Laboratoire 5

DURÉE PRÉVUE : 6H00

OBJECTIFS

Au terme de ce laboratoire, l'étudiant sera capable de :

→ Identifier dans un programme les traitements qui nécessitent l'emploi d'une boucle.

→ Utiliser les boucles (**`do-while`**, **`while`** et **`for`**) de manière adaptée au problème donné.

→ Elaborer des expressions booléennes exprimant correctement et clairement la logique de répétition des boucles.

AVANT-PROPOS

1. Dans votre projet **`prb.labos`**, créez un répertoire nommé **`labo5`** (en minuscules et sans espace). C'est dans ce répertoire que vous enregistrerez tous les programmes de ce 4e labo.

2. Afin d'utiliser avec plus de facilité la génération de nombres aléatoires de la fonction **`Math.`***`random`*, vous aurez accès aux fonctionnalités de la classe **`Aleatoire`**.

   Pour ce faire, téléchargez sur *Learn* le fichier **`Aleatoire.java`**.

   Placez ce fichier dans le package **`util`** de votre projet.

EXERCICE 1 : PETITS PROGRAMMES

Ces programmes vont vous permettre de vous familiariser au fonctionnement des instructions **`do-while`**, **`while`** et **`for`**.

LANCÉS DE DÉ

Réalisez un programme qui simule les **lancés successifs d'un dé** jusqu'à obtenir le nombre demandé par l'utilisateur (voir l'exemple d'exécution).

Après chaque lancé, le nombre obtenu est affiché. En fin d'exécution, le nombre total de lancés est également affiché.

Pour simuler les lancés du dé, utilisez une des fonctions **`Aleatoire.`***`aleatoire`*.

EXEMPLE D'EXÉCUTION

```
Nombre de 1 à 6 ? 5
2
6
1
5
Nombre de lancés = 4
```

CONSIGNES

1. Dans le package `labo5` créé précédemment, ajoutez une classe nommée **`De`** comportant une fonction *`main`*.

2. Codez l'entièreté du programme dans la fonction *`main`*.

   [SUGGESTION] Utilisez l'instruction **`do-while`**.

CASINO

Réalisez un programme qui simule les **paris réalisés par un joueur** dans un casino (voir l'exemple d'exécution).

Le joueur indique l'argent dont il dispose à son arrivée au casino (par exemple, 20 EUR) ainsi que les gains qu'il espère remporter (par exemple, 30 € en plus des 20 € de départ). Celui-ci fait des paris de 1€ jusqu'à ce qu'il perde tout son argent (0 €) ou jusqu'à ce qu'il remporte les gains qu'il s'était fixé (par exemple, 30 + 20 = 50 €).

Le joueur a une chance sur deux de gagner à chaque pari. Un pari gagné permet de doubler sa mise. Puisque le joueur ne mise que 1€ à la fois, il reçoit 2€ lorsqu'il gagne.

Pour simuler les paris, utilisez la fonction `Math.random`. Si la valeur obtenue est inférieure à 0,5, alors le joueur réussit son pari. Dans le cas contraire, il perd son pari.

À la fin du programme, affichez le nombre de paris réussis et le nombre de paris effectués.

EXEMPLE D'EXÉCUTION

```
Argent de départ ? 20
Gains souhaités ? 30
Vous avez réussi 177 paris sur 324
Vous repartez avec 30 EUR de plus
```

CONSIGNES

1. Dans le package `labo5`, ajoutez une classe nommée **`Casino`** comportant une fonction *`main`*.
2. Codez l'entièreté du programme dans la fonction *`main`*.

   [SUGGESTION] Utilisez l'instruction **`while`** (aucun pari n'est admis si l'argent de départ est de 0 €) .

PUISSANCES DE 2

Réalisez un programme qui affiche les **puissances de 2**, jusqu'à l'exposant fourni par l'utilisateur, sous la forme d'un tableau (voir l'exemple d'exécution).

Calculez les puissances de 2 sans utiliser la fonction **`Math.`***`pow`* !

EXEMPLE D'EXÉCUTION

```
Exposant maximum ? 10
Exp.   Puissance
   0           1
   1           2
   2           4
   3           8
   4          16
   5          32
   6          64
   7         128
   8         256
   9         512
  10        1024
```

CONSIGNES

1. Dans le package `labo5`, ajoutez une classe nommée **`PuissancesDe2`** comportant une fonction *`main`*.

2. Codez l'entièreté du programme dans la fonction *`main`*.

   [SUGGESTION] Utilisez l'instruction **`for`**.

   REMARQUE

   Il est possible de demander à la fonction *`printf`* d'afficher une valeur en la faisant précéder de plusieurs caractères espace `'␣'` (le symbole `'␣'` permet de représenter plus clairement le caractère espace). Correctement utilisée, cette technique permet d'**aligner verticalement des valeurs**.

   Pour ce faire, il faut spécifier une **largeur exprimée par un nombre entier** dans le format

d'affichage. Par exemple, la largeur 6 spécifiée ci-dessous permet l'affichage de `"␣␣3,14"` :

```
printf("%6.2f", 3.1415927);
```

En effet, la valeur à afficher comporte les 4 caractères `"3,14"` une fois celle-ci arrondie à 2 chiffres après la virgule. La fonction ajoute dès lors 2 caractères espace `"␣␣"` en préfixe pour atteindre la largeur spécifiée.

Cela fonctionne également pour les nombres entiers (par exemple, avec le format `"%9d"`).

3. Savez-vous quelle est la plus grande puissance de 2 que peut afficher votre programme si le type utilisé est **`int`** ?

FIZZ BUZZ

Réalisez un programme qui simule le jeu ***fizz buzz*** (voir l'exemple d'exécution).

Ce jeu est parfois utilisé lors de l'apprentissage de la division chez les enfants. Les enfants s'organisent en cercle, puis l'instituteur désigne celui qui commence le jeu. Ce dernier doit prononcer le nombre 1. Son voisin de gauche prononce ensuite le nombre 2, et ainsi de suite. La particularité de ce jeu est que les nombres divisibles par 3 sont prononcés *fizz* et les nombres divisibles par 5 sont prononcés *buzz*. Les nombres divisibles par 3 et 5 sont prononcés *fizzbuzz*.

Le dernier nombre qui doit être prononcé est fourni au début de l'exécution du programme par l'utilisateur.

EXEMPLE D'EXÉCUTION

```
Dernier nombre ? 30
1, 2, fizz, 4, buzz, fizz, 7, 8, fizz, buzz, 11, fizz, 13, 14, fizzbuzz,
16, 17, fizz, 19, buzz, fizz, 22, 23, fizz, buzz, 26, fizz, 28, 29,
fizzbuzz
```

CONSIGNES

1. Dans le package `labo5`, ajoutez une classe nommée **`FizzBuzz`** comportant une fonction *`main`*.
2. Codez l'entièreté du programme dans la fonction *`main`*.

   [SUGGESTION] Utilisez l'instruction **`for`**.

FACTEURS PREMIERS

Réalisez un programme qui affiche la décomposition d'un entier supérieur ou égal à 2 sous la forme d'un produit de nombres premiers (voir l'exemple d'exécution). Pour exemple, voici les nombres premiers inférieurs à 20 : 2, 3, 5, 7, 11, 13, 17, 19.

Un **nombre premier** est un entier naturel qui admet exactement deux diviseurs distincts, 1 et lui-même.

Par exemple, 126 = 2 × 3 × 3 × 7.

Pour y parvenir, une méthode simple consiste à tester tous les facteurs (même ceux qui ne sont pas premiers) en commençant par 2, c'est-à-dire les facteurs 2, 3, 4, 5, 6 ...

EXEMPLE D'EXÉCUTION

```
Entier >= 2 ? 4312
Décomposition de 4312 en produit de facteurs premiers = 2 2 2 7 7 11
```

CONSIGNES

1. Dans le package `labo5`, ajoutez une classe nommée **`FacteursPremiers`** comportant une fonction *`main`*.
2. Codez l'entièreté du programme dans la fonction *`main`*.

   [SUGGESTION] Utilisez l'instruction **`while`**.

FIZZ BUZZ - VARIANTE **[FACULTATIF]**

Réalisez un programme qui simule une **variante du jeu fizz buzz** (voir l'exemple d'exécution).

Dans cette variante, les nombres contenant un chiffre 3 sont prononcés *fizz*, les nombres contenant un chiffre 5 sont prononcés *buzz* et les nombres contenant ces deux chiffres sont prononcés *fizzbuzz*.

CONSIGNES

1. Dans le package `labo5`, ajoutez une classe nommée **`FizzBuzzVariante`** comportant une fonction *`main`*.

   Voici deux méthodes permettant de déterminer si un nombre contient un chiffre précis :

   → transformez le nombre en une chaîne de caractères avec la méthode **`String.`***`valueOf`*, puis utiliser la fonction *`indexOf`* ou *`contains`* de la même classe.

   → déclarez une fonction nommée **`contientChiffre`** qui indique si un nombre contient un chiffre spécifique :

   ```
   public static boolean contientChiffre(int nombre, int chiffre)
   ```

   Utilisez la **division euclidienne**. En divisant le nombre par 10, le reste obtenu est le dernier chiffre du nombre et le quotient obtenu est le nombre amputé du dernier chiffre. Par exemple, `128 % 10` donne `8` et `128 / 10` donne `12`. Combiné à cela, utilisez une instruction **`while`** afin de permettre l'obtention de chaque chiffre.

   > **IMPORTANT**
   >
   > Testez cette fonction dans une classe nommée **`FizzBuzzVarianteTest`** à l'aide de *JUnit*.

2. Codez ensuite le reste du programme dans la fonction *`main`*.

EXEMPLE D'EXÉCUTION

```
Dernier nombre ? 35

1, 2, fizz, 4, buzz, 6, 7, 8, 9, 10, 11, 12, fizz, 14, buzz, 16, 17, 18,
19, 20, 21, 22, fizz, 24, buzz, 26, 27, 28, 29, fizz, fizz, fizz, fizz,
fizz, fizzbuzz
```

EXERCICE 2 : CHATBOT

DESCRIPTION DU PROGRAMME

Nous allons créer un programme qui simule une discussion avec un **chatbot**. Les capacités de réaction du chatbot seront évidemment simplifiées. Si vous le souhaitez, vous pourrez facilement les étendre.

EXEMPLE D'EXÉCUTION

Vous devez respecter la mise en forme proposée dans l'exemple ci-dessous. Les **données saisies par l'utilisateur** sont indiquées en bleu.

```
CHATBOT : Bonjour ! Je suis un chatbot. Comment puis-je vous aider aujourd'hui ?
VOUS : Comment allez-vous ?
CHATBOT : Je suis un logiciel, donc je ne ressens rien, mais merci de demander.
VOUS : Pourriez-vous m'en dire plus sur le sens de la vie ?
CHATBOT : Le sens de la vie est un mystère, mais certains disent que c'est 42.
VOUS : Merci pour votre réponse. Au revoir !
CHATBOT : Au revoir !
```

CONSIGNES

1. Afin de vous faciliter la tâche, les capacités de réaction de base du chatbot ont déjà été programmées dans fonction nommée *choisirReponse*. Pour la récupérer, suivez la procédure ci-après.

   Téléchargez sur *Learn* le fichier **Chatbot.java**, puis placez ce fichier dans le package **labo5** de votre projet. La classe contient une fonction *main* vide et la fonction *choisirReponse*.

2. Codez l'entièreté du programme dans la fonction *main*. Choisissez l'instruction de boucle qui vous semble la plus appropriée. Cette dernière doit se répéter tant que la réponse du chatbot ne comporte pas la séquence de mots « **au revoir** ».

   REMARQUE

Pour vérifier la présence des mots « au revoir », aidez-vous des fonctions *toLowerCase* et *contains* de la classe **String**.

EXERCICE 3 : LIVRE DONT VOUS ÊTES LE HÉROS

DESCRIPTION DU PROGRAMME

Nous allons cette fois créer un programme qui permet de lire une histoire à la manière des **livres dont vous êtes le héros** (en abrégé, LDVELH). Ces livres proposent des histoires interactives, leur déroulement dépendant des choix du lecteur.

Ainsi, à certains moments-clés de l'histoire, le lecteur est confronté à différents choix représentant les actions que peut réaliser le personnage qu'il incarne. Chaque choix mène à une suite spécifique de l'histoire décrivant les conséquences du choix du lecteur.

EXEMPLE D'EXÉCUTION

Vous devez respecter la mise en forme proposée dans l'exemple ci-dessous. Les **données saisies par l'utilisateur** sont indiquées en bleu.

```
1er mars 1820, dans les mers territoriales françaises. Prisonnier depuis
des années, tu croupis dans un cachot, au cœur d'une forteresse
surmontant un îlot rocheux. Dans cette cellule sans fenêtre, le silence
n'est interrompu que par les pas réguliers du geôlier pendant ses
rondes. Bien que tu aies perdu toute notion du temps, tu sais que ta
libération est encore lointaine. Mais au bout du désespoir, une idée
germe en toi : tu dois t'évader.

Une fois de plus, tu tâtes chaque pierre de ta geôle, cherchant
désespérément une issue. Soudain, tu perçois un léger souffle d'air
s'échappant d'une fissure. Les joints de la pierre semblent fragiles.
Quelques coups frappés avec un caillou suffisent à la détacher. Derrière
elle, une cavité se dévoile dans la muraille.

Alors que tu t'engages dans l'obscurité, une bouffée d'adrénaline te
propulse en avant. Chaque mouvement est un mélange d'excitation et de
crainte. Après une interminable progression dans les ténèbres, tu
débouches sur un réseau de tunnels souterrains.

Choix A : Explorer le souterrain à la recherche d'une sortie.
```

```
Choix B : Creuser en ligne droite jusqu'à la mer.

Que choisis-tu (A ou B) ? A

Après des jours d'exploration, tu découvres une pièce creusée dans la
roche, ouverte sur la mer. D'un côté, des étagères chargées de
provisions, de l'autre, une vieille barque amarrée à un ponton. Mais
tout à coup, des bruits de pas se font entendre, se rapprochant
inexorablement.

Choix A : Prendre des provisions et fuir par le bateau.

Choix B : Attendre et observer pour voir qui arrive.

Que choisis-tu (A ou B) ? A

Tu saisis quelques provisions d'un geste rapide et te diriges vers la
vielle barque. Lorsque tu manies les rames, une douleur sourde te saisit
; tes muscles, endoloris par des années de captivité, peinent à réagir.
Pourtant, l'exaltation de la liberté retrouvée te donne l'énergie
nécessaire. En quelques coups de rames, tu t'éloignes déjà de la
forteresse.

En mer, une tempête se lève soudainement, et les vagues déchaînées
renversent ton embarcation. Tu luttes désespérément pour rester à flot,
mais les vagues t'engloutissent peu à peu. Epuisé, tu te laisses
submerger, et c'est dans les eaux tumultueuses que ta quête de liberté
trouve une fin tragique.

FIN
```

CONSIGNES

1. Toutes les parties de l'histoire ont été retranscrites dans un fichier. Chaque partie est associée à la séquence des choix y menant (par exemple, la **clé** **`"choix-a-a"`** permet d'accéder à la **valeur** *« Tu saisis quelques provisions d'un geste rapide… »*). Pour utiliser ce fichier dans votre programme, suivez la procédure ci-après.

   Téléchargez sur *Learn* le fichier **`ldvelh.txt`**, puis placez ce fichier dans le dossier `data` de votre projet.

2. Dans le package `labo5`, ajoutez une classe nommée **LDVELH** comportant une fonction *main*.

3. Dans la classe **LDVELH**, déclarez une fonction nommée *lireChoix* capable d'effectuer l'acquisition d'un caractère valide :

```
public static char lireChoix(String question, String choixAdmis)
```

Le paramètre `question` est la question à afficher à l'utilisateur au moment de l'acquisition.

Le paramètre `choixAdmis` indique tous les caractères admis comme réponse valide.

La valeur saisie est récupérée sous la forme d'une chaîne de caractères. La chaîne saisie est alors considérée comme valide seulement si elle ne comporte qu'un seul caractère qui correspond à l'un de ceux spécifiés par le paramètre `choixAdmis`.

Dans le cas où la saisie n'est pas valide, le message *« Choix incorrect ! Veuillez recommencer... »* est affiché et l'acquisition doit recommencer.

Voici un exemple d'exécution de cette fonction avec les chaînes **"Que choisis-tu (A ou B) ? "** et **"AaBb"** spécifiée respectivement pour les paramètres `question` et `choixAdmis` (le symbole ↲ représente ici la pression de la touche ***Entrée***) :

```
Que choisis-tu (A ou B) ? C↲
Choix incorrect ! Veuillez recommencer...
Que choisis-tu (A ou B) ? ↲
Choix incorrect ! Veuillez recommencer...
Que choisis-tu (A ou B) ? a
```

Dans ce cas, le caractère retourné est **'a'**. En effet, la casse de la réponse doit être conservée !

IMPORTANT

N'utilisez **pas d'expression régulière** dans cette fonction !

4. Malgré la présence de l'acquisition de données dans la fonction *lireChoix*, il est possible d'effectuer des tests automatisés sur cette dernière à l'aide de *JUnit*. Pour ce faire, il faut utiliser les fonctions *simulerSaisies* et *nettoyerSaisies* de la classe **Console**.

Vous pouvez reproduire le code ci-dessous dans une classe nommée **LDVELHTest**, puis le compléter :

```
@BeforeEach
void nettoyerSaisiesAvantChaqueTest() {
	Console.nettoyerSaisies();
}

@AfterAll
static void nettoyerSaisiesFinDesTests() {
	Console.nettoyerSaisies();
}

@Test
void lireChoix_saisieValide() {
	Console.simulerSaisies("A");
	assertEquals('A', LDVELH.lireChoix("A ou B ? ", "AaBb"));
}

@Test
void lireChoix_saisieNonValide() {
	Console.simulerSaisies("C", "", "a");
	assertEquals('a', LDVELH.lireChoix("A ou B ? ", "AaBb"));
}
```

5. Codez ensuite le reste du programme dans la fonction *main*. Pour récupérer le texte d'introduction, spécifiez la clé **"intro"**. Choisissez l'instruction de boucle qui vous semble la plus appropriée. Cette dernière doit se répéter tant que le texte obtenu ne se termine pas par la chaîne **"FIN\n"**.

> **REMARQUES**
>
> Pour **vérifier la terminaison du texte** par la chaîne **"FIN\n"**, aidez-vous de la fonction *endsWith* de la classe **String**.
>
> Comme cela avait déjà été le cas dans le programme 1 du labo 4 (« Authentification »), vous devez utiliser la fonction **Fichier.***lireString* en spécifiant la clé appropriée pour **récupérer le texte** correspondant du fichier **ldvelh.txt**.
>
> Une clé doit être formée par des concaténations successives sur base des choix effectués. Par exemple, la clé **"choix-a-a"** est formée de la chaîne **"choix"** augmentée de **"-a"** du fait du 1er choix effectué, puis de **"-a"** du fait du 2e choix.

EXERCICE 4 [À FAIRE À DOMICILE] : LES BOUCLES ET LES CHAÎNES DE CARACTÈRES

Pour vous familiariser davantage avec le fonctionnement des instructions de boucle, nous allons à nouveau utiliser l'outil *CodingBat*.

PREMIER EXERCICE

Rendez-vous sur la page *http://codingbat.com/java/String-2*.

Cliquez ensuite sur le premier exercice nommé **doubleChar**.

Voici une traduction de l'énoncé : *« Étant donnée une chaîne, retournez une nouvelle chaîne où tous les caractères de la chaîne reçue sont doublés, par exemple "The" produit la chaîne "TThhee" »*.

La déclaration de fonction à compléter est :

```
public String doubleChar(String str) {

}
```

Une fois votre code prêt, cliquez sur le bouton ***Go*** pour le tester. Répétez l'opération jusqu'à ce que tous les tests soient au vert !

CONSIGNES

Réalisez les 3 exercices suivants :

1. **countHi** : *« Étant donnée une chaîne, retournez le nombre d'apparitions de la chaîne "hi" (en tenant compte de la casse) dans la chaîne fournie, par exemple "Hi Bill, hi there!" contient une apparition de "hi" »*.
2. **catDog** : *« Étant donnée une chaîne, retournez vrai si les chaînes "cat" et "dog" apparaissent le même nombre de fois dans la chaîne fournie, par exemple le résultat est vrai pour la chaîne "A cat and two dogs" car elle contient une apparition de chacune »*.
3. **countCode** : *« Étant donnée une chaîne, retournez le nombre d'apparitions de la chaîne "code" (en tenant compte de la casse) dans la chaîne fournie, à la différence qu'ici n'importe quelle lettre peut remplacer le 'd', par exemple le résultat est 2 pour la chaîne "code and core" »*.

EXERCICE 5 : MATCH DE TENNIS

DESCRIPTION DU PROGRAMME

Les organisateurs d'un tournoi de tennis souhaitent un programme permettant de simplifier **le comptage et l'affichage des points** lors d'un match (voir l'exemple d'exécution).

À chaque point gagné par l'un des joueurs, l'arbitre doit encoder le numéro du joueur qui vient de marquer le point. Les nouveaux scores sont ensuite affichés. En tennis, les points d'un jeu se comptent de la manière suivante : 0, 15, 30, 40, jeu.

Pour **remporter un jeu**, il faut marquer 4 points et avoir au minimum une avance de deux points sur son adversaire. Par exemple, si le score est de 40 - 30 et que le joueur 1 marque le point suivant, alors ce dernier remporte le jeu.

En cas d'**égalité 40 - 40**, chaque joueur doit marquer deux points consécutifs pour remporter le jeu. A partir de ce moment, le joueur qui possède un point d'avance obtient l'avantage, noté « A » au tableau des scores (par exemple, A - 40). En cas de nouvelle égalité, les scores sont ramenés à 40 - 40.

Dans cette première version, le comptage des jeux et des sets ne doit pas être réalisé.

EXEMPLE D'EXÉCUTION

Vous devez respecter la mise en forme proposée dans l'exemple ci-dessous. Les **données saisies par l'utilisateur** sont indiquées en bleu.

```
Joueur 1 ? Goffin D.
Joueur 2 ? Mayer F.

Goffin D.         0
Mayer F.          0

Quel joueur gagne le point (1 ou 2) ? 1
Goffin D.        15
Mayer F.          0

Quel joueur gagne le point (1 ou 2) ? 2
Goffin D.        15
Mayer F.         15
```

```
Quel joueur gagne le point (1 ou 2) ? 2
Goffin D.       15
Mayer F.        30

Quel joueur gagne le point (1 ou 2) ? 1
Goffin D.       30
Mayer F.        30

Quel joueur gagne le point (1 ou 2) ? 1
Goffin D.       40
Mayer F.        30

Quel joueur gagne le point (1 ou 2) ? 2
Goffin D.       40
Mayer F.        40

Quel joueur gagne le point (1 ou 2) ? 1
Goffin D.        A
Mayer F.        40

Quel joueur gagne le point (1 ou 2) ? 1

Goffin D. remporte le jeu !
```

CONSIGNES

1. Dans le package `labo5`, ajoutez une classe nommée **`MatchDeTennis`** comportant une fonction *`main`*.

2. Il est plus simple de comptabiliser les points par incréments de 1 (0, 1, 2, 3, 4) plutôt que de respecter l'usage tennistique (0, 15, 30, 40 et A). Pour l'affichage des scores, il suffit dès lors d'effectuer une correspondance : 0 donne 0, 1 donne 15, 2 donne 30…

   Pour ce faire, déclarez une fonction nommée **`getScore`** qui retourne le score correspondant à un nombre de points donné sous la forme d'une chaîne de caractères :

   ```
   public static String getScore(int points)
   ```

   Voici les valeurs retournées par la fonction : **`" 0"`** pour 0 point, **`"15"`** pour 1 point, **`"30"`** pour 2 points, **`"40"`** pour 3 points, **`" A"`** pour 4 points et **`" ?"`** pour toute autre valeur.

> **IMPORTANT**
>
> Testez cette fonction dans une classe nommée **`MatchDeTennisTest`** à l'aide de *JUnit*.

3. Codez ensuite le reste du programme dans la fonction *`main`*. Pour calculer l'écart de points entre les deux joueurs, vous pouvez utiliser la fonction **`Math.`***`abs`*.

EXERCICE 6 [FACULTATIF] : MATCH DE TENNIS - AMÉLIORATION

DESCRIPTION DU PROGRAMME

Améliorez le programme précédent en réalisant le **comptage des jeux et des sets** jusqu'à la victoire de l'un des joueurs (voir l'exemple d'exécution).

Les règles du tournoi sont les suivantes :

→ le premier joueur qui remporte deux sets gagne le match.

→ le premier joueur qui remporte six ou sept jeux, avec au moins deux jeux d'écart, gagne le set (6 - 0, 6 - 1, 6 - 2, 6 - 3, 6 - 4 ou 7 - 5).

→ si au bout de douze jeux les joueurs sont à égalité (6 - 6), alors c'est le joueur qui remporte le jeu suivant qui gagne le set (7 - 6).

[REMARQUE] Il n'y a pas de jeu décisif (*tie-break*) dans ce tournoi.

EXEMPLE D'EXÉCUTION

Vous devez respecter la mise en forme proposée dans l'exemple ci-dessous. Les **données saisies par l'utilisateur** sont indiquées en bleu. Dans l'**ordre des colonnes**, voici les informations affichées : le nombre de sets remportés, le nombre de jeux remportés et les points remportés.

```
Joueur 1 ? Goffin D.
Joueur 2 ? Mayer F.

Goffin D.       0 | 0 |  0
Mayer F.        0 | 0 |  0

Quel joueur gagne le point (1 ou 2) ? 1
```

```
Goffin D.       0 | 0 | 15
Mayer F.        0 | 0 |  0

Quel joueur gagne le point (1 ou 2) ? 1
...

Quel joueur gagne le point (1 ou 2) ? 2
Goffin D.       1 | 5 | 40
Mayer F.        0 | 4 | 30

Quel joueur gagne le point (1 ou 2) ? 1
Goffin D.       2 | 0 |  0
Mayer F.        0 | 0 |  0

Goffin D. remporte le match !
```

CONSIGNES

1. Créez une copie du fichier **`MatchDeTennis.java`** et renommez-la en **`MatchDeTennisComplet.java`**.

   N'oubliez pas de renommer en conséquence la classe dans le fichier.

2. **[SUGGESTION]** Cette amélioration peut s'effectuer de deux manières différentes :

   → ajouter de nouvelles boucles en les imbriquant avec la boucle existante.

   → modifier la boucle existante, ce qui implique l'ajout à l'intérieur de cette boucle de plusieurs instructions **`if`** permettant de déterminer si un jeu ou un set est remporté par l'un des joueurs afin d'adapter les scores en conséquence.

   La deuxième approche est la plus simple à mettre en œuvre.

EXERCICE 7 [FACULTATIF] : JEU DU PENDU

DESCRIPTION DU PROGRAMME

Réalisez un jeu du pendu. Pour rappel, ce jeu consiste à trouver un mot en devinant les lettres qui le constituent (voir l'exemple d'exécution).

Au début de la partie, le mot à trouver est représenté par une succession de traits, un trait par lettre.

Le joueur propose ensuite des lettres, une par une. Si une lettre proposée est présente dans le mot, cette lettre est affichée aux positions correspondantes. Dans le cas contraire, l'erreur est comptabilisée en ajoutant un élément à la potence.

À la septième erreur, le joueur perd la partie.

EXEMPLE D'EXÉCUTION

Vous devez respecter la mise en forme proposée dans l'exemple ci-dessous. Les **données saisies par l'utilisateur** sont indiquées en bleu.

```
 ______
 |
 |
 |
 |
_|_
Mot caché      = -------
Lettres jouées =

Une lettre ? E
 ______
 |
 |
 |
 |
_|_
Mot caché      = ---E--
Lettres jouées = E

Une lettre ? A
```

```
 ______
 |
 |
 |
 |
_|_
Mot caché      = -A--EA-
Lettres jouées = EA

Une lettre ? S
 ______
 |    |
 |
 |
 |
_|_
Mot caché      = -A--EA-
Lettres jouées = EAS

Une lettre ? R
 ______
 |    |
 |    O
 |
 |
_|_
Mot caché      = -A--EA-
Lettres jouées = EASR

Une lettre ? ...
```

CONSIGNES

1. Afin de rendre le jeu plus intéressant, le mot à trouver sera choisi aléatoirement parmi 835 mots enregistrés dans un fichier. Pour utiliser ce fichier dans votre programme, suivez la procédure ci-après.

   Téléchargez sur *Learn* le fichier **`dictionnaire.txt`**, puis placez ce fichier dans le dossier `data` de votre projet.

2. Ensuite, pour faciliter l'obtention d'un mot depuis le fichier **`dictionnaire.txt`**, nous allons utiliser la classe **`Dictionnaire`**.

   Téléchargez sur *Learn* le fichier **`Dictionnaire.java`**, puis placez-le dans le package `labo5`.

   En combinant les deux fonctions de la classe **`Dictionnaire`** avec une des fonctions `util.`**`Aleatoire.`***`aleatoire`*, vous pourrez choisir de manière aléatoire un mot en début de partie.

3. Dans le package `labo5`, ajoutez une classe nommée **`JeuDuPendu`** comportant une fonction *`main`*.

4. Codez le programme dans la fonction *`main`* en vous aidant des deux fonctions suggérées au point 8. Pour obtenir le mot caché initial ne comportant que des tirets, aidez-vous de la fonction *`repeat`* de la classe **`String`**.

   Pour vous simplifier la tâche, ne codez dans un premier temps que les **fonctionnalités essentielles**. Ensuite, vous pourrez ajouter des fonctionnalités secondaires proposées ci-dessous.

   [AMÉLIORATION 1] La mémorisation des **lettres jouées** peut se faire simplement grâce à une chaîne de caractères à laquelle chaque nouvelle lettre est ajoutée par concaténation.

   [AMÉLIORATION 2] La saisie de l'utilisateur doit correspondre exactement à une lettre majuscule non accentuée n'ayant pas été jouée précédemment. En cas de **saisie incorrecte**, répétez l'acquisition. Pensez à utiliser les fonctions *`indexOf`*, *`matches`* et *`toUpperCase`* de la classe **`String`**.

   [AMÉLIORATION 3] En début de partie, demandez à l'utilisateur de saisir son nom afin d'**établir des statistiques** sur ses parties jouées (par exemple : nombre de parties jouées, nombre de parties gagnées, numéro de classement par rapport aux autres joueurs…). Pour ce faire, aidez-vous des fonctions *`ecrireValeur`*, *`lireString`*, *`lireInt`* et *`lireDouble`* de la classe **`Fichier`**.

5. Dans la classe **JeuDuPendu**, déclarez :

   a. une fonction nommée **genererPotence** qui permet de créer une chaîne représentant la potence selon le nombre d'erreurs du joueur :

   ```
   public static String genererPotence(int nbErreurs)
   ```

   Voici un modèle de potence complète :

   ```
    _____
    |    |
    |    O
    |   /|\
    |   / \
   _|_
   ```

   b. une fonction nommée **revelerLettre** qui permet d'obtenir le nouveau mot caché après avoir révélé une lettre apparaissant dans le mot à trouver :

   ```
   public static String revelerLettre(String motCache, String motATrouver, char lettre)
   ```

   Par exemple, si les arguments de l'appel de fonction sont respectivement **"----E--"**, **"TABLEAU"** et **'A'**, alors le résultat retourné est **"-A--EA-"**.

> **IMPORTANT**
>
> Testez vos fonctions dans une classe nommée **JeuDuPenduTest** à l'aide de *JUnit*.