Yann.Secq@univ-lille.fr
Fabien.Delecroix@univ-lille.fr

TP n°1

Objectifs : Écrire, compiler et exécuter ses premiers programmes.

Étapes à réaliser lors de chaque séance de TP

Il est conseillé d'organiser votre écran en trois zones : sur la gauche pour 50% de l'écran, mettre l'énoncé en PDF dans le navigateur, sur la droite sur les 50% restant, mettre l'éditeur de code (VSCode) avec les 2/3 consacré au code source et 1/3 pour le terminal (que vous pouvez ajouter en allant dans les menus). Il est préférable d'utiliser le terminal intégré à l'éditeur, ainsi il n'y a que deux applications : le navigateur et l'éditeur de code/terminal.

Préparation du contexte pour le nouveau TP

En début de chaque séance de TP, commencez par lancer cette commande :

```
...> ijava start
```

Lors de la première exécution de cette commande, le répertoire `~/ijava2` est automatiquement créé. Nous allons travailler durant tout le semestre dans ce répertoire `~/ijava2` et **il ne faut pas le déplacer**.

Pour se placer dans ce répertoire, utilisez cette commande :

```
...> cd ~/ijava2       -> cd = change directory
```

La commande `cd` correspond à `change directory` et vous place directement dans le répertoire `~/ijava2`.

Au début de chaque nouveau TP, vous devez créer un nouveau répertoire dans `~/ijava2` en respectant la syntaxe `tpX` où X est le numéro du TP. Ainsi, pour ce premier TP après vous être placé dans le répertoire `~/ijava2`, tapez cette commande :

```
...> mkdir tp1         -> mkdir = make directory
```

La commande `mkdir` correspond à `make directory` et crée le répertoire `tp1` là où vous êtes (normalement dans `~/ijava2`).

Finalement, on se place dans le répertoire `tp1` afin de débuter le TP :

```
...> cd tp1
```

Afin d'avoir la liste des exercices du TP ainsi que votre progression, tapez la commande :

```
...> ijava status      -> affiche la situation actuelle du TP
```

La commande `status` permet d'afficher la situation actuelle du TP et votre progression au fur et à mesure de la séance.

Vous devriez avoir un affichage similaire à celui-ci :

```
=== Student Progress Overview ===

Session: tp1

#   | Exercise                | Status | Score
---|-------------------------|--------|-----------------
1   | QCM0_accord_usage_traces |  ----  | (not tested)
2   | QCM1_ijava_commandes     |  ----  | (not tested)
3   | QCM2_ijava_program       |  ----  | (not tested)
4   | QCM3_types_variable      |  ----  | (not tested)
5   | HelloWorld               |  ----  | (not started)
6   | JeuxDeType               |  ----  | (not started)
7   | QCM4_chaines             |  ----  | (not tested)
8   | JeuxDeMots               |  ----  | (not started)
9   | Verlan                   |  ----  | (not started)
```

```
10 | Conversions              |  ----  | (not started)
11 | CaractereSuivant         |  ----  | (not started)
12 | MajMin                   |  ----  | (not started)
13 | RenduMonnaie             |  ----  | (not started)
14 | ChiffreAuHasard          |  ----  | (not started)
15 | LettreAuHasard           |  ----  | (not started)
16 | De                       |  ----  | (not started)
```

Cycle de développement durant une séance de TP

Les étapes à suivre lorsque vous réalisez les exercices du TP sont toujours les mêmes :

— **ATTENTION : il faut toujours effectuer les exercices dans l'ordre donné (ils sont classés du plus simple au plus complexe)**.

— obtenir le menu de la séance avec la commande : `ijava status`
La liste des exercices à réaliser s'affiche comme dans l'exemple ci-dessus.

— débuter un exercice avec la commande : `ijava init <NomExercice>`
Si c'est un QCM d'auto-évaluation, un onglet avec le questionnaire s'ouvre dans votre navigateur. Si c'est un exercice de programmation, un squelette est généré automatiquement et vous devez le compléter.

— utiliser un éditeur de texte pour modifier le code du programme.

— compiler votre programme avec la commande : `ijava compile <NomExercice.java>`
Lorsqu'il n'y a pas d'erreur un fichier <NomExercice.class> est généré, il est alors possible de passer à l'étape suivante pour l'exécuter.

— exécuter votre programme avec la commande : `ijava execute <NomExercice>`
ATTENTION : on donne juste le nom du programme, il ne faut aucune extension : ni .java, ni .class !!!.

— tester votre programme avec la commande : `ijava test <NomExercice>`
Au début, ce sont des tests écrits par les enseignants, mais plus tard dans le semestre, vous écrirez aussi vos propres tests.

Attention, si vous utilisez un ordinateur hors des salles de TP, pour que la commande `ijava` fonctionne, il faudra recréer l'alias en précisant le chemin de la librairie `ijava.jar`. Par exemple, si le fichier `ijava.jar` se situe à la racine de votre compte `~/ijava.jar`, on modifiera le fichier `.bashrc` en ajoutant :

```
alias ijava='java -cp ~/ijava.jar:. ijava2.clitools.MainCLI '
```

Exercice 1 : QCM0 : Demande d'accord d'usages des traces anonymisées à des fins de recherche

Avant de débuter le TP, nous souhaiterions obtenir votre accord afin de pouvoir exploiter les traces d'activités de programmation à des fins de recherche. Ceci nous permet d'améliorer le contenu de cet enseignement en affinant la progression pédagogique.

Comme expliqué en amphi, l'outil ijava génère des traces des différentes actions que vous réalisez avec, ce qui est principalement utilisé afin que vous puissiez visualiser la progression durant les séances de TP, mais aussi au niveau de l'équipe pédagogique pour le suivi des étudiants ou étudiantes qui pourraient bénéficier du dispositif de tutorat.

Ainsi, l'usage à des fins pédagogiques des traces est automatique, mais par contre, pour l'usage de ces mêmes données à des fins de recherche, nous demandons votre accord explicite.

Merci d'avance pour votre aide !

Afin de lancer ce premier "QCM", qui est plutôt une "signature" électronique de votre accord/désaccord, veuillez taper la commande suivante :
Vérifiez bien que vous êtes dans le répertoire `~/ijava2/tp1` et tapez `ijava init QCM0_accord_usage_traces`.

Votre navigateur devrait ouvrir un nouvel onglet avec les questions du QCM. N'oubliez pas de valider et ne vous formalisez pas par rapport à la "bonne réponse". Les QCM ne sont pas utilisés pour des accords habituellement ...

Exercice 2 : QCM1 : commandes de bases ijava

Durant les TP, ces QCM d'auto-évaluation vous permettent d'évaluer votre compréhension de certains concepts. Ces informations ne sont jamais utilisées dans le cadre de l'évaluation du module. C'est donc d'autant plus bénéfique si vous le faîtes sans "tricher", c'est-à-dire chercher les solutions dans l'énoncé ou le support de cours, ou encore en demandant à vos voisins ou voisines.

Vérifiez bien que vous êtes dans le répertoire `~/ijava2/tp1` et réalisez cette commande pour lancer le QCM : `ijava init QCM1_ijava_commandes`.

Votre navigateur devrait ouvrir un nouvel onglet avec les questions du QCM. N'oubliez pas de valider à la fin pour avoir le résultat.

Quelques rappels de cours

Structure d'un programme iJava

Pour l'instant, les programmes que vous écrirez en ce début de ce semestre auront la structure suivante (**en gras**, ce qui ne change pas d'un programme à l'autre) :

```
class MonProgramme extends Program{

    // fonction principale appelée lors de l'exécution du programme
    void algorithm (){
        // ceci est un exemple de commentaire, pratique pour les humains ;)
        String msg;                 // déclaration d'une variable de type chaîne
        msg = "Bonjour l'univers"; // affectation d'une vairable (initialisation)
        println(msg);               // appel de la fonction d'affichage
        msg = readString();         // appel de la fonction de saisie au clavier
        println(msg);               // affichage de la saisie de l'usager
    }
}
```

Le squelette de programme ci-dessus est commenté. La syntaxe des commentaires est `//` pour une ligne unique et `/* ... */` pour un bloc (une suite de lignes). Ce programme minimal peut bien être compilé, il ne produit cependant aucune action à l'exécution, sa fonction principale ne comportant aucune instruction. Dans l'exercice de ce TP, il s'agira pour chaque exercice d'écrire un nouveau programme, avec une suite d'instructions correcte et produisant le résultat demandé dans la consigne.

Exercice 3 : QCM2 : anatomie d'un programme simple en ijava

Rappel : C'est donc d'autant plus bénéfique si vous le faîtes sans "tricher" ...

Exercice 4 : L'incontournable HelloWorld

A l'aide de la commande `ijava init HelloWorld`, générez le squelette du code source pour cette exercice. Utilisez ensuite l'éditeur de texte afin (VSCode) d'avoir un programme qui affiche `Hello World !`
Vous n'avez pas à écrire plus d'une ligne de code !

Compilez votre programme (`ijava compile HelloWorld.java`) et exécuter le (`ijava execute HelloWorld`) afin de voir si le résultat attendu est bien affiché.

Finalement, lancez les tests avec `ijava test HelloWorld`, pour vérifier que vous avez la bonne solution.

Variables et types

Un type définit la nature d'une information et son intervalle de valeurs. Pendant ce premier semestre, dans un premier temps, vous allez utiliser :

— le type `boolean`,
— des types numériques: `int`, `double`, `char`,
— le type des chaînes de caractères `String`.

Une variable permet de stocker une information. En Java, chaque variable a un type et doit être déclarée avant toute utilisation.

Pour affecter une valeur à une variable, on utilise l'opérateur =.

Une constante stocke une valeur qui ne peut pas être modifiée (mot clé `final` avant la type pour indiquer que c'est une constante).

Voici quelques exemples de déclarations de variables et de constantes, et des précisions sur les noms de variables qu'on utilise en Java.

```java
int age;  // Un nom de variable commence par une lettre minuscule.
age = 19; // On donne à la variable âge la valeur 19
final double PI = 3.1415; //On déclare une constante en utilisant le mot réservé final.
// Si un nom de variable est composé, on marque le changement de mot par une majuscule.
String leMotLePlusLong;
```

Exercice 5 : QCM3 : types, variable et affectation

Troisième QCM, plus besoin de rappel, vous êtes rôdés mainteannt ;)

Exercice 6 : Types manquants

Dans le programme ci-dessous, vous devez compléter par le bon type partout où les ... apparaîssent.

```java
class JeuxDeType extends Program {

  void algorithm(){
    ... prenom = "Alan";
    ... nom = "Turing";
    ... naissance = 1912;
    ... annee = 2022;
    ... age = annee - naissance;
    ... initiale = charAt(prenom,0);
    println(initiale + ". " + nom + " aurait eu " + age + "ans en " + annee);
  }

}
```

Pour débuter l'exercice, utilisez la commande : `ijava init JeuxDeType`.

Éditez ensuite le squelette qui a été généré afin de compléter avec les types appropriés. Ensuite, compilez votre programme avec la commande : `ijava compile JeuxDeType.java`.

Finalement, exécutez votre programme avec : `ijava execute JeuxDeType` Vous devriez obtenir l'affichage suivant :

```
A. Turing aurait eu 110 ans en 2022
```

Afin de vérifier cela automatiquement, utilisez la commande : `ijava test JeuxDeType`

Après cette première étape, modifiez les valeurs des variables de manière à ce qu'il réalise le même comportement mais cette fois concernant Ada Lovelace née en 1815.

Amusons-nous avec les chaînes

Voici les fonctions disponibles sur les chaînes de caractère (type `String`) qui vous seront utiles dans cette partie.

— `int length (String uneChaine)`
  Retourne le nombre de caractères du paramètre `uneChaine`
— `char charAt(String uneChaine, int indice)`
  Retourne le caractère situé à l'indice indice dans la chaîne `uneChaine`.
  ATTENTION: le premier caractère a pour indice `0` !
— `String substring(String uneChaine, int indiceDebut, int indiceFin)`
  Retourne une copie de la chaîne `uneChaine` en commençant la copie à l'indice `indiceDebut` et en s'arrêtant à l'indice `indiceFin-1` (!).
  ATTENTION: le premier caractère a pour indice **0** !
— `boolean equals(String chaine1, String chaine1)`
  Test l'égalité de deux chaînes de caractères.
  ATTENTION: l'opérateur == ne doit pas être utilisé avec les chaînes !

Exercice 7 : QCM4 : le type String et les fonctions associées

Quatrième QCM : n'oubliez pas de ne pas regarder cet énoncé !

Exercice 8 : D'une chaîne à une autre

Étudiez le programme donné ci-dessous :

```
class JeuxDeMots extends Program {

   void algorithm(){
     ... mot = "etat";
     ... resultat;
     ... premiereLettre = charAt(mot, 0);
     ... resteDuMot = substring(mot, 1, length(mot));
     resultat = resteDuMot + premiereLettre;
     println(resultat);
   }
}
```

1. A votre avis, que réalise ce programme ? Notez cela sur votre brouillon.
2. Générez le squelette corrrespondant : `ijava init JeuxDeMots`
3. Éditez le code source pour compléter les déclarations de type manquantes.
4. Exécutez le programme pour vérifier votre hypothèse : `ijava execute JeuxDeMots`
5. Testez votre programme : `ijava test JeuxDeMots`
6. Choisissez un autre mot de départ et ré-exécutez votre programme pour être sûr du traitement réalisé par cet algorithme.

Exercice 9 : Affichage verlan

On souhaite créer un programme `Verlan`, qui inverse la deuxième partie du mot avec la première. Par exemple, avec la chaîne `"louche"`, le programme affichera *chelou*. Lisez attentivement le programme suivant :

```
 class Verlan extends Program {
   void algorithm(){
     ... mot = readString();
     ... tailleMot = ;
     ... indiceMilieu = ;
     ... debut = ;
     ... fin = ;
     println(fin+debut);
   }
 }
```

1. Générez automatiquement le squelette à l'aide de l'action dédiée de `ijava`.
2. Complétez ce programme, puis compilez le et exécutez le afin de vérifier qu'il fonctionne comme souhaité lorsque l'on saisit `louche`.
3. Une fois que votre programme marche avec cet exemple, vérifiez qu'il fonctionne bien dans tous les cas de figure. Par exemple, en saisissant la chaîne `"cheval"` par `"malin"` dans votre programme, après compilation et exécution, celui-ci devrait afficher *linma* si il est valide. Si tel n'est pas le cas, modifiez votre programme pour qu'il soit suffisamment général pour couvrir ces 2 cas.
4. Lorsque vous êtes sûr de votre programme lancer la commande : `ijava test Verlan`

Conversions de types

Dans certaines situations, il est nécessaire de pouvoir changer le type d'une information. La plupart du temps ce premier semestre, cela concernera la conversion de réels vers des entiers ou des entiers vers les caractères (ou vice-versa).

L'opérateur permettant de forcer un changement de type, ou opérateur de *cast* en anglais, utilise une syntaxe singulière: un type entouré de parenthèses.

Exemple: `int note = (int) 15.6;`

Dans l'exemple ci-dessus, le nombre réel `15.6` est converti vers l'entier `15` grâce à l'opérateur de cast `(int)`.

ATTENTION la conversion dans ce cas correspond à la partie entière du nombre réel (troncature) et non pas un arrondi mathématique.

ATTENTION l'opérateur de forçage de type est plus prioritaire que les opérations arithmétiques, soyez donc attentifs au parenthésage de vos expressions !

La conversion de types est possible uniquement entre types compatibles. Tous les types numériques sont compatibles entre eux. En iJava, `char` peut-être considéré comme un type numérique et donc il est possible de convertir une valeur de type `char` en valeur de type `int` et vice versa (cf. code ASCII). Par contre, le type `boolean` et le type `String` ne sont pas compatibles avec les types numériques. Ainsi, même si ça peut paraître étrange pour l'instant, l'opérateur de forçage de type ne permet pas de convertir une valeur de type `char` en valeur de type `String`.

Exercice 10 : D'un type à l'autre

Cet exercice vous aidera à comprendre le fonctionnement de l'opérateur de conversion. Dans un premier temps, ne copiez pas et ne cherchez pas à compiler ce programme.

```
class Conversions extends Program {

  void algorithm(){

    print( "(int) 4.6 ->" );
    println( (int) 4.6 );

    print( "(double) 4 ->" );
    println( (double) 4 );

    print("2.1 + 3 -> ");
    println(2.1 + 3);

    print("(int) 'A' -> ");
    println( (int) 'A' );

    print(" (char) 66 -> ");
    println( (char) 66);


    print(" (int) 3.7 * 2 -> ");
    println( (int) 3.7 * 2);

    print("(int) (3.7 * 2) -> ");
    println((int) (3.7 * 2));

    print(" \"ABC\" + (char) 65 ");
    println( "ABC" + (char) 65 );

  }
}
```

1. Essayez de prédire le résultat du programme `Conversions` ci-dessus en écrivant dans un fichier texte les affichages qu'il produira à l'exécution.
2. Appeler votre enseignant·e une fois que c'est terminé pour vérification.

Dans les exercices qui suivent, vous verrez des applications concrètes qui requièrent la conversion de types.

Exercice 11 : Suivant !

On souhaite disposer d'un programme qui nous indique quel est le caractère suivant dans la table ASCII pour un caractère donné. On souhaite que l'usager choisisse le caractère ce qui implique de réaliser une saisie au clavier. Pour ce faire, on fait un appel à la fonction `readChar()`, qui ne prend pas de paramètre et retourne le premier caractère saisi par l'utilisateur au clavier. Voici le programme presque complet si ce n'est qu'il manque la partie concernant le passage au caractère suivant.

```
class CaractereSuivant extends Program {

  void algorithm(){
    print("Entrez un caractère : ");
    char c = readChar();

    char suivant = ... ; // <- À COMPLÉTER
    println("Le caractère après " + c + " est " + suivant );
```

```
  }

}
```

1. Générez le squelette à l'aide de l'action dédiée de la commande `ijava`
2. Complétez le code source puis compilez le.
3. Exécutez le programme en l'essayant avec les deux scénarios suivant :

```
Entrez un caractère : a
Le caractère après a est b
```

```
Entrez un caractère : Z
Le caractère après Z est [
```

   Les éléments **en gras** correspondent aux entrées de l'usager au cours de l'exécution du programme.
4. Essayez avec au moins un autre caractère que les deux demandés.
5. Lancez les tests associés à ce programme pour terminer.

Exercice 12 : Des minuscules aux majuscules et vice-versa

On souhaite disposer d'un programme qui transforme une lettre de l'alphabet des minuscules vers les majuscules et réciproquement. N'oubliez pas qu'avec le code ASCII les différents caractères sont représentés par des nombres. Il est donc possible en appliquant un certain décalage sur un caractère (en lui ajoutant un entier) de le transformer en un autre caractère. Pas besoin de connaître la table ASCII pour réaliser un tel exercice, juste de savoir comment elle est strucutrée c'est-à-dire que les lettres majuscules se trouvent avant les minuscules [...,A-Z,...,a-z,...].

Voici le programme que vous avez à compléter :

```
class MajMin extends Program {

  void algorithm(){
    print("Entrez une lettre en minuscule");
    char lettreMin = readChar();

    char enMaj = ; // <- À COMPLÉTER
    println("La lettre " + lettreMin + " en majuscule donne : " + enMaj );

    print("Entrez une lettre en majuscule");
    char lettreMaj = readChar();

    char enMin = ; // <- À COMPLÉTER
    println("La lettre " + lettreMaj + " en minuscule donne : " + enMin );
  }

}
```

1. Générez le squelette puis complétez le code source avant de compiler.
2. Exécutez le programme en réalisant les saisies permettant d'obtenir l'affichage suivant lors de l'exécution :

```
Entrez une minuscule : a
La lettre a en majuscule donne : A
Entrez une majuscule : Z
La lettre Z en minuscule donne : z
```

   (Rappel, ce qui est **en gras** correspond aux entrées de l'usager.
3. Essayez avec au moins une autre minuscule et une autre majuscule.
4. Lancer les tests pour finir et valider cet exercice !

Exercice 13 : Rendre efficacement la monnaie

Sachant que l'on dispose de coupures de 20, 10, 5, 2 et 1 euro, comment procéder pour rendre la monnaie de la manière la plus efficace possible, c'est-à-dire en rendant le moins de coupure possible ?

On supposera que l'on a en entrée un nombre représentant la somme à rendre et que l'on affiche en sortie le nombre minimum de billets de 20 euros, de 10 euros, de 5 euros et de pièces de 2 et de 1 euro à rendre.

ATTENTION : comme nous n'avons pas encore vu cela en cours, il est bien sûr interdit d'utiliser le mot-clé `if` ....

Voici le squelette qu'il vous faut compléter :

```
/**
* Ce programme détermine le nombre minimal de coupures
* à restituer pour une somme donnée. Les coupures utilisables
* sont les billets de 20, 10, 5 et les pièces de 2 et 1 euros.
*
* @author yann.secq@univ-lille.fr
*/
class RenduMonnaie extends Program {

   void algorithm(){
     int somme, nb20, nb10, nb5, nb2, nb1, reste;
     print("Quelle est le montant que vous souhaitez rendre en monnaie ?");
     somme = readInt();
     // à vous de compléter ce qui suit par les calculs permettant de
     // déterminer le nombre minimal de chaque coupure nécessaire.



     // ne pas modifier les lignes ci-dessous, ni les déplacer
     println("Nombre de billets de 20 : " + nb20);
     println("Nombre de billets de 10 : " + nb10);
     println("Nombre de billets de  5 : " + nb5);
     println("Nombre de pièces  de  2 : " + nb2);
     println("Nombre de pièces  de  1 : " + nb1);
   }
}
```

1. Ecrivez le programme permettant de rendre la monnaie en un minimum de coupure. Voici 2 exemples d'exécutions attendues :

```
 Quelle est le montant que vous souhaitez rendre en monnaie ? 4
Nombre de billets de 20 : 0
Nombre de billets de 10 : 0
Nombre de billets de  5 : 0
Nombre de billets de  2 : 2
Nombre de billets de  1 : 0
```

```
 Quelle est le montant que vous souhaitez rendre en monnaie ? 38
Nombre de billets de 20 : 1
Nombre de billets de 10 : 1
Nombre de billets de  5 : 1
Nombre de billets de  2 : 1
Nombre de billets de  1 : 1
```

2. Combien de variables avez-vous utilisé ? Est-il possible d'utiliser moins de variables ? Si c'est le cas, quel est le nombre minimal de variables nécessaire ! :)
3. Essayer votre programme avec au moins 2 autres exemples de votre choix.
4. Tester votre programme à l'aide de la commande dédiée de ijava !

Un peu d'aléatoire...

Dans ces trois exercices, nous aurons besoin de programmes qui se comportent aléatoirement. Autrement dit, des exécutions avec les mêmes entrées pourront produire des résultats différents. Pour ce faire, on va recourir à la fonction `random()` qui renvoie un nombre réel aléatoire entre 0.0 inclus et 1.0 exclus, de type `double`. Par exemple, après l'instruction `double alea = random();`, la variable `alea` contient une valeur aléatoire dans $[0.0, 1.0[$, qui peut aussi bien être `0.34522876893747156` que `0.7983732526723374` ou n'importe quel autre nombre de l'intervalle $[0.0, 1.0[$.

Exercice 14 : Des chiffres...

On souhaite créer le programme `ChiffreAuHasard` qui réalise juste l'affichage d'un nombre à un chiffre (une valeur entre 0 et 9) au hasard. Lors de chaque exécution, on obtient potentiellement n'importe quel chiffre de l'intervalle $[0, 9]$. Il ne faut pas réaliser de saisie et produire juste l'affichage de ce nombre aléatoire.

Sachant que l'appel à `random()` nous renvoie une valeur dans l'intervalle de réels $[0.0, 1.0[$, trouver les opérations à faire pour arriver dans l'intervalle d'entiers $[0, 9]$, programmez ensuite l'algorithme `ijava` correspondant dans un programme nommé `ChiffreAuHasard`.

Exercice 15 : ... et des lettres !

De manière similaire à l'exercice précédent, on souhaite désormais créer un programme `LettreAuHasard` qui affiche une lettre au hasard parmi les majuscules (par exemple pour jouer au mot le plus long en l'appelant autant de fois que de lettres souhaitées).

Exercice 16 : Des dés !

1. Créer un programme `De` simulant un dé à 6 faces. Il affichera une valeur aléatoire entre 1 et 6 à son exécution.
2. Assurez-vous avec une dizaine d'essais que votre programme produit bien toutes valeurs possibles et rien que celles-ci.
3. Créer un nouveau programme `DeMultiface` ce programme de manière à ce qu'il demande à l'utilisateur un dé à combien de faces il veut lancer et affiche le résultat. Pour ce faire, on pourra utiliser la fonction `readInt()` fonctionnant sur le même principe que `readChar()` vue précédemment mais renvoyant un entier, comme son nom l'indique.

Prolongement

Les exercices qui suivent sont proposés pour celles et ceux qui veulent s'entraîner ou aller plus loin.

Exercice 17 : Par ici la monnaie

De retour de vacances passées en partie en Angleterre, Alice et Bob ont dans leur bourse commune 59 livres Sterling et 42 euros. Afin de partager en deux la somme restante, ils souhaitent savoir la valeur totale en euros qu'ils détiennent.

Q1. Écrire le programme `Conversion` dont la fonction principale `algorithm` :

1. stocke dans une variable `bourse` le résultat de la multiplication de 59 par une constante `COURS_LIVRE` égale à 1,09 (car 1 livre sterling = 1.09 euros).
2. lui ajoute 42
3. affiche à l'écran le résultat total puis la part qui revient à chacun.

Q2. Quelques mois plus tard, Alice et Bob partent voir des amis en Suisse et en profitent pour aller en Pologne. Ils se retrouvent désormais avec des francs suisses et des zlotys (la monnaie polonaise). Écrivez un programme qui permet de convertir n'importe quelle somme d'une monnaie en une autre étant donné le taux de conversion entre ces monnaies. Pour saisir une somme et un taux, on pourra faire appel à la fonction `readDouble` qui, à la manière de `readChar`, ne prend aucun paramètre et retourne le double entré par l'utilisateur au clavier (par exemple `0,123`, attention à bien utiliser une virgule).

Exercice 18 : Additionnatrice

Réalisez un programme `Additionnatrice` qui demande à l'utilisateur d'entrer deux réels et en affiche la somme.

Exercice 19 : Heure aléatoire

Réalisez un programme `HeureAleatoire` qui affiche une heure aléatoire au format HH:MM entre 00:00 et 23:59. Dans le cas d'un nombre d'heure ou de minutes inférieur à 10, on pourra omettre les zéros. Par exemple, 8:5 sera accepté pour 08h05

Exercice 20 : Caractère aléatoire dans un intervalle

Réalisez un programme `CaractereAuHasard` qui demande à l'utilisateur d'entrer deux caractères de la table ASCII et en retourne un au hasard entre ceux-ci, le premier inclus et le dernier exclus.

Exercice 21 : Chrono

Sachant que la fonction `long getTime()`, retourne le nombre de millisecondes écoulées depuis le 1er janvier 1970, réalisez un programme `Chrono` qui indique combien de secondes et millisecondes se sont écoulées entre le démarrage du programme et le moment où l'utilisateur entre un caractère.