un nombre

Ecrivez un programme appelé *ValeurAbsolue* qui calcule la valeur absolue d'un nombre. N'oubliez pas de le commenter.
Vous aurez besoin de **déclarer** et d'**initialiser** une variable de type double dans la méthode *main*.

Exercice 2. Lecture au clavier

Pour exécuter le programme précédent avec différentes valeurs, on est obligé de modifier le code du programme, puis de le recompiler et de l'exécuter à nouveau.
Pour vous permettre de saisir des valeurs au clavier plutôt que de les initialiser dans le code, vous allez récupérer et installer une classe Java : la classe *Console*. Vous avez dû recevoir cette classe par courrier électronique : sauvez le fichier *Console.class* dans votre répertoire *java*.

Cette classe ne fait pas partie de l'environnement Java de base, il va falloir indiquer au compilateur l'endroit où se trouve ce fichier sur le disque dur.
Si vous avez déjà dans la racine de votre compte un fichier nommé *.bashrc*, ouvrez le et ajoutez-y la ligne suivante (pour voir ce fichier, il faut utiliser la commande **ls -la** car c'est un fichier caché comme tous ceux dont le nom commence par un point) ; sinon créez ce fichier dans la racine de votre compte et placez-y la ligne suivante (en remplaçant ***login*** par votre login UNIX, sans mettre aucun espace, et sans oublier le point à la fin) :

export CLASSPATH=~***login***/prive/java:.

Utilisez le bon chemin pour votre répertoire *java* : utilisez exactement les noms des répertoires que vous avez créés dans votre compte.

Le fichier *.bashrc* est exécuté à chaque fois que vous ouvrez un terminal et la ligne ci-dessus règle la variable d'environnement *CLASSPATH* dont se sert le compilateur Java pour trouver les classes qui ne sont pas fournies en standard.

Une fois que vous avez sauvé le fichier *.bashrc*, fermez tous vos terminaux et ouvrez-en un nouveau (qui va donc lire le fichier *.bashrc*).
Vérifiez bien que vous n'avez pas d'erreur en ouvrant ce terminal (sinon, corrigez l'erreur dans le fichier *.bashrc*).
Tapez la commande *env* pour vérifier que la variable *CLASSPATH* existe et a la bonne valeur.

La classe *Console* vous fournit trois méthodes utiles : *readInt(String)* pour lire un entier au clavier, *readDouble(String)* pour lire un nombre réel au clavier, et *readLine(String)* pour lire du texte au clavier.
Ces méthodes vérifient que la valeur entrée est correcte et demandent une autre valeur si ce n'est pas le cas.

La syntaxe de ces méthodes est la suivante : si on a déclaré une variable *n* de type *int*, on peut lire sa valeur au clavier en écrivant : *n=Console.readInt("Entrer un entier :");*
La chaîne de caractères entre guillements dans les parenthèses s'affiche pour demander à l'utilisateur qu'on attend qu'il entre une valeur.
Modifier le programme de l'exercice 1 pour qu'il demande à l'utilisateur de saisir une valeur au clavier et affiche sa valeur absolue.

Exercice 3. Cercle

Ecrivez un programme appelé *Cercle* dans lequel vous **déclarez** et **initialisez** une constante *PI* égale à *3.14* et deux variables réelles *circonference* et *aire*. Ce programme doit saisir au clavier la valeur du rayon *r* d'un cercle, puis calculer et afficher la circonférence (2**PI***r*) et l'aire (*PI***r***r*) de ce cercle.
Remarque : pour déclarer une constante au lieu d'une variable, on utilise la syntaxe (à placer en dehors de la méthode *main*) : **final static double PI=3.14;**

Exercice 4. Signe d'un produit

Ecrivez un programme appelé *Signe* dans lequel vous **déclarez** et **initialisez** deux variables entières *a* et *b*, et qui affiche le signe du produit de *a* et *b* sans faire la multiplication. Utilisez la méthode *readInt()* pour permettre la saisie au clavier de ces deux entiers.
Dessinez ci-dessous ce qui se passe en mémoire lorsque votre programme s'exécute.

Exercice 5. (facultatif)

Reprenez les algorithmes que vous avez vu au TD numéro 1 d'Algorithmique et écrivez les programmes correspondant pour vérifier que vos algorithmes fonctionnent.

Programmation Java 1A - TP3
Instructions conditionnelles - Lecture au clavier (suite)

Pour faire les exercices de ce TP, créez un dossier *tp3* dans le dossier *Java* situé sur votre compte.
Vous pouvez créer également dans le dossier *tp3* des sous-dossiers différents pour chacun des exercices.

Exercice 1. Classer trois nombres

On veut classer par ordre croissant trois nombres réels saisis au clavier.
Commencez par **écrire l'algorithme sur papier** en utilisant des instructions conditionnelles.
Ecrivez ensuite le programme correspondant, qui saisit trois nombres de type *double* au clavier et affiche ensuite ces trois nombres par ordre croissant.

Exercice 2. Menu

Ecrivez un programme qui affiche à l'écran le menu suivant, puis saisit un entier au clavier correspondant au choix de l'utilisateur.
Si l'utilisateur saisit 0, le programme affichera "Au revoir..." ; si il saisit un entier entre 1 et 3, le programme affichera "Votre ... est en cours de préparation" où les points de suspension sont remplacés par la boisson sélectionnée ; si il saisit une autre valeur, le programme affichera "Mauvais choix".

1. Café
2. Lait
3. Thé
0. Quitter

Exercice 3. switch

Reprenez l'exercice précédent en remplaçant les instructions *if* par l'instruction *switch* dont la syntaxe est la suivante :

```
switch (variable) {
      case valeur1 :
            instruction;
            break;
      case valeur2 :
            instruction;
            break;
      ...
      default :
            instruction;
}
```

Cette instruction aiguille le programme sur l'une ou l'autre des instructions suivant la valeur de la variable. Si la variable ne correspond à aucune des valeurs, l'instruction par défaut est exécutée.

Exercice 4. Caractères

Modifiez les deux programmes précédents pour afficher plutôt le menu suivant :

C. Café
L. Lait
T. Thé
Q. Quitter

Vous pourrez faire en sorte que le programme fonctionne même si on entre un caractère minuscule au clavier.

Exercice 5. (facultatif)

Reprenez l'exercice "La Poste" (affranchissement de lettres en fonciton de leur poids et du type de service désiré) que vous avez vu en TD d'Algorithmique et écrivez le programme correspondant.

Programmation Java 1A - TP4
Boucles

Pour faire les exercices de ce TP, créez un dossier *tp4* dans le dossier *Java* situé sur votre compte.

Exercice 1. Décompte

1. Compilez et corrigez le programme suivant :

```
public class Decroissant {
// affiche Au revoir lorsqu'on entre 0 au clavier
    public static void main(String[] args) {
        int nombre;
        nombre=Console.readInt("Entrez un nombre entier : "); // lecture au clavier
        if (nombre = 0) // si on a saisi 0
            System.out.println("Au revoir...");
    }
}
```

2. Modifiez ce programme (et le commentaire de la première ligne) en utilisant une boucle *while* de façon à ce que lorsqu'on rentre un nombre négatif, le programme affiche *Au revoir...* et lorsqu'on entre un nombre positif, le programme affiche à l'écran tous les entiers par ordre décroissant depuis le nombre entré jusqu'à 0 puis affiche *Au revoir....* Ecrivez votre programme de la façon la plus simple possible (par exemple dans cette question, n'utilisez pas de *if* si ce n'est pas nécessaire).

Exercice 2. Affichage avec tabulations

En utilisant les séquences d'échappement et la boucle **while**, écrivez un programme qui saisit un entier *n* au clavier et affiche à l'écran la table de multiplication par *n-1*,*n*, et *n+1*.Par exemple si on saisit 7, on doit obtenir :

Table de multiplication :

	par 6	par 7	par 8
1	6	7	8
2	12	14	16
3	18	21	24
4	24	28	32

etc.
(on affichera par exemple les vingt premières lignes de la table).

Exercice 3. Menu (suite)

Recopiez le programme *Menu.java* du TP précédent (celui avec les caractères) dans le répertoire *tp4* et modifiez-le de façon à ce que lorsqu'on saisit autre chose que C, L, T ou Q le programme affiche à nouveau le menu et redemande d'entrer une valeur au clavier. Quel type de

boucle allez-vous utiliser ?

Plutôt que d'afficher un message pour chacun des choix ce qui oblige à répéter plusieurs fois la commande *System.out.println*, modifiez le programme en déclarant une variable de type *String* (texte), à laquelle vous donnerez une valeur différente pour chacun des cas, et affichez cette variable seulement à la fin du programme.

Exercice 4. Factorielle

a. Ecrivez un programme utilisant la boucle *while* et la boucle *do...while* qui permet de saisir un nombre **positif** et **plus petit que 10** au clavier et affiche ensuite sa factorielle (on rappelle que la factorielle de l'entier n se note n! et vaut n*(n-1)*(n-2)*...*2*1 et que la factorielle de 0 vaut 1.
On souhaite un affichage du type : **4!=24** lorsqu'on a saisi 4.

b. Ecrivez une seconde version de ce programme dans laquelle vous remplacez la boucle *while* par une boucle *for*.

Programmation Java 1A - TP5
Boucles (suite)

Exercice 1. Boucle simple

Ecrivez un programme qui affiche les *n* premiers entiers positifs, *n* étant saisi au clavier. L'affichage se fera sous la forme suivante où chaque entier est séparé du précédent par une tabulation et où chaque ligne ne contient que 10 entiers au maximum :

```
1   2   3   4   5   6   7   8   9   10
11  12  13  14   ...
```

Exercice 2. Table de multiplication

Ecrivez un programme qui affiche une table de multiplication selon le format ci-dessous (bien sûr, on utilisera des boucles...).

	par 1	**par 2**	**par 3**	**par 4**	**par 5**	**par 6**	**par 7**	**par 8**	**par 9**	**par 10**
1	**1**	**2**	**3**	**4**	**...**	**...**	**...**	**...**	**...**	**...**
2	**2**	**4**	**6**	**8**	**...**	**...**	**...**	**...**	**...**	**...**
3	**3**	**6**	**9**	**12**	**...**	**...**	**...**	**...**	**...**	**...**
4	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**
5	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**
6	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**
7	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**
8	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**
9	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**
10	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**	**...**

Exercice 3. Sous-programmes

On souhaite afficher une table de conversion de degrés Farenheit en degrés Celsius.
La formule de conversion est : *degréCelsius=(degréFarenheit-32)*5/9*.

Remarque : pour les questions qui suivent, on n'utilisera que des variables entières, et une division entière (ce qui permet d'avoir des valeurs arrondies dans la table). Tous les sous-programmes seront placés dans la classe*Conversion*.

1. Ecrire un sous-programme *afficheFenC* qui prend en argument un entier représentant des degrés Farenheit et affiche sa conversion en degrés Celsius.

2. Ecrire un sous-programme *afficheTableFenC* qui prend en argument deux valeurs *temperatureA* et *temperatureB* exprimées en degrés Farenheit et affiche une table de conversion pour les valeurs Farenheit comprises entre ces deux valeurs par pas de 20. Par exemple, voici la table obtenue en entrant -20 et 20 comme deux valeurs extrêmes.

Farenheit	Celsius
-20	-28
0	-17
20	-6

Programmation Java 1A - TP6

Procédures et fonctions

Exercice 1. Programme bizarre

Vous avez reçu par e-mail le programme *Bidon.java*. Sauvegardez-le.

1. Ouvrez ce programme et essayez de comprendre à quoi il sert. Renommez alors la classe et la procédure.

2. Ecrivez un jeu de test, et testez le programme.

3. Dans quels cas ce programme n'est-il pas correct ? Proposez une correction.

Exercice 2. Fonctions simples

Pour cet exercice, vous écrirez toutes les fonctions demandées dans une même classe *FonctionsSimples* et écrirez une méthode *main()* qui permet de les tester l'une après l'autre.

Faites très attention aux déclarations de variables :
- chaque variable que vous utilisez doit être déclarée dans la fonction où vous l'utilisez.
- aucune variable ne doit être déclarée en dehors d'une fonction.
- on peut utiliser un argument d'une fonction dans la fonction sans le redéclarer ni l'initialiser

(l'initialisation se fait avec la valeur qu'on place entre parenthèses lors de l'appel de la fonction)

1. Ecrire une fonction *commission()* qui prend en argument un nombre réel représentant le montant en euros d'une transaction et renvoie le montant de la commission : si le montant est supérieur à 1000 €, la commission est de 10% du montant, sinon elle est de 5% avec dans tous les cas un minimum de 20€.
Exemple : commission(1000)=100 ; commission(1500)=150 ; commission(100)=20.

2. Ecrire une fonction *cumul()* qui prend en argument deux entiers *i* et *j*, et renvoie la somme des entiers compris entre *i* et *j* (si *i>j*, la fonction retourne -1).
Exemple : cumul(0,3) = 6 ; cumul(4,5)=9 ; cumul(12,-4)=-1.

3. Ecrire une fonction *valeurEuro()* qui prend en argument un nombre réel représentant un montant en francs et renvoie le montant correspondant en euros. On rappelle que 1 €=6,55957 F et qu'on arrondit les montants convertis à deux chiffres après la virgule.
Remarque : pour obtenir un montant à deux chiffres après la virgule, on peut utiliser un montant 100 fois plus grand, prendre sa partie entière, puis rediviser par 100 ; pour prendre la partie entière d'une variable réelle, on place (int) devant :
exemple : **double** val_francs=10.5 ; **int** val_euros_fois_100=(int)(100.0*val_francs/6.55957) ; **double** val_euros=val_euros_fois_100/100.0).

4. Ecrire une fonction booléenne estMultiple( ) qui prend en argument deux entiers *i* et *j*, et retourne vrai si *i* est un mulple de *j*.

Programmation Java 1A - TP7
Tableaux

Exercice. Parcours de tableau

Le but de cet exercice est d'écrire une classe *ParcoursTableau* permettant de saisir un tabeau d'entiers, puis de le parcourir "à l'endroit" ou "à l'envers".

1. Ecrire une procédure *parcours* qui prend en paramètre un tableau d'entiers et un booléen *gaucheAdroite* ; ce booléen vaut vrai pour parcourir le tableau "de gauche à droite" et faux pour le parcourir "de droite à gauche". Cette procédure affiche à l'écran le contenu du tableau.

2. Ecrire une méthode *main* dans laquelle un tableau est déclaré, créé (on demandera à l'utilisateur de saisir la longueur du tableau) et remplis (valeurs saisies par l'utilisateur).
Le contenu de ce tableau est ensuite affiché, dans une sens puis dans l'autre, en faisant appel à la procédure *parcours*.

Dessiner ci-dessous l'état de la mémoire :
1. Après la déclaration des variables de la méthode *main* ;
2. Après la création du tableau ;
3. Après le remplissage du tableau ;
4. En cours d'affichage du contenu du tableau par la procédure *parcours*.

3. Ecrire une procédure *saisieTableau* qui prend un tableau d'entiers en paramètre, et saisit son contenu au clavier. Cette procédure suppose que le tableau a déjà été créé (sa longueur est donc connue. Dans la procédure, vous pouvez utiliser*nomTableau.length* pour obtenir cette longueur).
Rédigez soigneusement la spécification de cette procédure.

4. Ecrire une méthode main (afin de conserver la trace du travail fait précédemment, la méthode *main* de la question 2 pourra être placée en commentaire) permettant d'afficher et de traiter le menu suivant :

1. Saisir le tableau.
2. Afficher le contenu de gauche à droite.
3. Afficher le contenu de droite à gauche.
0. Quitter.

Lorsque l'utilisateur effectue le choix 1, on lui demande tout d'abord combien d'entiers il veut saisir (c'est-à-dire : la longueur du tableau), on crée le tableau puis on appelle la procédure *saisieTableau*.
Pour les choix 2 et 3, si le tableau n'a pas été rempli auparavant, un message d'avertissement demandant la saisie du tableau doit être affiché.
Ce menu est réaffiché après chaque opération jusqu'à ce que l'utilisateur fasse le choix 0.

Dessiner ci-dessous l'état de la mémoire pendant l'exécution de la procédure *saisieTableau*.

4. Ecrire une fonction *saisieTableau* qui n'a aucun paramètre et retourne un tableau d'entiers, de façon à ce que cette fois-ci la longueur du tableau soit demandée dans la fonction *saisieTableau* (et non plus dans le *main*).

Dessiner ci-dessous l'état de la mémoire pendant l'exécution de cette fonction *saisieTableau*.

Programmation Java 1A - TP8

Fonctions - Tableaux (suite)

Exercice 1. Minimum et maximum d'un tableau

1. Ecrire une procédure *miniMaxi* qui prend en paramètre un tableau d'entiers et affiche à l'écran la plus grande et la plus petite valeur de ce tableau.

2. Mettre en place un jeu de test pour la procédure *miniMaxi* et écrire la méthode *main* affichant les différents tests à l'écran.

3. Modifier la procédure précédente pour en faire une fonction : quel peut être le type de ce qui est retourné par cette fonction ?

Testez cette fonction avec le jeu de test élaboré pour la question précédente.

Exercice 2. Somme de deux tableaux de même longueur

Le but de cet exercice est de construire un tableau qui contient la somme des éléments de deux autres tableaux de même longueur, case par case : chaque case du tableau que l'on construit contient la somme des deux cases de même indice dans les deux autres tableaux.

1. Ecrire une fonction *sommeTableaux()*. Cette fonction doit prendre en paramètre deux tableaux d'entiers, et retourner un tableau d'entiers qui contient la somme case à case des deux tableaux en paramètre.
Cette fonction doit vérifier que les deux tableaux reçus en paramètre sont de même longueur (dans le cas contraire, la fonction retournera *null*).

2. Ecrire une méthode *main()* réalisant un jeu de test pour la fonction *sommeTableaux()*. Vous testerez autre autres le cas où les deux tableaux sont de longueurs différentes.

Dessinez l'état de la mémoire pendant l'exécution de la méthode *sommeTableaux()* :

Exercice 3. Egalité de deux tableaux

Le but de cet exercice est de tester l'égalité de deux tableaux (même longueur et mêmes contenus case à case).

1. Ecrire une fonction *sontEgaux()*. Cette fonction doit prendre en paramètre deux tableaux d'entiers, et retourner un booléen indiquant si les deux tableaux sont égaux ou non. Si les tableaux ne sont pas de même longueur, la fonction renvoie *false*.

2. Ecrire une méthode *main()* réalisant un jeu de test pour la fonction *sontEgaux()*.

Dessinez l'état de la mémoire pendant l'exécution de la méthode *sontEgaux()*

Programmation Java 1A - TP9
Fonctions, procédures et tableaux

Exercice. Fonctions, procédures et tableaux

Le but de cet exercice est de bien comprendre ce qui se passe en mémoire lorsqu'on passe un tableau en argument d'une procédure ou d'une fonction.

Récupérez la classe *TestTableaux* que vous avez reçue par courrier électronique et ajoutez les commentaires manquants.
Essayez de deviner ce qui va être affiché à l'écran durant l'exécution du programme.

Dessinez ce qui se passe en mémoire tout au long de l'exécution de ce programme.

Programmation Java 1A - TP11
Tableaux (encore !)

Tableau d'indices

Ecrire une classe TableauIndices qui contient :

- une fonction *indicesDesNégatifs()* qui prend en argument un tableau d'entiers et retourne un tableau d'indices. Ces indices indiquent les emplacements contenant des entiers négatifs dans le tableau reçu en entrée .
  Exemple : la fonction *indicesDesNégatifs()* reçoit le tableau {4, -9, 0, 1, -5}, elle retourne le tableau {1, 4}.
  Indication : vous commencerez par compter le nombre d'entiers négatifs dans le tableau reçu en entrée, pour pouvoir créer un tableau d'indices qui contient le nombre exact de cases. Vous ferez donc deux parcours successifs du tableau d'entrée.
  Votre programme devra bien sûr fonctionner dans le cas où le tableau d'entrée ne contient aucun négatif..

- une fonction *main()* permettant de tester la fonction *indicesDesNégatifs()*.

Programmation Java 1A - TP10
Tableaux à deux dimensions

Bataille navale

On souhaite réaliser un jeu de bataille navale.

On choisit d'utiliser un plateau de jeu carré (par exemple 9 cases par 9). Ce plateau sera représenté en Java sous la forme d'un **tableau de caractères à deux dimensions** dans lequel les positions des bateaux seront stockées selon le codage suivant : **v**pour une case vide, **a** pour un bateau appartenant au joueur A, **b** pour un bateau appartenant au joueur B, et **2** lorsque les deux joueurs ont un bateau sur la même case (on utilise un seul tableau pour stocker les deux plateaux de jeu et donc les bateaux peuvent se superposer).

On limite la taille des bateaux à **une seule case**.

1. Dans la classe *BatailleNavale*, déclarer quatre **constantes** : *VIDE, JOUEURA, JOUEURB, LES2JOUEURS* qui valent respectivement 'v', 'a', 'b' et '2'.
Ecrire la méthode *main()* :
- déclarer un tableau de *char* à deux dimensions
- demander à l'utilisateur quelle taille il veut pour le plateau de jeu (entre 2 et 9)
- réserver la mémoire nécessaire
- remplir toutes les cases du tableau avec le caractère *VIDE*

2. Ecrire une méthode *afficher* qui prend en argument un tableau à deux dimensions représentant un plateau de jeu et affiche à l'écran son contenu sous la forme suivante (utiliser l'instruction **switch**) : une case vide est représentée à l'écran par un**espace**, un bateau du joueur A par la lettre A, un bateau du joueur B par la lettre B, et lorsque les deux joueurs ont un bateau sur une case on affiche le chiffre **2**.
De plus, on numérote les colonnes par des chiffres et les lignes par des lettres.

On obtiendra par exemple :

	1	2	3	4	5	6	7	8	9
A	A								
B					2				
C	B		A			B		A	
D									
E	B								
F							B		
G		2							A
H									
I		B				A			

Indications :
En Java, **char** est un type de base (comme **int**, **double**, **boolean...**). On peut donc par exemple écrire :

```
char lettre; // déclaration
lettre='a'; // affectation
if (lettre=='b') { ... } // comparaison avec ==

switch (lettre) { // bifurcation suivant dfférentes valeurs
   case 'a': ...; // cas du caractère 'a'
... }
```

Pour écrire les lettres de la première colonne, vous pouvez vous inspirer du code suivant (et vous souvenir des TP d'Archi sur les codes ASCII) qui affiche *ABCDE* à l'écran :

```
char lettre='A'; // la première lettre est 'A'
for (int i=0;i<5;i++) { // on veut afficher 5 lettres en tout
```

```
    System.out.print(lettre); // on affiche une lettre
    lettre=(char)(lettre+1); // on passe à la lettre suivante (le code ASCII augmente de 1)
  }
```

3. Ecrire une méthode *placerBateauxAleatoirement()* qui prend en argument un caractère indiquant le joueur pour lequel on place les bateaux, et un tableau représentant un plateau de jeu, et qui place aléatoirement *N* bateaux pour le joueur indiqué (où *N*est taille du plateau de jeu).

Attention à ne pas placer deux bateaux du même joueur sur une même case !

Indication : le code suivant fournit un nombre entier aléatoire entre 0 et 4 :

```
int aleatoire=(int)(Math.random()*5);
```

4. Ecrire une méthode *placerBateaux()* qui a les mêmes arguments que *placerBateauxAleatoirement()* mais qui permet au joueur indiqué de placer lui-même ses bateaux. Pour l'instant, on se contentera de saisir deux chiffres entre 1 et *N* pour représenter le numéro de ligne et le numéro de colonne (et non pas une lettre et un chiffre)...

5. Maintenant que vous disposez de méthodes permettant de placer les bateaux, il reste à écrire la méthode qui permet à un joueur de lancer une torpille et de voir si un bateau ennemi est touché

Ecrire une méthode jouer() qui prend en argument le nom d'un joueur et un tableau représentant le plateau de jeu et qui renvoie un booléen indiquant si le joueur qui vient de jouer a gagné.

Cette méthode permet de lire des coordonnées au clavier, et de mettre à jour le plateau de jeu en fonction de la case qui a été torpillée : si un bateau ennemi est touché, il disparait (attention au cas où les deux joueurs avaient un bateau sur la case torpillée) ; si un bateau du joueur se trouve seul sur la case ou s'il n'y a aucun bateau, on considère que le coup est dans l'eau (un joueur ne peut pas couler ses propres bateaux !).

Un message est affiché à l'écran indiquant le résultat du coup ("Coulé" ou "Dans l'eau").

On compte ensuite combien il reste de bateaux ennemis et s'il n'en reste plus on renvoie true (le joueur qui vient de jouer a gagné !).

6. Il ne reste maintenant plus qu'à compléter la méthode main() pour jouer à la bataille navale à deux joueurs.

La méthode main devra :

- demander à l'utilisateur de saisir la taille du plateau de jeu ;
- créer le tebleau de jeu et le remplir avec la constante VIDE ;
- demander au joueur A de placer ses bateaux
- demander au joueur B de placer ses bateaux
- (pour tester, vous pourrez afficher le plateau. Bien sûr, pour jouer vraiment, il ne faut pas que les joueurs voient la position des bateaux...)
- Tant qu'aucun joueur n'a gagné, faire jouer un joueur, (afficher le plateau pour tester), passer au joueur suivant.
- Afficher lle vainqueur.

7. (facultatif)

On peut bien sûr progammer une bataille navale à un joueur (joueur contre ordinateur)...

Programmation Java 1A - TP11
Utiliser des classes prédéfinies de Java
La classe ArrayList

L'objectif de ce TP¨est d 'apprendre à utiliser des classes prédéfinies de Java.
Nous utiliserons pour cela la classe ArrayList.

a. On considère l'algorithme ci-dessous :

```
Algorithme Noms
  // Récupère des noms de personnes. On ne sait pas à l'avance combien il y en a.
Variables
   noms : liste de texte  // la liste des noms
   i : entier // indice de parcours de la liste
   nom : texte // variable de saisie
Début
  noms.initialiser( )
  nom <- saisir ("entrez le nom d'une personne (tapez "stop" pour arrêter)")
  TantQue nom != "stop" Faire
    noms.insérer (nom)
    nom <- saisir ("entrez le nom d'une personne (tapez "stop" pour arrêter)")
  FinTanTQue
  Pour i de 0 à noms.longueur ( ) -1 Faire
    afficher (noms.lire(i) + "\t")
  FinPour
FinAlgo
```

Traduire en Java cet algorithme.
On se contentera pour l'instant de placer tout le code dans une méthode *main*.

Remarque 1 : pour utiliser la classe ArrayList, il faut indiquer au compilateur Java où celle-ci se trouve en indiquant en début de programme (avant la classe):
import java.util.ArrayList ;

Remarque 2 : pour tester en Java l'égalité de deux textes, les comparateurs = et != ne fonctionnent pas (pour la même raison que comparer deux tableaux avec = ou != ne fonctionne pas).
Si *s1* et *s2* sont des variables de type String, on peut utiliser *s1.equals(s2)*.
Par exemple, on pourra écrire le code suivant : *if (! s1.equals(s2)) ...*

b. Ecrire une méthode *afficheListe* qui prend en paramètre une ArrayList de String et affiche tous ses éléments. Modifier le programme principal pour faire appel à cette méthode.

c. Ecrire une méthode *compteTextesCommencantPar* qui prend en paramètre une ArrayList de String et un type *char*, et retourne le nombre de textes qui commencent par la lettre en question. Modifiez le programme principal pour faire appel à cette méthode et afficher le nombre de noms commençant par la lettre *f*.
Rappel : vous avez vu en début d'année comment récupérer le premier caractère d'un type String

sous la forme d'un type char.

Documentation Java

La documentation Java décrit (entre autres) l'ensemble des classes prédéfinies de Java.
Une copie de cette documentation est accessible sur www.iut.univ-paris8.fr/~rety/docJava/api/index.html

Consultez la documentation de la classe ArrayList.
Repérez en particulier les méthodes de cette classe.
Parmi les méthodes de la classe ArrayList, repérez en particulier celles correspondant aux opérations élémentaires sur les listes vues en cours : longueur, lire, affecter, insérer, supprimer.

d. Ecrire une méthode *supprimeTextesSeTerminantPar_e* qui prend en paramètre une ArrayList de String et supprime de la liste tous les textes se terminant par la lettre e (vous chercherez dans la documentation Java de la Classe String comment accéder au dernier caractère d'un type String).
Modifier le programme principal pour exécuter cette méthode et afficher ensuite la liste des noms pour vérifier le résultat.

e. En vous appuyant sur la documentation Java de la classe ArrayList, écrire une méthode qui teste si le nom Hyppolite est présent dans la liste (la méthode retourne un booléen). Ce test est sensible à la casse. Bien sûr : il faut modifier le programme principal pour mettre en oeuvre cette méthode et s'assurer qu'elle fonctionne...

f. En vous appuyant sur la documentation de la classe String, écrire une méthode qui met tous les noms de la liste entièrement en majuscules.

g. Repérer dans la documentation de la classe String comment comparer deux String en fonction de l'ordre alphabétique.
Ecrire une méthode *insérerTrié* qui, en supposant que la liste soit déjà triée, insère un nouveau nom de telle façon que la liste reste triée. Attention à l'écriture de la spécification de cette méthode. Celle-ci doit être précise et complète (et bien sûr : bien rédigée).
Ecrire une méthode main (mettre en commentaire l'ancienne méthode main afin de garder la trace de votre travail précédent) dans laquelle des noms sont saisis et insérés dans la liste à l'aide de la méthode *insérerTrié*. A la fin de la saisie la liste est affichée... celle-ci doit bien sûr être triée.

Remarque : cette façon de maintenir une liste triée n'est pas efficace. D'autres méthodes biens meilleures seront étudiées en deuxième année en cours d'Algorithmique.

h. (facultatif). Ecrire une méthode qui retourne (sous la forme d'un double) le nombre moyen de caractères composant les noms de la liste. Attention à donner un comportement cohérent à cette méthode en cas de liste vide (en particulier, la méthode ne doit pas planter). Ecrire soigneusement la spécification.

Programmation Java 1A - TP14
une première classe

L'objectif de ce TP est d'écrire une classe en Java et d'utiliser celle-ci dans un programme principal.

On considère la classe ci-dessous (classe étudiée en amphi) :

Classe Compte
// Définit ce qu'est un compte en banque.
// Cette version n'a pas de limite de découvert... on peut donc débiter de l'argent autant que l'on veut !!

Attributs
// le titulaire du compte
privé titulaire : texte
// le numéro de compte
privé numéroCompte : entier
// le solde du compte
privé solde : réel

Méthodes

publique Procédure Initialiser(**d** nomTitulaire : texte, **d** nouveauNum : entier)
// C'est le constructeur de la classe.
// Son rôle est d'initialliser un nouvel objet de type Compte (nouvelle instance de la classe Compte)
// en donnant des valeurs initiales aux attributs.
Début
titulaire <- nomTitulaire
numéroCompte <- nouveauNum
solde <- 0
FinProcédure

publique Procédure créditer (**d** montant : réel)
// crédite le compte du montant passé en paramètre.
Début
solde <- solde + montant
FinProcédure

publique Procédure débiter (**d** montant : réel)
// débite le compte du montant passé en paramètre
Début
solde <- solde - montant
FinProcédure

publique Fonction lireSolde ( ) : réel
// retourne le solde
Début

Retourner solde
FinFonction

publique Fonction lireTitulaire ( ) : réel
// retourne le nom du titulaire
Début
Retourner titulaire
FinFonction

FinClasse

Question 1.
Programmez cette classe en Java.
Ecrivez un *main* dans lequel sont créés et manipulés deux objets de type Compte (il vous faut donc deux variables de type Compte). Vous pourrez réaliser quelques opérations de crédit et de débit, puis afficher le solde avec le nom du titulaire.

Question 2.
Ajoutez dans la classe Compte une fonction booléenne *estADécouvert* qui retourne VRAI si le compte est à découvert.
Utilisez cette méthode dans le *main*.

Question 3 (facultatif).
Ecrivez un *main* dans lequel vous déclarez une ArrayList de Compte.
Un menu sera affiché permettant :

- d'ouvrir d'un nouveau compte (qui sera bien sûr ajouté dans l'ArrayList) ;
- de créditer un compte dont on connait le numéro ;
- de débiter un compte dont on connaît le numéro ;
- d'afficher le solde et le nom du titulaire d'un compte dont on connait le numéro ;
- d'afficher le numéro de compte quand on connaît le nom du titulaire ;
- d'afficher la somme et la moyenne des soldes de tous les comptes ;
- de supprimer un compte existant (on ne pourra supprimer un compte que si son solde est égal à zéro).

Programmation Java 1A - TP15

Comptes en banque ; Banque

Reprenez la classe Compte du TP 14. Copiez *Compte.java* dans votre dossier *tp15*.

L'objectif de ce TP est d'écrire la classe *Banque*, qui permet de gérer une liste de comptes en banque.
La classe *Banque* utilisera la classe *Compte*.

1. Classe *Banque*

Ecrivez la classe *Banque* (vous obtiendrez donc deux fichiers java dans votre répertoire : *Compte.java* et *Banque.java*. Vous pourrez compiler d'un coup l'ensemble en tapant *javac *.java*).

Une banque est un objet qui contient une liste de comptes (la classe *Banque* possède donc un attribut privé de type *ArrayList <Compte>*)

Les opérations publiques sur une *Banque* sont :

- ouvrir un nouveau compte (qui est bien sûr ajouté dans l'*ArrayList*) ;
- créditer un compte dont on connait le numéro ;
- débiter un compte dont on connaît le numéro ;
- afficher le solde et le nom du titulaire d'un compte dont on connait le numéro ;
- afficher le numéro de compte quand on connaît le nom du titulaire ;
- afficher la somme et la moyenne des soldes de tous les comptes ;
- afficher la liste des comptes qui sont à découvert (pour l'instant, on se contentera d'afficher les numéros des comptes qui sont à découvert)
- supprimer un compte existant (on ne pourra supprimer un compte que si son solde est égal à zéro).

Vous devrez écrire les méthodes (non static) permettant de réaliser ces opérations.

Attention : ces méthodes n'effectuent aucune saisie. Lorsqu'une méthode a besoin d'informations (par exemple, la méthode permettant de créditer un compte a besoin du numéro de compte et du montant à créditer), elle reçoit ces informations en paramètre.

Pour l'instant, lorsqu'un nouveau compte est créé, le numéro de compte sera reçu en paramètre (Dans le main, le numéro de compte sera donc entré par l'utilisateur. Lorsque vous ferez des tests, il faudra veiller à ce que les numéros de comtes saisis soient uniques). Nous verrons plus tard comment générer automatiquement des numéros de compte uniques.

N'oubliez pas d'écrire aussi le constructeur de la classe *Banque*.
Rappel : le rôle du constructeur est d'initialiser les attributs de la classe. Ici, le constructeur devra donc créer l'*ArrayList*.

Vous écrirez un *main* dans lequel est déclarée et créée une variable de type *Banque*.
Ce *main* affiche et fait fonctionner un menu permettant de choisir et d'effectuer les opérations listées ci-dessus.
Remarque : toutes les saisies doivent avoir lieu dans le main (par exemple, avant de créditer un compte en appelant la méthode non static correspondante, la saisie du numéro de compte et du montant à créditer ont lieu dans le *main*).

2. Méthodes publiques, méthodes privées
Plusieurs opérations nécessitent de récupérer un compte (un objet de type *Compte*) à partir de son numéro.
Plutôt que de dupliquer ce morceau de code dans plusieurs méthodes, il est préférable d'en faire un méthode (une fonction qui prend en paramètre un numéro de compte et retourne un type *Compte*). Cette méthode sera alors déclarée privée (*private*, en Java) car elle est destinée à être appelée par d'autres méthodes de la même classe, alors qu'un utilisateur de la classe *Banque* n'a aucune raison de l'appeler.

Ecrivez la méthode privée *chercherCompte (int numéro)*. Cette méthode retourne un objet de type Compte (c'est-à-dire une adresse car en Java les objets sont obligatoirement manipulés par l'intermédiaire d'une adresse).
Modifiez les autres méthodes pour faire appel à *chercherCompte*.

Programmation Java 1A - TP16
Comptes en banque ; Banque (suite)

Reprenez les classes du TP 15. Copiez *Compte.java* et Banque.java dans votre dossier *tp16*.
L'objectif de ce TP est de permettre à votre banque de gérer des comptes avec découvert.

La primitive *erreur("description de l'errreur")* utilisée en cours d'Algorithmique correspond aux *exceptions* en Java. Le mécanisme des exception est complexe, et sera étudié en deuxième année de DUT. Pour l'instant, nous nous contenterons d'utiliser (sans plus l'expliquer) l'instruction suivante :

throw new RuntimeException("description de l'erreur");

Lorsqu'une telle instruction est exécutée dans un programme, cela interromp l'exécution de la machine virtuelle avec affichage du message d'erreur.

1. comptes avec découvert autorisé

Modifiez la classe *Compte* de manière à gérer un découvert maximum autorisé.
Il faudra en particulier créer les accesseurs *lireDécouvertAutorisé* et *affecterDécouvertAutorisé*.
Pensez à expliciter les invariants de classe dans les commentaires de la classe.
Assurez-vous que ces invariants de classe soient respectés (encapsulation des données + vérification des invariants de classe par l'ensemble des méthodes).

2. banque avec gestion des découverts

Les exceptions sont là pour prévenir une "mauvaise" utilisation de la classe *Compte*. Si un programmeur (utilisateur de la classe) essaie par exemple d'affecter un découvert autorisé négatif, le programme va "planter" plutôt que de se retrouver dans situation incohérente (affecter le découvert autorisé négatif serait incohérent ; ne pas l'affecter et continuer le programme comme si de rien n'était le serait aussi).
Votre classe *Banque* doit faire "bonne" utilisation de la classe *Compte*, c'est-à-dire ne jamais déclencher d'exception. Autrement dit : votre programme ne doit jamais "planter" !
Modifiez la classe *Banque* de manière à ne jamais déclencher d'exception. Chaque fois que nécessaire, il faut donc tester les valeurs entrées par l'utilisateur.
Ajoutez une opération permettant de modifier la valeur du découvert autorisé.

3. Génération automatique des numéros de comptes

Ecrivez une méthode permettant de générer automatiquement un nouveau numéro de compte unique.
Conseil : si les numéros de comptes sont des entiers, il suffira de chercher le plus grand numéro de compte présent dans la banque, puis d'ajouter 1.
Cette méthode doit-elle être publique, ou bien privée ?
Modifiez l'opération de création d'un nouveau compte en faisant appel à cette méthode.

Programmation Java 1A - TP17
Pile et Stack

Exercice 1. Pile de nombres

1. Ecrire une classe *PileDeReels* ayant pour attributs un entier *sommet* et un tableau *elements* de *double*.

Ecrire un constructeur prenant en argument la hauteur maximale de la pile.

Ecrire les méthodes *valeurSommet()* qui retourne la valeur située au sommet de la pile, *empiler(double d)* et *depiler()* qui permettent d'ajouter ou de retirer une valeur dans la pile (et ne retournent rien), *estVide()* et*estPleine()* qui retournent chacune un booléen indiquant si la pile est vide ou non, ou si la pile est pleine ou non.

Remarque : dans les cas où la pile est vide, il vaut mieux que la méthode *valeurSommet()* ne renvoie pas de valeur ; pour cela, on peut utiliser l'instruction : *throw new RuntimeException("description de l'erreur");* (vous apprendrez en deuxième année le fonctionnement précis des exceptions en Java).

2. Ecrire une méthode *main()* dans la classe *PileDeReels* pour tester les différentes méthodes de la classe. On pourra par exemple créer une pile de 3 éléments, tester si la pile ainsi créée est vide, empiler trois valeurs, tester si la pile est pleine, puis essayer d'empiler une autre valeur, d'afficher et de dépiler quatre valeurs...

Exercice 2. Classe Stack

Regarder dans la documentation Java l'aide de la classe *Stack*.
Quel type de variable peut-on stocker dans un *Stack* ?
Quand on dépile un élément, quel type d'objet obtient-on ?

Ecrire une classe *StackDeReels* sur le même modèle que la classe *PileDeReels* mais avec en attribut un seul objet de type *Stack*.
Recopier la méthode *main()* de l'exercice **1.** pour tester les différentes méthodes (il vous suffit normalement de changer UNIQUEMENT la déclaration de votre pile : *StackDeReels* au lieu de *PileDeReel*)

ATTENTION : la classe *StackDeReels* doit avoir exactement les mêmes méthodes que la classe *PileDeReels* ; par exemple, la méthode *getSommet()* renvoie un *double* (et non pas un *Double* ou un *Object*).
Vous pourrez utiliser la classe *Double* qui permet de stocker un nombre réel sous la forme d'un objet. Regardez la documentation de la classe *Double*, en particulier la méthode *doubleValue()*.

Programmation Java 1A - TP18

Calculatrice avec Pile et Stack

Le but de ce TP est de simuler le fonctionnement d'une calculatrice. Cette calculatrice, basée sur une pile, fonctionne en notation polonaise inversée : pour effectuer l'opération *2+3*, on entrera la valeur *2*, puis la valeur *3*, et enfin on effectuera l'addition, qui dépilera les deux valeurs et empilera le résultat.

1. Ecrire une classe *CalculatricePile* ayant pour attribut un objet de type *PileDeReels* (voir TP précédent), représentant la pile dans laquelle sont stockés les valeurs et les résultats des opérations.

Ecrire un constructeur qui initialise les attributs.

Ecrire une méthode *entrerValeur()* qui prend en paramètre un double et l'empile.

Ecrire la méthode *public String toString()* qui affiche la valeur située au sommet de la pile (utile pour visualiser le résultat de la dernière opération).

Ecrire quatre méthodes correpondant aux quatre opérations de base (ajouter, soustraire, multiplier, diviser).

2. Ecrire une classe *TestCalculatricePile* contenant uniquement la méthode *main()*.

Dans cette méthode, déclarer un objet de type *CalculatricePile*, empiler plusieurs valeurs et tester les différentes opérations en affichant la valeur située au sommet de la pile après chaque opération (pour vérifier les calculs).

Que se passe-t-il dans le cas où on tente une opération alors qu'il n'y a qu'un élément dans la pile, ou dans le cas d'une division par 0 ? Comment modifier la classe *PileDeReels* et les quatre opérations pour que le programme ne plante pas ?

3. Sur le même modèle que la classe *CalculatricePile*, écrire une classe *CalculatriceStack* utilisant un attribut de type *StackDeReels* à la place de *PileDeReels*.
INDICATION : normalement il vous suffit de remplacer l'attribut et le début du *main()*, et tout devrait fonctionner... ou presque

4. Ecrire toujours sur le même modèle une classe *CalculatriceStack2* dans laquelle l'attribut est maintenant de type *Stack* et qui n'utilise plus ni *PileDeReels* ni *StackDeReels*.

Programmation Java 1A - TP19

Héritage d'attributs

Exercice. Personne, Skieur et Slalommeur

1. Ecrire une classe *Personne* ayant pour attributs **privés** une chaîne de caractères *nom* et un entier *age*.

Ecrire un constructeur prenant en arguments une chaîne de caractères et un entier.

Ecrire les accesseurs correspondant aux attributs.

Ecrire une méthode *toString* permettant d'afficher le nom d'une personne suivi de son age entre parenthèses (exemple : *Jean Dupont (36 ans)*).

Ecrire une méthode *main* pour tester la classe *Personne*.

2. Ecrire une classe *Skieur*, sous-classe de la classe *Personne*, ayant pour attribut **privé** supplémentaire un booléen *forfait* (qui est vrai si le skieur a un forfait, faux sinon).

Essayer de compiler la classe personne, sans rien ajouter d'autre. Que se passe-t-il ?

Ecrire un constructeur prenant en arguments une chaîne *nom*, un entier *age*, et un booléen *forfait*. Ce constructeur devra commencer par : *super(nom,age)* pour construire d'abord la partie *Personne* de l'objet *Skieur*. Que se passe-t-il à la compilation si vous mettez en commentaire la ligne contenant l'appel à *super* ?

Ecrire les accesseurs correspondant à l'attribut *forfait*.

Ecrire une méthode *toString* qui affichera une phrase comme : *Jean Dupont (36 ans) a un forfait* (ou *n'a pas de forfait* suivant le cas). On peut faire appel à la méthode *toString* de la classe *Personne* en écrivant :*super.toString()*

Ecrire une méthode *main* pour tester la classe *Skieur*.

3. Dans la méthode *toString* de *Skieur*, que se passe-t-il si vous essayez de récupérer le *nom* et l'*âge* du skieur sans passer par les accesseurs de la classe *Personne* ? Modifier les attributs de *Personne* pour qu'ils soient**protected** : que se passe-t-il maintenant dans la méthode *toString* ? Modifiez aussi l'attribut de la classe *Skieur* pour qu'il soit **protected**.

4. Ecrire une classe *Slalommeur*, sous-classe de la classe *Skieur*, ayant pour attribut supplémentaire *temps* de type double (représentant son temps à l'épreuve de slalom).

Ecrire un constructeur prenant en argument un *Skieur*. Ce constructeur servira à inscrire des skieurs à l'épreuve de slalom, et initialisera leur temps à 0. Si le skieur n'a pas de forfait, le constructeur affichera un message indiquant : *Jean Dupont doit se procurer un forfait avant le début de l'épreuve*.

Ecrire les accesseurs correspondant à l'attribut *temps*. L'accesseur permettant de régler le temps devra vérifier que le slalommeur a un forfait ; dans le cas contraire, l'attribut temps est réglé à 0 car sans forfait on ne peut pas participer !

Ecrire une méthode *toString* qui affichera une phrase comme : *Jean Dupont (36 ans) : 188 secondes* (ou bien si le skieur n'a pas de forfait : *Jean Dupont ne peut pas participer sans forfait !*).

Ecrire une méthode *main* pour tester la classe *Slalommeur*.

Programmation Java 1A - TP20
Héritage (suite)

Médiathèque

Une médiathèque souhaite gérer les livres, CD et cassettes vidéos disponibles au prêt.
Chaque article disponible au prêt possède une cote, servant à le repérer dans la médiathèque (ex : AZ123).

Un livre est caractérisé par son titre, son auteur, son nombre de pages.
Un CD est caractérisé par son titre, son interprète, sa durée.
Une vidéo est caractérisée par son titre, son réalisateur, une liste d'acteurs, et sa durée.

1. Ecrire une classe *Article* ayant pour attributs **protected** une chaîne de caractères *cote*, une chaîne de caractère *titre*, et un booléen *emprunte* indiquant si l'article est emprunté (*true*) ou disponible (*false*). Ecrire le constructeur, et les accesseurs.

2. Ecrire trois classes *Livre*, *CD*, et *Video* dérivant de la classe *Article* et permettant de stocker chacun des trois types d'objets de la médiathèque. Ecrire dans ces trois classes les constructeurs et accesseurs de chaque classe, ainsi que les trois méthodes *toString()* permettant l'affichage des propriétés de chaque article.

3. Ecrire une classe *Mediatheque* ayant pour attribut un tableau d'*Article* et un entier *nbArticles*. Dans cette classe, on écrira les méthodes suivantes : *ajouterArticle(Article a)*, *emprunterArticle(String cote)*, *rendreArticle(String cote)*, *getNbArticles()*, *listerArticles()*, *voirDetailArticle(String cote)*. On pourra aussi écrire une méthode *Article getArticle(String cote)* qui renvoie l'article de cote donnée.
On écrira également une méthode *main()* permettant de tester les différentes méthodes de toutes les classes.

4. Regarder la documentation de la classe *ArrayList* : à quoi sert cette classe, quelles sont les méthodes que vous allez utiliser pour ajouter ou retirer des articles du "tableau" ? Ecrire une classe *MediathequeArrayList* ayant pour attribut un objet de type *ArrayList*. Dans cette classe, on écrira les mêmes méthodes qu'à la question précédente, et une méthode *main()* permettant de tester les différentes méthodes de toutes les classes.

Programmation Java 1A - TP21

Collections

Tableau trié ou non trié de personnes

On souhaite écrire une classe Java qui nous permettra de stocker des personnes, éventuellement triées par ordre alphabétique de nom.
Recopier la classe *Personne* du TP 18 dans un nouveau répertoire.

1. **Tableau non trié**

   Lire la documentation de la classe *ArrayList* (au moins le début du texte, ainsi que les constructeurs et méthodes disponibles).
   Cette classe va nous permettre de créer un "tableau" sans utiliser la notation avec les crochets.

   Ecrire une classe *TableauPersonnes* ayant pour seul attribut un objet de type *ArrayList*.
   Dans cette classe, écrire un constructeur, une méthode *void ajouter(Personne p)* qui permet d'ajouter une personne à la liste, une méthode *toString()* qui sera utile pour afficher toutes les personnes (on utilisera la méthode*toString* de la classe *ArrayList*), et une méthode *main* dans laquelle vous créez quelques personnes, les ajoutez au tableau, et affichez le contenu du tableau.
   Vérifiez bien que si vous ajoutez les personnes dans le désordre, elles s'affichent de façon non triée (l'affichage se fait dans l'ordre d'ajout).

   2. Tableau trié

   Lisez la documentation de la classe *TreeSet* (au moins le début du texte, ainsi que les constructeurs et méthodes disponibles).

   Recopiez la classe *TableauPersonnes* dans un deuxième fichier nommé *TableauTriePersonnes.*
   Dans la classe*TableauTriePersonnes,* remplacez le type de l'attribut *ArrayList* par *TreeSet* et faites les autres modifications nécessaires (constructeur, *main*, etc.) puis compilez et exécutez. Les personnes sont-elles triées ? Que se passe-t-il ?
   Modifier la classe *Personne* en ajoutant sur la première ligne : *class Personne implements Comparable* pour indiquer à Java que les objets de type *Personne* peuvent être comparés à d'autres objets.
   Compilez : que se passe-t-il ?

   Ecrivez une méthode *public int compareTo(Object o)* : elle doit renvoyer un entier dont le signe indique si l'objet sur lequel on appelle cette méthode est plus grand ou plus petit que l'objet passé en paramètre. Vous pourrez utiliser la méthode *compareTo* de la classe String (allez voir la documentation) pour comparer les noms des deux personnes. La première ligne de *compareTo* doit convertir l'objet *o* en *Personne* (ici on ne va comparer que des personnes entre elles, pas des objets quelconques) : *Personne p=(Personne)o;*

Dans le *main*, appelez la méthode *compareTo* sur plusieurs personnes pour regarder la valeur de l'entier renvoyé et comprendre le fonctionnement de cette méthode.
Remarque : si on ajoute dans un *TreeSet* des objets que Java sait comparer (voir les documentation des classes *String*, *Integer*, *Double* par exemple), on n'a pas besoin d'écrire de méthode *compareTo*...

Modifier la méthode *compareTo* de la classe *Personne* pour que les personnes soient désormais triées par ordre d'âge.

Programmation Java 1A - TP22

Interfaces graphiques

Calculatrice

Le but de ce TP et du suivant est de programmer une calculatrice simple (offrant seulement les 4 opérations de base).

Nous allons écrire la calculatrice en quatre étapes (vérifiez bien que chaque étape fonctionne avant de passer à la suivante !) : tout d'abord l'affichage des boutons à l'écran, puis l'ajout d'un écouteur à chaque bouton, puis dans le prochain TP les clics sur les chiffres qui permettent de saisir un nombre correctement, et enfin les clics sur les opérateurs.

Remarque : bien que ce programme soit graphique, vous pouvez utiliser la méthode *System.out.println* pour afficher dans le terminal des indications sur le déroulement du programme : cela sera utile par exemple pour vérifier que les nombres que vous saisissez avec les touches de la calculatrice sont stockés comme il faut dans les variables...

1. Les touches (attributs et constructeur de la classe Calculatrice)

La calculatrice contient un affichage, et des boutons.

Ecrire une classe *Calculatrice* étendant la classe *Panel* afin d'obtenir l'affichage ci-contre.

Vous pouvez utiliser un tableau de boutons pour stocker les touches de chiffres, cela vous permet d'initialiser rapidement (avec une boucle) chacun de ces boutons.

Tous les boutons seront placés dans un *Panel* nommé *touches*, afin de pouvoir obtenir la disposition que l'on souhaite (*GridLayout* de 4 sur 4).

Il faut également un champ de texte pour l'affichage des nombres, nommé *affichage*.

Pour la disposition du champ de texte et du panel *touches* dans la fenêtre principale, vous utiliserez un *BorderLayout* avec le champ de texte au nord et les touches au centre.

Ecrire la méthode *main* dans laquelle vous créerez une fenêtre ; dans cette fenêtre vous placerez un objet de type *Calculatrice.*

Remarque : voici le code qui permet qu'une fenêtre se ferme lorsqu'on appuie sur la petite croix dans la barre de titre. Ce code ajoute un écouteur de fenêtre à l'objet *f*, cet écouteur est de type *WindowAdapter* et indique que lorsqu'un message de fermeture de fenêtre arrive (c'est-à-dire lorsqu'on clique sur la croix) il faut faire disparaître la fenêtre et arrêter le programme.

```
Frame f;
f.addWindowListener(new WindowAdapter() {
    public void windowClosing (WindowEvent e) { // lorsqu'on ferme la fenêtre
    e.getWindow().dispose(); // on fait disparaître la fenêtre
    System.exit(0); // on arrête le programme
    }
});
```

2. L'écouteur

A chaque bouton doit être associé un écouteur : écrire une classe *CalculatriceEcouteur* et compléter la classe *Calculatrice*.

Maintenant, que doit-il se passer lorsqu'on appuie sur l'un des boutons ? Cela dépend si c'est l'un des 5 opérateurs, ou alors un chiffre ou le point décimal...

Ecrire la fonction *actionPerformed* qui suivant le cas fera appel à l'une des deux fonctions que nous verrons dans le prochain TP : *traiterSaisieNombre()* ou *traiterClicSurOperateur()*.

Pour le moment, contentez-vous dans chacune des deux méthodes de traitement d'afficher dans le terminal le nom du bouton sur lequel on a cliqué.

Programmation Java 1A - TP23

Interfaces graphiques (suite)

Calculatrice (suite)

Vérifiez que les deux étapes du TP précédent fonctionnent correctement : affichage de la calculatrice, et fonctionnement des boutons avec affichage de chaque label dans le *TextField*.

Il nous reste à écrire les deux dernières étapes (vérifiez bien que chaque étape fonctionne avant de passer à la suivante !) : après l'affichage des boutons à l'écran et l'ajout d'un écouteur à chaque bouton, il faut maintenant gérer les clics sur les chiffres qui permettent de saisir un nombre correctement, et les clics sur les opérateurs.

Remarque : Vous pouvez savoir quel bouton a été cliqué grâce à la ligne : *Button b = (Button)(e.getSource());* puis tester par exemple si *b==bouton_plus* (où *bouton_plus* est le nom du bouton correspondant à la touche + que vous aurez **déclaré en attribut**).

1. Le traitement des chiffres

Pour que la calculatrice fonctionne, il faut qu'elle sache saisir des nombres à plusieurs chiffres, et avec au plus un point décimal !

Ecrire la fonction *void traiterSaisieNombre(String leLabel)* qui prend en argument une chaîne représentant l'intitulé d'un bouton (soit un chiffre, soit le point décimal).

Pour écrire cette fonction, vous pourrez déclarer en attribut de la classe *Calculatrice* un booléen *debut* qui sera vrai lorsqu'on est au début de la saisie d'un nombre (on saisit le premier chiffre) et faux lorsqu'on a déjà saisi au moins un chiffre, ainsi qu'un booléen *decimal* qui sera vrai dès qu'on a saisi un point décimal dans un nombre.

2. Le traitement des opérateurs

Ecrire la fonction *void traiterClicSurOperateur(String leLabel)* qui gère les clics sur l'un des 5 opérateurs.

Pour cette fonction, vous aurez besoin d'un attribut *dernierOperateur* de type *String* dans la classe *Calculatrice*, indiquant le label de l'avant-dernier opérateur saisi (celui qui avait été traité lors du passage précédent par cette fonction).

Vous aurez également besoin d'un attribut *premierArgument* de type *double* (le deuxième argument n'a pas besoin d'être en attribut : ce sera une variable locale à la fonction *traiterClicSurOperateur*).

Ainsi, si ce dernier opérateur était =, ça signifie que depuis on a saisi un nouveau nombre, qui doit servir de premier argument à une nouvelle opération.

Si le dernier opérateur était l'une des quatre opérations, cela signifie que depuis on a saisi le deuxième argument, et qu'il faut maintenant effectuer l'opération et afficher le résultat.

Pour cette fonction, vous pourrez utiliser les méthodes *Double.parseDouble(String)* et *Double.toString(double)* qui permettent de convertir respectivement une chaîne en nombre et un nombre en chaîne.

Programmation Java 1A - TP24

Interfaces graphiques : classe Canvas

Les méthodes drawArc et fillArc de la classe Graphics

Le but de ce TP est d'écrire un programme qui permet de tracer des arcs (pleins ou en fil de fer) en lisant au clavier les valeurs de l'angle de départ et la longueur de l'arc à tracer (sous forme d'angle également) :

1. Regarder dans la documentation Java à quoi servent les deux méthodes *drawArc* et *FillArc* et comment elles s'utilisent.

Regarder la documentation de la classe *Canvas* : vous remarquerez notamment qu'on doit écrire une méthode *paint* lorsqu'on étend la classe *Canvas*. Regarder à quoi ressemble cette méthode *paint* dans la classe *Canvas*, ainsi que les méthodes *update* et *repaint* dans la classe Component.

2. Ecrire une classe *MonCanvas* qui étend la classe *Canvas* et qui a trois attributs : *angleDebut* et *angleTotal* sont des entiers et permettent de savoir à partir de quel angle on dessine et sur quelle longueur d'arc, le booléen *plein* indique si on dessine en fil de fer (*false*) ou pas *(true*); cette classe contiendra une méthode *paint*.

Ecrire la méthode *paint* qui permet de dessiner l'arc (soit plein, soit en fil de fer) en rouge (utiliser *setColor* de la classe *Graphics*), ainsi que les deux axes du repère en noir, et éventuellement des graduations comme sur la figure ci-contre (utiliser pour cela la méthode *getBounds* de la classe *Component* ou les méthodes *getWidth* et *getHeight*).

3. Ecrire une classe *MonPanel* permettant de mettre en place l'interface du programme : les boutons en bas, le *canvas* au dessus. On pourra tout écrire dans le constructeur, sans rien mettre en attribut.

4. Ecrire une classe *MonEcouteur* qui contient deux attributs de type *TextField* et un attribut de type *Canvas*. Cette classe est chargée de mettre à jour le booléen *plein* et les deux entiers *angleDebut* et *angleTotal* et de rafraichir l'affichage lorsqu'on clique sur l'un des boutons.

5. Ecrire une classe *TestCanvas* contenant une méthode *main* pour tester le programme.

Programmation Java 1A - TP25

Interfaces graphiques : Swing

Interface graphique avec Swing et HTML

Le but de ce TP est de réaliser le programme ci-contre qui permet de saisir du texte dans un JTextField (en haut), éventuellement au format HTML, qui montre l'apparence dans le JLabel en dessous, et qui permet de sauver le texte tapé dans un fichier.

Dans l'exemple ci-contre, si on positionne le curseur après "<html" et qu'on tape le caractère '>', on souhaite que l'affichage change dans le JLabel et devienne simplement le mot "test" écrit en rouge.

Dans le TP d'aujourd'hui, on s'intéresse uniquement à l'interface et aux écouteurs.
Lorsqu'on cliquera sur l'un des boutons, on affichera juste un message dans la console.

Pour les plus rapides, lorsqu'on clique sur un bouton, on ouvrira un JFileChooser qui permettra de sélectionner le fichier à charger ou à sauver ; dans ce cas, le message dans la console reprendra le nom du fichier sélectionné par l'utilisateur (par exemple : "on charge le fichier test.txt").