Laboratoire 4

DURÉE PRÉVUE : 8H00

OBJECTIFS

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

→ Utiliser les alternatives (`if` et `switch`) de manière adaptée au problème donné.

→ Construire des expressions booléennes en s'aidant des opérateurs relationnels et logiques.

→ Contrôler la couverture des tests unitaires afin de s'assurer que tous les cas de figure ont été envisagés.

AVANT-PROPOS

Dans votre projet `prb.labos`, créez un répertoire nommé `labo4` (en minuscules et sans espace). C'est dans ce répertoire que vous enregistrerez tous les programmes de ce 4e labo.

EXERCICE 1 : AUTHENTIFICATION

DESCRIPTION DE LA 1ÈRE VERSION DU PROGRAMME

Réalisez un programme qui permet à une personne de s'authentifier sur base d'un nom d'utilisateur et d'un mot de passe, tous deux saisis par cet utilisateur. Pour cette 1ère version, le programme doit se contenter de vérifier que les nom d'utilisateur et mot de passe saisis correspondent respectivement à « **Thor** » (en ne tenant pas compte de la casse) et « **marteau83** » (en tenant compte de la casse).

EXEMPLES D'EXÉCUTION

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

```
Nom utilisateur ? Thor
Mot de passe ? marteau83
Bienvenue Thor ! Vous êtes maintenant connecté.
```

```
Nom utilisateur ? Thor
Mot de passe ? marteau
Données incorrectes ! Veuillez réessayer...
```

CONSIGNES

1. Dans le package `labo4` créé précédemment, ajoutez une classe nommée **`Authentification`** comportant une fonction `main`.
2. Codez l'ensemble de l'application dans la fonction `main` à l'aide d'une instruction **`if`** afin de gérer les deux cas de figure (connexion réussie ou échouée).

> **IMPORTANT**
>
> N'oubliez pas qu'il faut utiliser les fonctions *`equals`* et *`equalsIgnoreCase`* de la classe **`String`** pour comparer deux chaînes de caractères, et non l'opérateur relationnel `==` !
>
> Si nécessaire, aidez-vous de la documentation officielle à ce sujet :
> *https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/String.html*

DESCRIPTION DE LA 2E VERSION DU PROGRAMME

Modifiez votre programme pour qu'il consulte dorénavant les données relatives aux comptes utilisateur dans un fichier « .txt » afin de déterminer si le nom d'utilisateur saisi existe et, le cas échéant, si le mot de passe saisi correspond au mot de passe extrait du fichier.

FICHIER DES COMPTES UTILISATEUR

Chaque ligne du fichier comporte les informations relatives à un compte utilisateur, à savoir son **nom d'utilisateur** suivi de son **mot de passe**, séparées par un symbole égal **=**.

Voici le contenu de ce fichier :

```
thor=marteau83
iron man=tony39
captain america=shield11
loki=odin85
black panther=wakanda66
thanos=titan55
black widow=ivan52
hulk=gamma5
```

Pour ajouter ce fichier à votre projet *Eclipse*, suivez la procédure suivante :

1. Téléchargez sur *Learn* le fichier **`authentification.txt`**.
2. Ce fichier n'étant pas un fichier source, nous allons le placer dans un dossier à part.

   Dans le volet *Package Explorer*, faites un clic droit sur le projet **`prb.labos`** pour faire apparaître le menu contextuel, puis sélectionnez ***New*** > ***Folder***. Nommez ce nouveau dossier **`data`**.
3. Placez le fichier **`authentification.txt`** dans le dossier **`data`**.

LA CLASSE FICHIER

Afin de vous faciliter l'accès au contenu du fichier **`authentification.txt`**, vous utiliserez la classe **`Fichier`** disponible sur *Learn*. Cette dernière propose, entre autres, la fonction *`lireString`* pour récupérer sous la forme de chaînes de caractères des informations contenues dans un fichier.

Téléchargez sur *Learn* le fichier **`Fichier.java`**.

Pour utiliser cette dernière dans votre programme, placez le fichier **`Fichier.java`** dans le package **`io`** de votre projet.

Cette fonction prend deux arguments :

1. Le **chemin d'accès au fichier**, dans ce cas `"data\\authentification.txt"` (ou, de manière équivalente, `"data/authentification.txt"`).

   > **REMARQUE**
   >
   > Pour référencer ce chemin d'accès, il est recommandé de créer une constante.

2. La **clé**, ici le nom d'utilisateur, permettant d'accéder à la **valeur associée**, ici le mot de passe.

   > **REMARQUES**
   >
   > Les fonctions *`ecrireValeur`*, *`contientCle`*, *`lireString`*, *`lireInt`* et *`lireDouble`* de la classe **`Fichier`**, permettent d'utiliser un fichier à la manière d'un **tableau associatif** (aussi appelé dictionnaire ou Map).
   >
   > Afin de **ne pas tenir compte de la casse pour la clé** (cette dernière étant le nom d'utilisateur saisi en console par l'utilisateur), utilisez la fonction *`toLowerCase`* de la classe **`String`**.
   >
   > La fonction *`trim`* de la classe **`String`** peut également s'avérer utile afin d'**ignorer les caractères blancs** (situés en début et en fin de chaîne) maladroitement saisis par l'utilisateur lorsque ce dernier doit spécifier son nom d'utilisateur.

CONSIGNES

Faites en sorte que le programme gère dorénavant 3 cas de figure au lieu de 2, à savoir : connexion réussie, nom d'utilisateur inconnu et mot de passe incorrect.

```
Nom utilisateur ? Tor
Mot de passe ? marteau83
Nom d'utilisateur inconnu ! Veuillez réessayer...
```

```
Nom utilisateur ? Thor
Mot de passe ? marteau
Mot de passe incorrect ! Veuillez réessayer...
```

REMARQUE

Essayez de structurer les instructions **`if`** sous la forme « **`if ... else if ... else ...`** ».

EXERCICE 2 : ANNIVERSAIRE

DESCRIPTION DU PROGRAMME

Réalisez un programme qui permet de déterminer si l'anniversaire d'une personne est proche (de 31 jours ou moins) de la date actuelle.

EXEMPLES D'EXÉCUTION

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

En supposant que nous sommes le 6 novembre :

```
La date de votre anniversaire (j/m) ? 2/12
Super ! C'est bientôt votre anniversaire.
```

```
La date de votre anniversaire (j/m) ? 21/12
Dommage ! Votre anniversaire est encore loin.
```

En supposant que nous sommes le 15 décembre :

```
La date de votre anniversaire (j/m) ? 15/1
Super ! C'est bientôt votre anniversaire.
```

LA CLASSE DATE

Téléchargez sur *Learn* le fichier **`Date.java`**.

Afin de vous donner accès à quelques opérations de base relatives aux dates, la classe **`Date`** propose, entre autres, les fonctions *`numeroJour()`*, *`numeroJour(int, int, int)`* et *`anneeAuj`*.

Pour utiliser ces dernières dans votre programme, placez le fichier **`Date.java`** dans un package nommé **`util`** de votre projet.

CONSIGNES

1. Dans le package `labo4`, ajoutez une classe nommée **`Anniversaire`** comportant une fonction `main`.
2. Codez l'ensemble de l'application dans la fonction `main` à l'aide d'une (ou plusieurs) instruction(s) **`if`** afin de gérer les deux cas de figure (anniversaire proche ou lointain).

   IMPORTANT

   Il est recommandé de réaliser une phase préparatoire pour le **calcul du nombre de jours restants** avant la date de l'anniversaire.

   Ne perdez pas de vue que vous devrez utiliser les fonctions *`numeroJour()`* et *`numeroJour(int, int, int)`* pour y parvenir. Veillez donc à lire attentivement leurs documentations au préalable !
3. Pour réaliser vos tests, spécifiez vous-même dans le code de votre programme la date actuelle (au lieu d'utiliser la date réelle).

VERSION AMÉLIORÉE

Faites en sorte que le programme gère dorénavant 3 cas de figure au lieu de 2, à savoir : c'est votre anniversaire, anniversaire proche et anniversaire lointain.

En supposant que nous sommes le 6 novembre :

```
La date de votre anniversaire (j/m) ? 6/11
Joyeux anniversaire !
```

REMARQUE

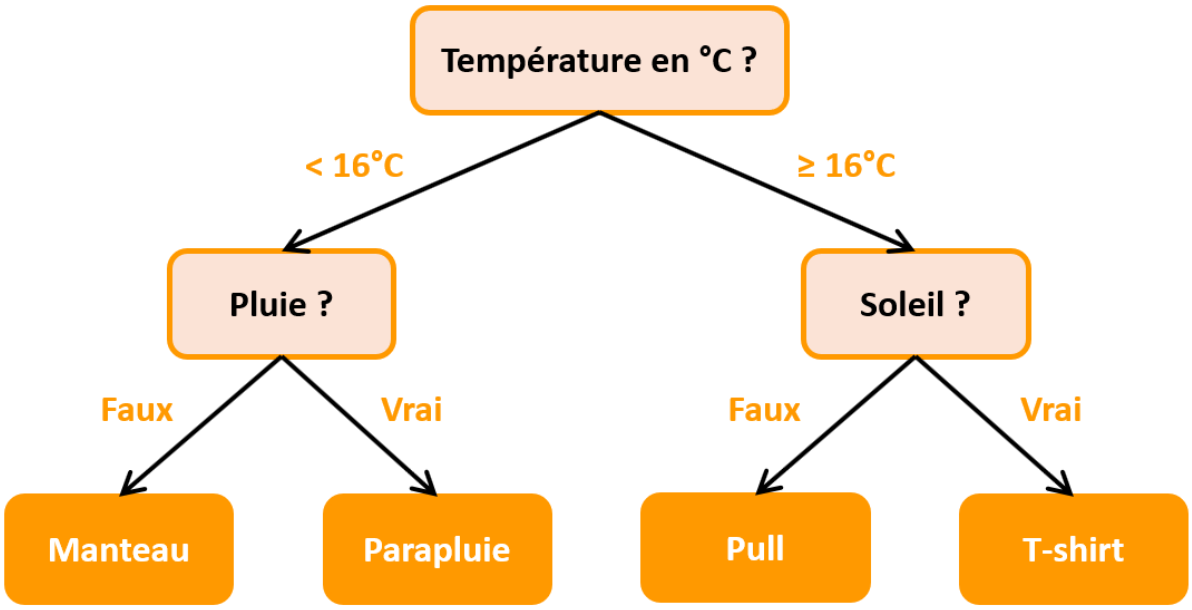
Essayez de structurer les instructions **`if`** sous la forme « **`if ... else if ... else ...`** ».

EXERCICE 3 : MÉTÉO

DESCRIPTION DU PROGRAMME

Réalisez un programme qui permet de conseiller une personne sur la manière de se préparer avant de sortir de chez elle en fonction de la météo.

Les critères de décision sont représentés dans le schéma ci-dessous :



EXEMPLES D'EXÉCUTION

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

```
Température en °C ? 12
Est-il en train de pleuvoir (o/n) ? o
Vous devriez prendre un parapluie.
```

```
Température en °C ? 17
Fait-il ensoleillé (o/n) ? n
Vous devriez porter un pull.
```

CONSIGNES

1. Dans le package `labo4`, ajoutez une classe nommée **`Meteo`** comportant une fonction `main`.

2. Codez l'ensemble de l'application dans la fonction `main` à l'aide d'instructions **`if`** et **`switch`** afin de gérer les quatre cas de figure (Porter un manteau, prendre un parapluie, porter un pull ou porter un t-shirt).

> **REMARQUE**
>
> En cas de réponse binaire attendue (« oui » ou « non »), l'utilisateur **peut ne saisir qu'un seul caractère**. Par exemple, à la question *"Y a-t-il de la pluie : (O)ui/(N)on ?"*, l'utilisateur peut répondre une lettre `'o'` ou `'n'`, minuscule ou majuscule.
>
> Pour **lire la saisie**, nous avons le choix entre récupérer une chaîne ou un seul caractère. Afin de simplifier le codage, il est recommandé de choisir la seconde solution en utilisant la fonction **`Console.`***`lireChar`*.
>
> Pour **tester la valeur du caractère saisi**, essayez d'utiliser l'instruction **`switch`**.

VERSION AMÉLIORÉE

Faites en sorte que le programme gère une saisie invalide pour la température (non comprise entre -30°C et 40°C) et pour une réponse binaire (différente des lettres `'o'` ou `'n'`, minuscule ou majuscule). Le cas échéant, un message d'erreur doit être affiché.

> **REMARQUE**
>
> **Quand la valeur d'un caractère est testée**, complétez l'instruction **`switch`** correspondante par une clause **`default`**.

```
Température en °C ? 120
Cette température est incorrecte !
```

```
Température en °C ? 17
Fait-il ensoleillé (o/n) ? b
Cette réponse est incorrecte !
```

EXERCICE 4 [À FAIRE À DOMICILE] : LES BASES AVEC CODINGBAT

Pour vous familiariser davantage avec le fonctionnement de l'instruction **`if`**, des opérateurs relationnels et des opérateurs logiques, nous allons à nouveau utiliser l'outil *CodingBat*.

PREMIER EXERCICE

Rendez-vous sur la page *http://codingbat.com/java/Warmup-1*.

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

Voici une traduction de l'énoncé : *« Le paramètre weekday est vrai si c'est un jour de semaine, et le paramètre vacation est vrai si c'est un jour de vacances. Nous faisons la grasse matinée si ce n'est pas un jour de semaine ou si c'est un jour de vacances. Retournez vrai si nous faisons la grasse matinée »*.

La déclaration de fonction à compléter est :

```
public boolean sleepIn(boolean weekday, boolean vacation) {

}
```

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 !

IMPORTANT

Voici quelques recommandations lors de la réalisation de cet exercice et des suivants :

- → Réduisez le nombre d'instructions **`if`** en regroupant les expressions booléennes à l'aide d'opérateurs logiques.
- → Essayez de n'utilisez qu'une seule instruction **`return`** par fonction. Pour ce faire, déclarez une variable supplémentaire pour stocker le résultat à retourner à la fin du corps de la fonction.
- → Pour chaque exercice de cette série, *CodingBat* propose une solution accessible via le bouton ***Show solution***. Dans votre propre intérêt, essayez de réaliser l'exercice par vous-même avant de regarder la solution.

CONSIGNES

Réalisez au moins quatre exercices parmi ceux-ci :

1. **monkeyTrouble** : *« Nous avons deux singes. Les paramètres aSmile et bSmile indiquent pour chacun d'eux s'ils sont souriants. Ceux-ci nous causent des problèmes lorsqu'ils sont tous les deux souriants ou lorsqu'aucun des deux n'a le sourire. Retournez vrai s'ils nous causent des problèmes ».*

2. **sumDouble** : *« Étant donnés deux entiers, retournez leur somme. Dans le cas où les deux valeurs sont identiques, retournez le double de leur somme ».*

3. **diff21** : *« Étant donné un entier n, retourner la valeur absolue de la différence entre n et 21. Dans le cas où n est plus grand que 21, retournez le double de la valeur absolue de la différence (Astuce : utilisez la fonction* `Math.abs`*) ».*

4. **parrotTrouble** : *« Nous avons un perroquet qui parle fort. Le paramètre hour donne l'heure actuelle exprimée de 0 à 23. Le perroquet nous cause des problèmes s'il parle avant 7h ou après 20h. Retournez vrai s'il nous cause des problèmes ».*

5. **makes10** : *« Étant donnés deux entiers, retournez vrai si l'un d'eux est 10 ou si leur somme vaut 10 ».*

6. **nearHundred** : *« Étant donné un entier, retournez vrai si celui-ci est à une distance inférieure ou égale à 10 de 100 ou de 200 (Astuce : utilisez la fonction* `Math.abs`*) ».*

7. **posNeg** : *« Étant donnés deux entiers, retournez vrai si l'un est positif et l'autre négatif. Dans le cas où le paramètre negative est à vrai, retournez vrai seulement si les deux entiers sont négatifs ».*

8. **notString** : *« Étant donnée une chaîne, retournez une nouvelle chaîne avec le mot "not" suivi de la chaîne fournie. Dans le cas où la chaîne commence déjà par "not", retournez la chaîne sans modification (Astuce : utilisez la fonction* `startsWith` *de la classe* `String` *au lieu de la fonction* `equals`*) ».*

EXERCICE 5 [À FAIRE À DOMICILE] : UN PEU DE LOGIQUE AVEC CODINGBAT

Cette série d'exercices sur le site web *CodingBat* nécessite la construction de conditions plus complexes que dans la série précédente.

PREMIER EXERCICE

Rendez-vous sur la page *http://codingbat.com/java/Logic-1*.

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

Voici une traduction de l'énoncé : *« Lorsque les écureuils se réunissent pour une fête, ils aiment avoir des cigares. Une fête est réussie lorsque le nombre de cigares est compris entre 40 et 60 inclus. Si c'est le week-end, il n'y a pas de limite supérieure sur le nombre de cigares. Retournez vrai si la fête est réussie, ou faux dans le cas contraire »*.

La déclaration de fonction à compléter est :

```
public boolean cigarParty(int cigars, boolean isWeekend) {

}
```

Lors de l'appel de cette fonction, le nombre de cigares est copié dans le paramètre formel cigars et l'indicateur de week-end dans le paramètre formel isWeekend (**`true`** si c'est le week-end, **`false`** dans le cas contraire).

> **REMARQUE**
>
> La condition optimale pour ce premier exercice n'est constituée que de **3 expressions booléennes** et, dès lors, de **2 opérateurs logiques**.

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 au moins quatre exercices parmi ceux-ci :

1. **dateFashion** : *« Vous et votre petit(e) ami(e) essayez d'obtenir une table dans un restaurant. Le paramètre you est une note de 0 à 10 indiquant l'élégance de vos vêtements, et date est une note indiquant l'élégance des vêtements de votre petit(e) ami(e). Votre chance d'obtenir une*

table est représentée par l'une de ces trois valeurs : 0 = non, 1 = peut-être, 2 = oui. Si l'un de vous est très élégant, 8 ou plus, alors le résultat est 2 (oui). Cependant, si l'un de vous est peu élégant, 2 ou moins, alors le résultat est 0 (non). Sinon, le résultat est 1 (peut-être) ».

2. **squirrelPlay** : *« Les écureuils de Palo Alto passent la majeure partie de la journée à jouer. Plus précisément, ils ne jouent que si la température est comprise entre 60 et 90 °F inclus. Si c'est l'été, la limite supérieure est de 100 °F au lieu de 90 °F. Étant donnés la température et l'indicateur d'été, temp et isSummer, retournez vrai si les écureuils jouent, ou faux dans le cas contraire ».*

3. **caughtSpeed** : *« Vous conduisez trop vite et un policier vous arrête. La décision du policier est représentée par l'une de ces trois valeurs : 0 = pas d'amende, 1 = une petite amende, 2 = une amende élevée. Si la vitesse est de 60 mph ou moins, le résultat est 0. Si la vitesse est comprise entre 61 et 80 mph inclus, le résultat est 1. Si la vitesse est de 81 mph ou plus, le résultat est 2. Le jour de votre anniversaire, votre vitesse peut être 5 mph plus élevée dans tous les cas ».*

4. **sortaSum** : *« Étant donnés deux entiers, retournez leur somme. Dans le cas où la somme est comprise entre 10 et 19 inclus, retournez 20 ».*

5. **alarmClock** : *« Étant donné un jour de la semaine représenté par un entier (0 = Dimanche, 1 = Lundi, 2 = Mardi, ..., 6 = Samedi) et un booléen indiquant si c'est un jour de vacances, retournez une chaîne de la forme "7:00" renseignant l'heure à laquelle le réveil doit sonner. L'heure de réveil doit être "7:00" les jours de semaine et "10:00" le week-end. Dans le cas où c'est un jour de vacances, l'heure de réveil est "10:00" les jours de semaine et "off" le week-end ».*

6. **love6** : *« Le nombre 6 est un nombre génial. Étant donnés deux entiers a et b, retournez vrai si l'un d'eux est 6, ou si leur somme ou leur différence (a - b ou b - a) vaut 6 (Astuce : utilisez la fonction* `Math.abs`*) ».*

7. **in1To10** : *« Étant donné un entier, retournez vrai s'il est compris entre 1 et 10 inclus. Dans le cas où le paramètre outsideMode est à vrai, retournez vrai si l'entier est plus petit ou égal à 1 ou plus grand ou égal à 10 ».*

8. **specialEleven** : *« Un nombre est dit spécial s'il est un multiple de 11, ou s'il est supérieur de 1 à un multiple de 11. Retournez vrai si le nombre entier positif donné est spécial (Astuce : utilisez l'opérateur modulo % pour déterminer qu'un nombre est divisible par un autre) ».*

EXERCICE 6 : DROITES

DESCRIPTION DU PROGRAMME

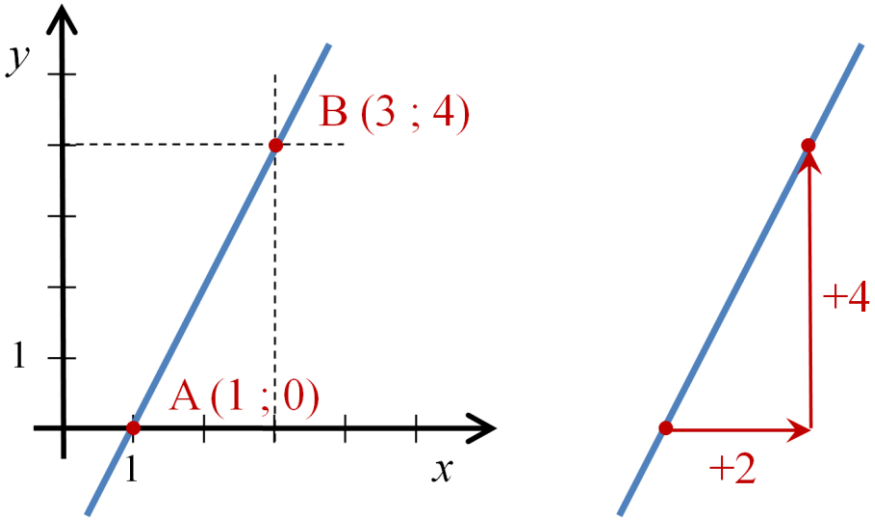
Un étudiant en mathématiques souhaite un programme qui permet de comparer deux droites du plan en indiquant si celles-ci sont sécantes, parallèles ou perpendiculaires.

Pour y parvenir, il faut calculer les **coefficients directeurs** des deux droites afin de les comparer.

Le coefficient directeur d'une droite, ou pente de la droite, décrit l'inclinaison d'une droite. Si deux points distincts d'une droite sont connus, A de coordonnées $(x_A; y_A)$ et B de coordonnées $(x_B; y_B)$, et si la droite n'est pas verticale ($x_A \neq x_B$), il se calcule alors de la manière suivante :

$$coefficient\ directeur = \frac{y_B - y_A}{x_B - x_A}$$

Voici un exemple avec les points A (1 ; 0) et B (3 ; 4) :



Le coefficient directeur de cette droite est $\frac{4-0}{3-1} = \frac{4}{2} = 2$.

Une **droite verticale** n'a pas de coefficient directeur. Sa pente est considérée comme infinie.

Une fois les coefficients directeurs connus pour deux droites, ces droites peuvent être comparées :

→ si les coefficients sont égaux, alors les droites sont **parallèles**.

→ si le produit des coefficients est égal à -1, alors les droites sont **perpendiculaires**.

→ dans le cas contraire, les droites sont **sécantes**.

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.

```
Coordonnées du point A ? 1 0
Coordonnées du point B ? 3 4
Coordonnées du point C ? 0 0
Coordonnées du point D ? 4 -2
Coefficient directeur de la droite AB = +2,000
Coefficient directeur de la droite CD = -0,500
Les droites sont perpendiculaires.
```

CONSIGNES

1. Dans le package `labo4`, ajoutez une classe nommée **`Droites`** comportant une fonction `main`.

2. Dans la classe **`Droites`**, déclarez :

   → une fonction nommée **`lireCoordX`** qui permet d'obtenir l'abscisse (coordonnée x) d'un point du plan à partir des coordonnées exprimées dans une chaîne de caractères de la forme `"x y"` (un caractère espace sépare les coordonnées) :

   ```
   public static double lireCoordX(String coordPoint)
   ```

   → une fonction nommée **`lireCoordY`** qui permet d'obtenir l'ordonnée (coordonnée y) d'un point du plan à partir des coordonnées exprimées dans une chaîne de caractères de la forme `"x y"` :

   ```
   public static double lireCoordY(String coordPoint)
   ```

   → une fonction nommée **`coefficientDirecteur`** qui permet d'obtenir le coefficient directeur d'une droite sur base des coordonnées de deux points distincts de cette droite :

   ```
   public static double coefficientDirecteur(double xA, double
   yA, double xB, double yB)
   ```

   Si les points A et B sont confondus (coordonnées identiques), la fonction retourne **`Double.`***`NaN`*.

   Si la droite est verticale, la fonction retourne **`Double.`***`POSITIVE_INFINITY`*.

CONTRÔLER LA COUVERTURE DES TESTS UNITAIRES

1. Pour tester vos fonctions à l'aide de ***JUnit***, téléchargez sur *Learn* le fichier **`DroitesTest.java`**.

   Créez ensuite un package nommé **`labo4`** dans le dossier des sources **`tests`**, puis placez le fichier **`DroitesTest.java`** dans ce nouveau package.

2. Dans un premier temps, exécutez les tests de cette classe afin de valider le comportement des fonctions *`lireCoordX`*, *`lireCoordY`* et *`coefficientDirecteur`*.

   Si des problèmes sont détectés, corrigez vos fonctions.

3. Nous allons maintenant vérifier si la **couverture du code** par ces tests est suffisante. Autrement dit, il faut nous assurer que les cas d'utilisation de nos fonctions aient été couverts, en particulier pour la fonction *`coefficientDirecteur`* qui propose plusieurs chemins d'exécution en raison de la présence des instructions **`if`**.

   Pour ce faire, voici deux méthodes possibles :

   a. Ouvrez en édition le fichier **`DroitesTest.java`**, puis exécutez les tests en cliquant sur le bouton ***Launch*** 

   b. Faites un clic droit sur le fichier **`DroitesTest.java`**, puis sélectionnez ***Coverage As > JUnit Test***.

   Rendez-vous maintenant dans le fichier **`Droites.java`** pour observer le résultat de l'analyse des instructions couvertes. Par exemple :

```
88    public static double coefficientDirecteur(double xA, double yA, double xB, double yB) {
89        double coeffDir = Double.POSITIVE_INFINITY;
90        if (xA == xB && yA == yB) {
91            coeffDir = Double.NaN;
92        } else if (xA != xB) {
93            coeffDir = (yB - yA) / (xB - xA);
94        }
95        return coeffDir;
96    }
```

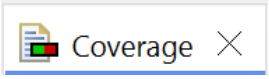
   Légende :

   → En vert, les lignes dont le contenu a été **intégralement** exécuté/évalué.

   → En jaune, les lignes dont le contenu a été **partiellement** exécuté/évalué.

> **REMARQUE**
>
> Dans ce cas, survolez avec le curseur de votre souris la (les) pastille(s) située(s) à gauche du numéro de ligne afin d'obtenir davantage d'informations.

→ En rouge, les lignes dont le contenu n'a **pas du tout** été exécuté/évalué.

Il est également possible de consulter un rapport plus détaillé dans le panneau ***Coverage*** qui est apparu 

.

Après avoir développé les parties ***prb.labos > src > labo4 > Droites.java > Droites***, les informations obtenues ressemblent à celles-ci :

Element	Coverage	Covered Instructions	Missed Instructions
labo4	10,6 %	41	345
Droites.java	21,6 %	41	149
Droites	21,6 %	41	149
main(String[])	0,0 %	0	143
coefficientDirecteur(double, double, doub	88,9 %	24	3
lireCoordX(String)	100,0 %	8	0
lireCoordY(String)	100,0 %	9	0

Lisez attentivement les lignes relatives à vos fonctions, en particulier le pourcentage indiqué dans la colonne ***Coverage***.

> **REMARQUE**
>
> La **couverture de la fonction** `main` n'est pas à prendre en compte dans ce cas puisque cette dernière n'a pas été exécutée par les tests unitaires.

4. Normalement, comme c'est le cas ci-dessus, vous devriez obtenir une **couverture de code inférieure à 100%** pour votre fonction *`coefficientDirecteur`*. Cela signifie que les tests proposés ne sont pas suffisants.

   Pour faire disparaître la surbrillance du code, cliquez sur l'icône ***Remove All Sessions***  du panneau *Coverage*.

   Dans la classe **`DroitesTest`**, complétez les tests de la fonction *`coefficientDirecteur`* afin d'en obtenir une couverture maximale.

SUITE DES CONSIGNES

Maintenant que vos fonctions ont été validées par des tests unitaires suffisamment complets, codez l'ensemble de l'application dans la fonction `main` en vous aidant de ces dernières ainsi que de plusieurs instructions **`if`** pour gérer les différents cas de figure. N'oubliez pas le cas d'erreur (**`Double.`***`NaN`*) !

EXERCICE 7 : LA DATE DU LENDEMAIN

DESCRIPTION DU PROGRAMME

Un service administratif souhaite un programme qui permet de déterminer la date du lendemain d'un jour donné. Par exemple, le lendemain du 28 février 2025 est le 1er mars 2025.

EXEMPLES D'EXÉCUTION

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

```
Date au format j/m/aaaa ? 28/2/2024
Date du lendemain = 29/02/2024
```

```
Date au format j/m/aaaa ? 30/9/2025
Date du lendemain = 01/10/2025
```

```
Date au format j/m/aaaa ? 31/12/2026
Date du lendemain = 01/01/2027
```

CONSIGNES

1. Dans le package `labo4`, ajoutez une classe nommée **`Date`**.
2. Dans la classe **`Date`**, déclarez :

   → une fonction nommée **`getJour`** qui permet d'obtenir la valeur du jour spécifié dans une date exprimée au format [j]j/[m]m/aaaa :

   ```
   public static int getJour(String date)
   ```

   → une fonction nommée **`getMois`** qui permet d'obtenir la valeur du mois spécifié dans une date exprimée au format [j]j/[m]m/aaaa :

```
public static int getMois(String date)
```

→ une fonction nommée **getAnnee** qui permet d'obtenir la valeur d'une année spécifiée dans une date exprimée au format [j]j/[m]m/aaaa :

```
public static int getAnnee(String date)
```

→ une fonction nommée **estBissextile** qui permet de déterminer si une année est bissextile :

```
public static boolean estBissextile(int annee)
```

→ une fonction nommée **joursParMois** qui permet d'obtenir le nombre de jours dans un mois :

```
public static int joursParMois(int mois, int annee)
```

Si la valeur spécifiée pour le mois n'est pas correcte, la fonction peut retourner arbitrairement 0.

CONTRAINTES

Aidez-vous ici de l'instruction **switch** ainsi que de la fonction *estBissextile* déclarée précédemment.

→ une fonction nommée **formaterDate** qui permet d'obtenir une date exprimée sous la forme jj/mm/aaaa :

```
public static String formaterDate(int jour, int mois, int
annee)
```

Par exemple, si les valeurs de `jour`, `mois`, et `annee` sont respectivement 1, 5 et 2025, alors la chaîne retournée est `"01/05/2025"`.

REMARQUE

Aidez-vous de la fonction **String.*format*** en spécifiant le format `"%02d/%02d/%04d"`.

RÉALISER DES TESTS UNITAIRES

1. Pour tester vos fonctions à l'aide de ***JUnit***, créez une classe (via ***New > JUnit Test Case***) nommée **`DateTest`** dans le package **`labo4`** du dossier des sources **`tests`**.

2. Dans cette classe, déclarez 3 fonctions de test pour chaque fonction de la classe **`Date`**. Il est important de **tester des cas de figure suffisamment distincts** et d'**assurer une couverture du code maximale**, en particulier pour les fonctions *`estBissextile`* et *`joursParMois`*.

SUITE DES CONSIGNES

1. Dans le package `labo4`, ajoutez une classe nommée **`Lendemain`** comportant une fonction `main`.

2. Maintenant que vos fonctions ont été validées par des tests unitaires suffisamment complets, codez l'ensemble de l'application dans la fonction `main` en vous aidant de ces dernières ainsi que de plusieurs instructions **`if`** pour gérer les différents cas de figure.

VERSION AMÉLIORÉE

Actuellement, notre programme accepte une date inexistante (par exemple, le 29/02/2025), ce qui le conduit à afficher des résultats incohérents.

Pour empêcher cela, nous allons intégrer les vérifications nécessaires :

1. Dans la classe **`Date`**, déclarez une fonction nommée **`estValide`** qui permet de vérifier si une date existe :

```
public static boolean estValide(int jour, int mois, int annee)
```

2. Dans la classe **`DateTest`**, ajoutez au moins 3 fonctions de test pour la fonction *`estValide`*, tout en veillant à obtenir 100% de couverture.

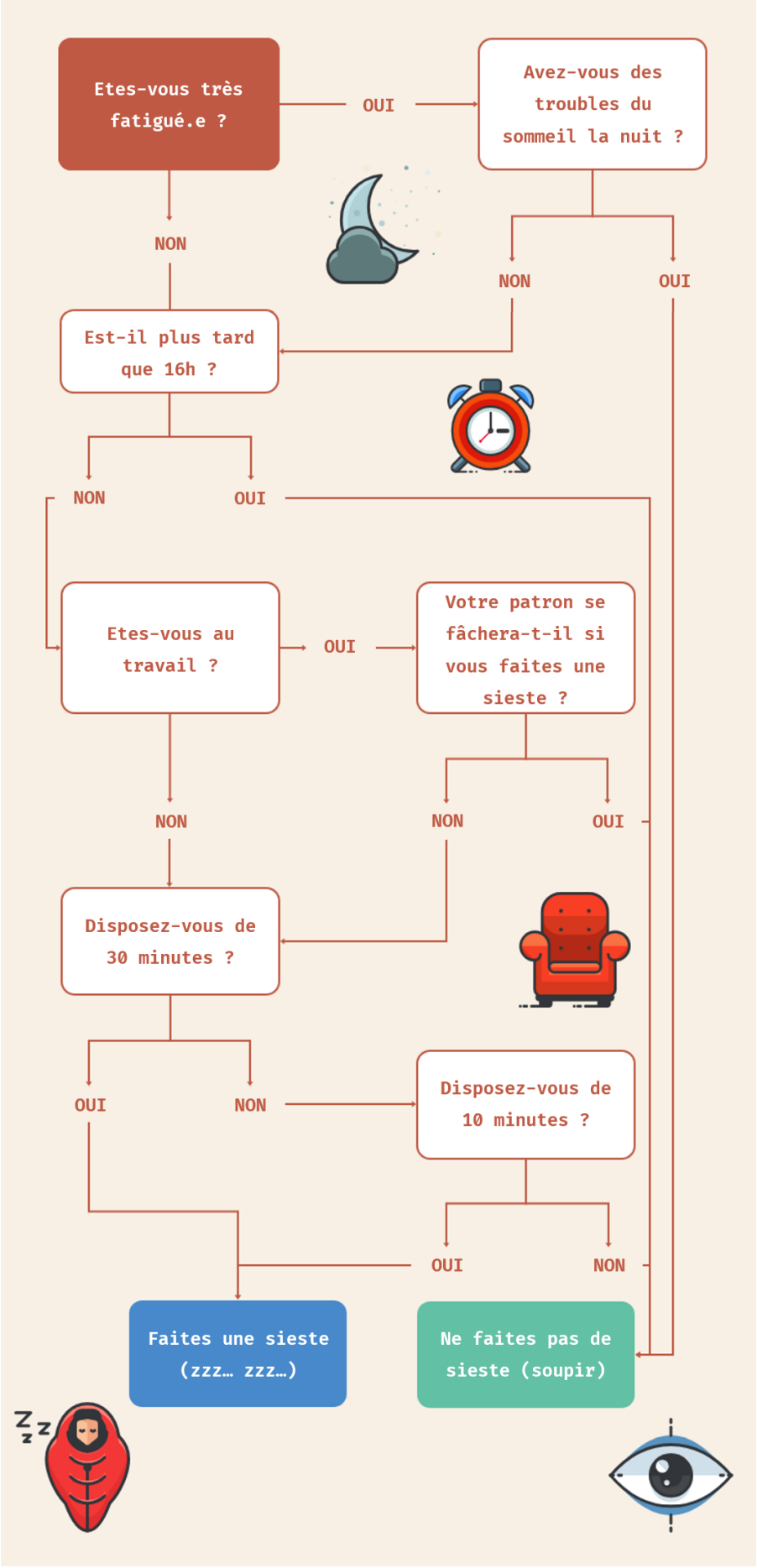
3. Dans la fonction `main` de la classe **`Lendemain`**, après l'acquisition de la date, intégrez la vérification de l'existence de la date saisie par l'utilisateur. Si la date est valide, la date du lendemain est affichée. Dans le cas contraire, le message *« Date incorrecte ! »* est affiché.

EXERCICE 8 [FACULTATIF] : FAIRE UNE SIESTE

DESCRIPTION DU PROGRAMME

Réalisez un programme qui permet de déterminer si une personne doit/peut faire une sieste.

Pour cela, respectez les critères de décision représentés dans le schéma de la page suivante.



Source : https://fr.venngage.com/templates/diagrams/illustrative-decision-flowchart-a47a0eda-255e-45e1-a160-e49bc0f37258

EXEMPLES D'EXÉCUTION

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

```
Etes-vous très fatigué.e ? oui
Avez-vous des troubles du sommeil la nuit ? non
Est-il plus tard que 16h ? non
Etes-vous au travail ? oui
Votre patron se fâchera-t-il si vous faites une sieste ? non
Disposez-vous de 30 minutes ? oui
Faites une sieste (zzz... zzz...)
```

```
Etes-vous très fatigué.e ? oui
Avez-vous des troubles du sommeil la nuit ? non
Est-il plus tard que 16h ? oui
Ne faites pas de sieste (soupir)
```

CONSIGNES

1. Dans le package `labo4`, ajoutez une classe nommée **`FaireUneSieste`** comportant une fonction `main`.

2. Dans la classe **`FaireUneSieste`**, déclarez :

   → les deux constantes suivantes :

   ```
   public static final String FAIRE_UNE_SIESTE = "Faites une
   sieste (zzz... zzz...)";

   public static final String NE_PAS_FAIRE_UNE_SIESTE = "Ne
   faites pas de sieste (soupir)";
   ```

   IMPORTANT

   Ces constantes ne doivent pas être déclarées dans la fonction `main` ! En effet, celles-ci appartiennent spécifiquement à la classe en raison du qualificatif **`static`**.

   Elles sont **utilisées dans la classe de tests** qui vous est fourni.

→ une fonction nommée **choixBinaire** qui permet d'effectuer l'acquisition d'une réponse binaire (« oui » ou « non ») et de retourner la valeur booléenne correspondante :

```
public static boolean choixBinaire(String question)
```

Le paramètre `question` est la question qui doit être affichée à l'utilisateur au moment de l'acquisition.

L'utilisateur peut saisir une partie seulement de la réponse (par exemple, « o » au lieu de « oui »). De plus, la casse de la réponse ne doit pas être prise en compte (par exemple, « Oui » est considéré comme identique à « oui »).

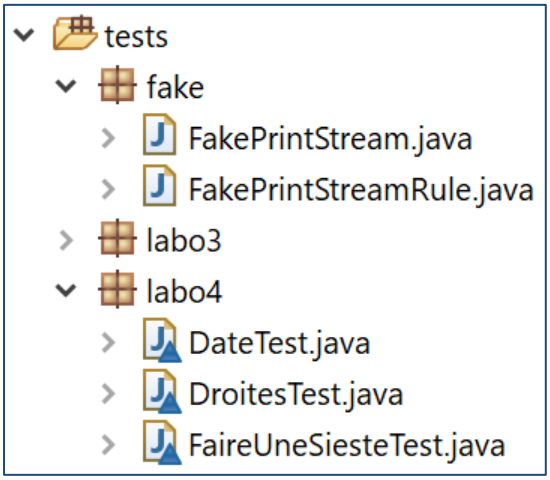
> **REMARQUE**
>
> Aidez-vous pour ce faire des fonctions *trim*, *toLowerCase*/*toUpperCase* et *startsWith* de la classe **String**.

3. Pour tester vos fonctions à l'aide de ***JUnit***, téléchargez sur *Learn* les trois fichiers suivants : **FakePrintStream.java**, **FakePrintStreamRule.java** et **FaireUneSiesteTest.java**.

   Créez un package nommé **fake** dans le dossier des sources **tests**, puis placez les fichiers **FakePrintStream.java** et **FakePrintStreamRule.java** dans ce nouveau package.

   Placez le fichier **FaireUneSiesteTest.java** dans le package **labo4** du dossier des sources **tests**.

   Vous devriez obtenir le résultat suivant :



Ces tests permettent de tester directement votre fonction `main` afin de vérifier que tous les chemins d'exécutions aboutissent à une réponse correcte.

4. Codez maintenant l'ensemble de l'application dans la fonction `main` en vous aidant de la fonction *`choixBinaire`* ainsi que de plusieurs instructions **`if`** pour gérer les différents cas de figure.

> **ASTUCE 1**
>
> Afin d'éviter l'utilisation de techniques court-circuit non recommandées, telles que **`return;`** et **`System.`***`exit()`*, essayez de suivre les conseils ci-dessous.
>
> Déclarez une variable de type **`String`** permettant de mémoriser le résultat à afficher et initialisez-la avec la référence **`null`**. La valeur de cette variable ne sera affichée qu'à la fin de l'exécution de la fonction `main`.
>
> Lorsque l'un des deux résultats est déterminé (faire une sieste ou ne pas faire une sieste), affectez à cette variable la valeur de la constante appropriée, ***`FAIRE_UNE_SIESTE`*** ou ***`NE_PAS_FAIRE_UNE_SIESTE`***.
>
> Si d'autres tests sont effectués par la suite (avant l'affichage du résultat), il devient alors facile de déterminer si la variable a déjà reçu un résultat valide (≠ **`null`**).

> **ASTUCE 2**
>
> Il est également possible de suivre une autre méthode que celle proposée précédemment, cette dernière réduisant drastiquement le code en exploitant la propriété court-circuit des opérateurs **`&&`** et **`||`**.
>
> En procédant de la sorte, une seule instruction **`if`** est nécessaire !