ECOLE SUPERIEURE D'INFORMATIQUE DE GESTION

ÉPREUVE REGROUPÉE

Janvier 2021

BRANCHE : **ALGORITHMES ET PROGRAMMATION OBJET (APO)**

CLASSE : **ESIG2 Classe A (Groupes 1 et 2)**

DATE : 20 janvier 2021

NOM : .....

PRÉNOM : .....

PROFESSEUR : Eric Batard

N° du poste de travail : .....

N° de clé USB : .....

Modalités

- Durée : 240 minutes.
- Travail individuel.
- Documentation personnelle (livres, papiers) : Autorisée.
  Documentation électronique (disquettes, CD, clé USB, …) : Autorisée si elle a été recopiée sur `C:\ESIGUsers` ***avant*** le début de l'épreuve.
- Tout partage de ressources de votre poste de travail avec le réseau ainsi que toute tentative de communication seront considérés comme fraude et sanctionnés comme tels par la note minimale.

Démarrage

- Connectez-vous au réseau sur le poste de travail qui vous a été attribué.
- Créez le dossier `C:\ESIGUsers\APO` ou, s'il existe déjà, videz-le ou renommez-le.
- Copiez dans `C:\ESIGUsers\APO` le *contenu* du dossier réseau qui vous sera indiqué au début de l'épreuve, normalement
  G:\ESIG\Distribution\2020_2021\ESIG-2\APO\Eléments ER APO 20-01-21
- Les éléments fournis comprennent cet énoncé et un répertoire `APO_SARS` qui est un ***répertoire de projet IntelliJ IDEA***. Vous rendrez le répertoire de projet dans lequel vous avez travaillé.

Travail à faire

- Lisez tous les documents fournis.
- Complétez les procédures/méthodes demandées de manière à ce qu'elles répondent aux spécifications de l'énoncé.
- Vous rendrez le répertoire du projet avec les fichiers modifiés.

C'est *à vous* de vérifier que le fichier enregistré et rendu contient bien
la dernière version de votre travail

Le nom du *répertoire de projet* IntelliJ IDEa doit être préfixé par vos initiales

----

A ajouter au plus tard juste avant la reddition

Quand les deux cas se présentent, le masculin est utilisé dans cet énoncé de façon générique.

Partie I – APO au secours d'ALP avec APO_SARS

Le respect des noms demandés et surtout de ceux qui sont *fournis*
est essentiel pour le bon fonctionnement de l'application demandée.

Contexte

Comme vous en avez peut-être le souvenir, vous êtes évalués en ALP (en partie) avec des exercices sans ordinateur et sur papier (« *unplugged* » dans le jargon).

Cette année, vos camarades de 1ère année devaient à la première épreuve, entre autres, écrire un programme de tortue graphique. Pour un enseignant qui les corrige, il n'est pas forcément évident au premier coup d'œil de savoir si une suite d'instructions écrites à la main produit bien la figure demandée.

Pour se faciliter la vie (au moins cette partie de sa vie), un enseignant a eu l'idée d'écrire un programme Java pour lui permettre de *saisir rapidement* les programmes rendus par vos camarades. Afin de pouvoir les exécuter et même d'imprimer le résultat de façon à aboutir à une évaluation indiscutable.

C'est ainsi qu'est né le programme **SARS**, *Saisie Accélérée des Rendus Studieux*. Et votre travail sera de dupliquer ce programme (un peu simplifié quand même).

Pour fixer les idées

Comme les programmes à ce stade d'ALP sont très simples, l'idée est d'avoir des boutons permettant de saisir en un clic les instructions les plus courantes (dans la fenêtre principale) ainsi que les valeurs d'argument les plus fréquentes (dans la boîte de dialogue).

Voici ces interfaces à obtenir. Chaque élément est détaillé dans la suite.

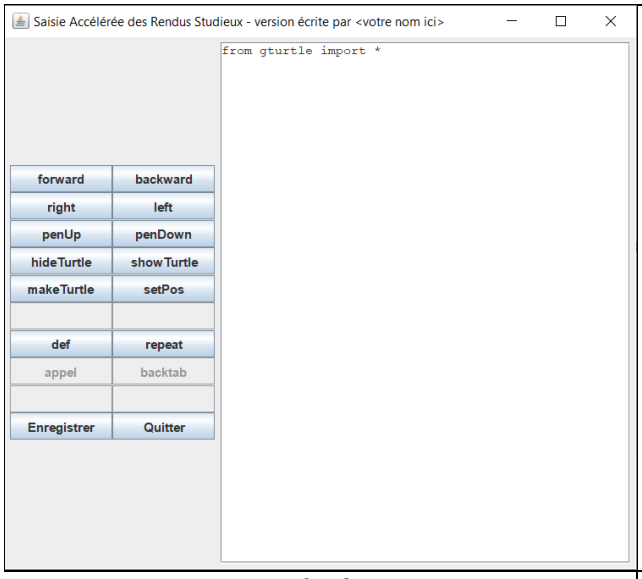


Figure 1 - La fenêtre principale
classe `FenSaisie`

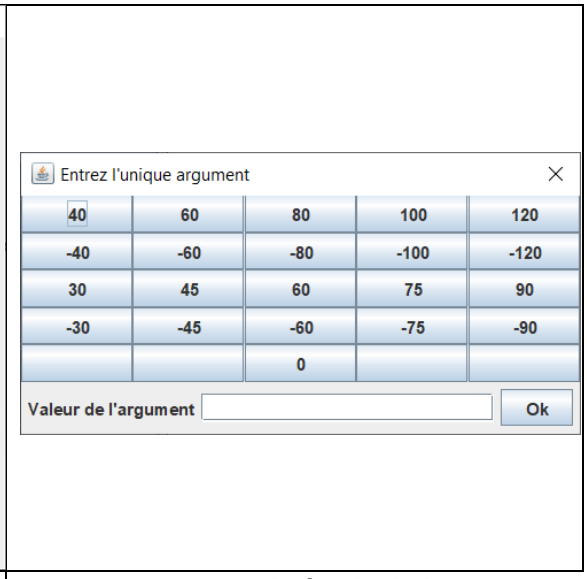


Figure 2 - La boîte de dialogue
classe `DialogueSaisie`

Le principe général de l'application est le suivant :
A chaque fois que l'utilisateur clique sur un bouton (à gauche dans la fenêtre principale de saisie), le texte est automatiquement inscrit dans la zone de texte (à droite dans la fenêtre principale de saisie).

Présentation des boutons de la fenêtre principale

Il y a 10 ***boutons d'opérations*** : `forward`, `backward`, `right`, `left`, `penUp`, `penDown`, `hideTurtle`, `showTurtle`, `makeTurtle` et `setpos`.

Dans les cas où l'opération prend 1 (`forward`, `backward`, `right`, `left`) ou 2 arguments (`setpos`), on utilisera la boîte de dialogue `DialogueSaisie` pour faciliter leur saisie.

Il y a 4 ***boutons spéciaux*** dont les opérations « spéciales » (`def`, `repeat`, `appel`, `backtab`) seront détaillées plus loin ainsi que 2 ***boutons d'action*** (`Enregistrer` et `Quitter`) expliqués plus loin aussi.

Structure de l'interface de la fenêtre principale – classe `FenSaisie`

- Vous penserez à modifier le titre de cette fenêtre en ajoutant votre nom dans la classe `RunSARS`.

Il y a visiblement deux zones côte à côte dans cette fenêtre.

Zone de droite

La zone de droite est une zone de texte de 35 lignes sur 55 colonnes, ***avec*** barres de défilement. Elle sera modifiable par l'utilisateur et contiendra, au départ, une ligne avec le texte `"from gturtle import *"`
Une constante `MESSAGE_DEPART` est déjà définie pour cela dans la classe.

Cette première ligne devra être suivie d'une ligne vide. Une autre constante `NL` est déjà définie pour cela dans la classe.

Pour des questions de lisibilité, vous penserez également à modifier la police de caractères utilisée dans la zone de texte avec l'appel à la méthode `setFont()` comme ceci :

```
<votre zone de texte>.setFont(new Font("Courier New", Font.PLAIN,12));
```

Zone de gauche

La zone de gauche est organisée en une grille de deux colonnes et ne contient que des boutons.

Il se pose deux questions :

1) comment obtenir les textes des boutons (qui ont un texte) pour les créer et
2) comment faire pour avoir ces apparentes lignes vides

Pour répondre à 1) il faudra lire le fichier texte `Op.txt` fourni.

Chaque ligne de ce fichier contient deux informations pour chacune des opérations prises en compte : en premier un entier qui indique le nombre d'arguments de l'opération et en second le nom de l'opération.
Si l'entier est négatif, il s'agit d'un code. Il sera essentiel que vous puissiez associer à chaque nom d'opération l'entier correspondant (cf. plus loin). Le plus simple est d'utiliser une `HashMap`. C'est ce qui est demandé, mais toute autre solution sera acceptée à condition qu'elle dépende des données lues dans le fichier. L'approche qui ne rapporterait pas du tout de point (sur cet aspect) est la recopie manuelle des textes.

Pour répondre à 2) il faut comprendre que ces apparentes lignes vides sont en fait des boutons désactivés. Il suffira de donc de trouver un moyen pour introduire deux boutons désactivés entre la dépose sur l'interface des 10 boutons d'opérations et celle des 4 boutons spéciaux ainsi qu'entre celle des 4 boutons spéciaux et celle des 2 boutons d'action.

Au départ vous pouvez laisser actifs tous les boutons non vides de cette partie.

Structure de l'interface de la boîte de dialogue – classe `DialogueSaisie`

Le squelette de cette classe est fourni. Elle hérite notamment de `SarsDialog` qu'il ne faudra surtout **PAS** modifier. `DialogueSaisie` est un descendant indirect de `JDialog` mais vous pourrez créer son interface comme celle d'une `JFrame` habituelle.

Il y a deux zones : celle du bas et celle du bas.

Zone du haut

La zone du haut est organisée en une grille de cinq lignes ou cinq colonnes et ne contient que des boutons.

Pour obtenir le texte des boutons, y compris les chaînes vides correspondant aux boutons sans texte, vous utiliserez, *à choix*, le tableau de chaîne `tabValeurs` ou l'`ArrayList` de chaînes `listeValeurs`. Ces variables sont déjà définies et initialisées dans `SarsDialog` et donc directement disponibles dans `DialogueSaisie`.

Zone du bas

La zone du bas comprend un `JLabel`, un `JTextField` de 20 colonnes et un `JButton`.

Vous noterez que dans la classe `DialogueSaisie`, une méthode `main` est fournie pour tester votre interface. Pour l'exécuter, pensez bien à changer la configuration dans IntelliJ IDEA.

Enfin notez qu'il est normal de voir le message `Vous avez oublié d'appeler assign()` tant que vous n'avez pas rempli `actionPerformed()`.

Comment faire fonctionner ces interfaces ?

Comme certaines opérations dépendent de `DialogueSaisie`, nous allons commencer par les explications pour cette classe/boîte de dialogue. Toutefois plusieurs parties sont relativement indépendantes les unes des autres et vous pouvez écrire ces parties dans l'ordre que vous voulez.

Fonctionnement de `DialogueSaisie`

Le principe est que le *texte* du bouton (de la zone du haut) sur lequel clique l'utilisateur est recopié dans le `JTextField`. Ce texte remplace ce qui pouvait exister précédemment. Une valeur peut aussi être directement tapée dans le `JTextField`.

Enfin l'utilisateur doit confirmer en cliquant sur OK, ce qui normalement ferme la boîte de dialogue. Sauf si l'utilisateur clique sur OK alors qu'il n'y a rien dans le `JTextField`. Dans ce cas, la boîte n'est ***pas*** fermée.

COMPLÉMENTS TECHNIQUES TRÈS IMPORTANTS

Pour fermer la boîte de dialogue, il faut utiliser `dispose()`.

Pour que la valeur saisie soit correctement répercutée au programme appelant (cf. plus loin), il faut utiliser `assign()` en passant en paramètre la chaîne saisie. (Pour info `assign()` est définie dans la classe `SarsDialog`).

En forme de rappel :
Le plus simple pour obtenir le texte d'un bouton sur lequel on a cliqué est d'utiliser la méthode `getActionCommand()` sur l'instance d'ActionEvent en paramètre d'actionPerformed(). Ce n'est cependant pas la seule possibilité et toutes les manières dépendant de l'événement sont acceptées.

Fonctionnement de `FenSaisie`

Une approche en étapes successives est proposée mais vous avez toute liberté pour sauter certaines étapes, quitte à y revenir par la suite.

COMPLÉMENTS TECHNIQUES IMPORTANTS

Une méthode `aff` pour afficher une ligne dans un `JTextArea` est fournie sous deux formes :

- la forme habituelle à deux paramètres : le nom de votre `JTextArea` et la chaîne à afficher,
- l'autre forme prend un troisième paramètre, le nombre d'espaces écrits avant le texte. Son intérêt et son utilisation sont détaillés plus loin dans la partie Challenge.

Traitement des boutons d'opérations

Un clic sur n'importe lequel des boutons d'opérations doit au minimum écrire le texte du bouton cliqué dans la zone de texte (sur une ligne à part).

Il faut ensuite se préoccuper des arguments. Pour cela, si vous avez suivi ma suggestion, il faut demander à la `HashMap` le nombre d'arguments de l'opération à partir du nom de l'opération.

- Si c'est 0 (zéro), il suffit d'écrire à la suite du nom de l'opération une paire de parenthèses et passer à la ligne. Par exemple `makeTurtle()`
- Si c'est 1, il faut avoir recours une fois à `DialogueSaisie.yield()` (cf. ci-dessous) et écrire la valeur retournée entre parenthèses puis passer à la ligne. Par exemple `forward(100)`
- Si c'est 2, il faut avoir recours deux fois de suite à `DialogueSaisie.yield()` (cf. ci-dessous) et écrire les 2 valeurs retournées entre parenthèses et séparées par une virgule, puis passer à la ligne. Par exemple `setpos(0, 0)`

Comment afficher un `DialogueSaisie` *et récupérer une valeur ?*

En appelant la fonction statique `yield()` comme ceci, avec comme unique paramètre le titre de la boîte de dialogue affichée :

```
DialogueSaisie.yield("Pour tester votre interface de DialogueSaisie")
```

Si vous affichez le résultat de cet appel avec `System.out.println()` vous verrez la valeur saisie et si vous voyez apparaître `Vous avez oublié d'appeler assign()` c'est que… vous avez oublié d'appeler `assign()` dans votre `DialogueSaisie` ou que vous l'avez mal fait (ou pas encore).

Traitement du bouton d'action `Enregistrer`

Le bouton Enregistrer affiche juste un message comme ceci :

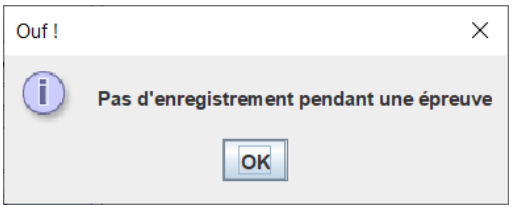


Figure 3

Il doit aussi réinitialiser la zone de texte ainsi que désactiver 2 des boutons spéciaux (cf. plus loin)

Bouton `Quitter` *et gestion de la fermeture de la fenêtre principale*

Un clic sur le bouton `Quitter` ***aussi bien*** qu'un clic sur la case de fermeture provoquent l'affichage de cette boîte de dialogue (de type *confirm*)

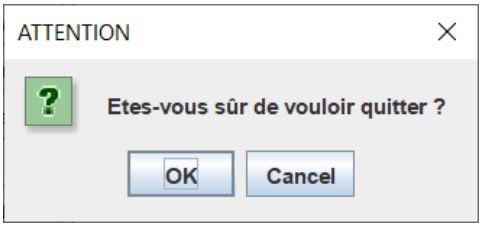


Figure 4

Clairement un clic sur `OK` ferme l'application. Et un clic sur `Cancel` revient à la fenêtre principale, inchangée.

Pour gérer correctement cette confirmation (ou pas) de fermeture, il faut notamment (mais pas uniquement) indiquer une opération par défaut de fermeture `DO_NOTHING_ON_CLOSE` (et pas `EXIT_ON_CLOSE` comme d'habitude).

Traitement du bouton spécial `repeat`

Un clic sur ce bouton provoque l'affichage d'une boite de dialogue (de type *input*) où l'utilisateur pourra donner le nombre d'itérations.

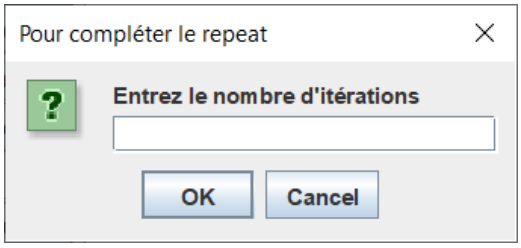


Figure 5

Si l'utilisateur ferme la boîte sans rien indiquer, il ne se passe rien. Sinon le texte `repeat` est affiché suivi du nombre d'itérations suivi de deux-points pour respecter la syntaxe de Python.

Il y aura un traitement de l'indentation ultérieur, expliqué plus loin dans la partie Challenge.

Traitement du bouton spécial `def`

Un clic sur ce bouton provoque l'affichage d'une boite de dialogue (de type input) où l'utilisateur pourra donner le nom de la fonction.

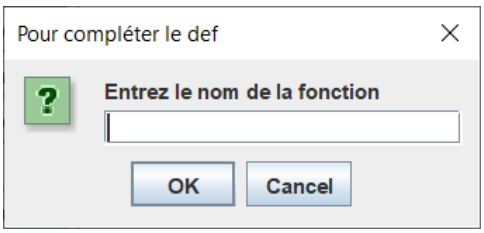


Figure 6

Si l'utilisateur ferme la boîte sans rien indiquer, il ne se passe rien. Sinon le texte `def` est affiché suivi du nom de la fonction, lui-même suivi d'une paire de parenthèses et de deux-points pour respecter la syntaxe de Python.

Ici aussi traitement de l'indentation ultérieur, expliqué plus loin dans la partie Challenge.

Il faudra également mémoriser le nom de la fonction dans une liste globale de chaînes. Cette liste des fonctions définies servira pour le bouton `appel` (ci-après).

Traitement du bouton spécial `appel`

Un clic sur ce bouton provoque l'affichage d'une boite de dialogue (de type *input*) où l'utilisateur pourra choisir le nom d'une des fonctions précédemment définies à partir de la liste complétée au moment des définitions de fonctions.

Ci-dessous affichage après la définition de 3 fonctions : `hexagone`, `triangle` et `cercle`.

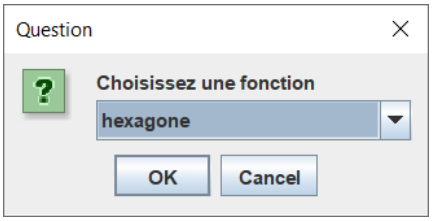


Figure 7

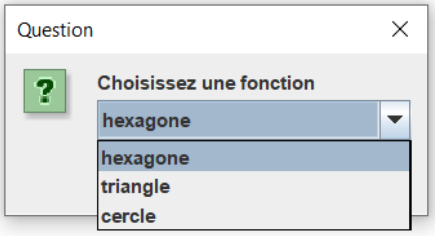


Figure 7 bis – liste déroulée

Si l'utilisateur ferme la boîte sans rien indiquer, il ne se passe rien. Sinon le nom de la fonction choisie est affiché suivi d'une paire de parenthèses pour respecter la syntaxe de Python.

Comme vous travaillerez certainement avec une instance d'`ArrayList<String>` pour mémoriser les noms de fonctions, voici l'instruction qui permet d'obtenir facilement un tableau d'objets à partir d'une `ArrayList` :

`Object[]` <le nom de votre tableau> `=` <le nom de votre `ArrayList`>`.toArray();`

Gestion de l'activation ou pas du bouton `appel`

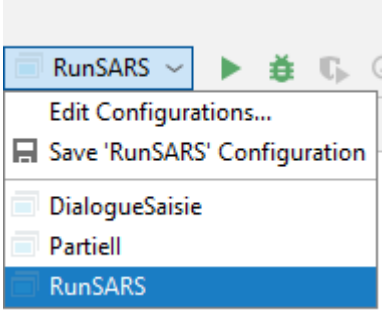
Le principe est simple : au départ on ne peut pas appeler une fonction car on en a défini aucune (la liste des fonctions définies est vide). Dès qu'une fonction est définie, l'écriture d'un appel devient possible et le bouton `appel` est activé.

Notez que quand on (ré)initialise la zone de texte, la liste des fonctions définies doit être vidées (méthode `clear()` par exemple) et par conséquent le bouton `appel` doit être désactivé.

COMPLÉMENT TECHNIQUE GÉNÉRAL

Vous devrez trouver une façon d'identifier les boutons `appel` et `backtab` à part des autres. Il y a là encore plusieurs façons de faire. Dans ma solution je n'identifie à part que ces deux boutons mais vous pouvez tout à fait procéder autrement en donnant un nom à chacun des boutons.

COMPLÉMENT TECHNIQUE CONCERNANT LE PROJET FOURNI



Le projet est fourni avec 3 configurations (au sens d'IntelliJ) :

— RunSARS pour lancer la fenêtre principale

— DialogueSaisie pour tester la boîte de dialogue et vérifier que vous utilisez bien assign() pour renvoyer une valeur.

— PartieII pour exécuter la… partie II !

N'oubliez pas de changer de configuration si nécessaire !

Partie Challenge : Traitement de l'indentation et du bouton `backtab`

Il n'est pas question ici de faire de l'analyse syntaxique mais simplement de gérer les espaces pour respecter les principes de Python. Il est recommandé de traiter cette partie une fois que le reste fonctionne de façon suffisante et satisfaisante. C'est pourquoi elle donnera lieu plutôt à des bonus.

Pour réaliser cette gestion de l'indentation, vous utiliserez la forme à 3 paramètres de la méthode d'affichage fournie, `aff`.

Pour gérer le nombre d'espaces à afficher, ma recommandation est d'utiliser une variable globale dans `FenSaisie`, un entier initialisé à 0 qui indique combien d'espaces on doit écrire avant d'afficher de nouvelles lignes.

Le principe est simple : dès que vous avez « ouvert » une définition de fonction (par le bouton `def`) ou une boucle `repeat`, il faut décaler de 4 espaces (convention Python) les instructions suivantes.

Comme on supposera que l'utilisateur ne définira pas de fonctions dans une fonction, on peut (re)mettre toutes les définitions à l'indentation 0.

Enfin le bouton `backtab` permet, comme son nom l'indique, de revenir en arrière (de 4 espaces à chaque clic). Ce bouton ne s'active que quand la variable globale pour l'indentation n'est plus nulle.

Exemple de programme à saisir avec les 3 fonctions en exemple	*Résultat sous TigerJython*

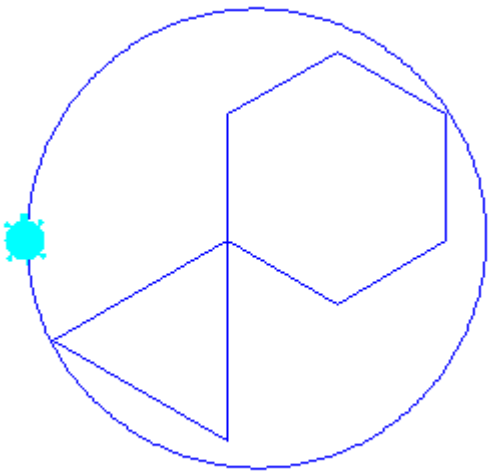
```
from gturtle import *

def hexagone():
    repeat 6:
        forward(63)
        right(60)

def triangle():
    repeat 3:
        left(120)
        forward(100)

def cercle():
    setPos(-100, 0)
    repeat 360:
        forward(2)
        right(1)


makeTurtle()
hideTurtle()
hexagone()
triangle()
cercle()
showTurtle()
```



Partie II – APO au secours d'ALP – le retour

Les correcteurs de l'épreuve d'ALP de septembre attendaient deux fichiers source en Python (donc avec l'extension `.py`). Mais il y a eu quelques surprises…

L'objectif de cette partie est de compléter la classe `PartieII` pour permettre la vérification des rendus.

Les données : tous les fichiers et répertoires se trouvent dans le répertoire `Rendus`, fourni et directement dans le répertoire de projet IntelliJ IDEA. Merci donc d'utiliser la constante `REP_DEPART` fournie.

Le répertoire `Rendus` contient un répertoire pour chaque étudiant (les noms ont été brouillés).

Pour obtenir l'extension d'un fichier, `findExtension()` est fourni.

Le problème : Normalement dans chaque sous-répertoire d'étudiant on devrait trouver uniquement deux fichiers `.py` (cas d'un répertoire *conforme*). Mais ce n'est pas toujours vrai.

Le travail à faire : Vous allez écrire ***deux procédures/méthodes*** de vérification. La première sera non-récursive, la seconde nécessitera un parcours récursif.

La première vérification : Le programme rejette comme non conforme avec le message indiqué dans les sorties produites :

- Les sous-répertoires qui ne contiennent rien (p. ex. `Rdugheb`)
- Les sous-répertoires qui ne contiennent pas exactement 2 éléments (p. ex. `Hyzxesov` et `Lapcyx`)
- Les sous-répertoires qui contiennent exactement 2 éléments mais l'un des deux (au moins) est un répertoire (p. ex. `Hedtaah`)
- Les sous-répertoires qui contiennent exactement 2 fichiers mais l'un des deux (au moins) n'a pas l'extension `.py` (p. ex. `Ijwewasu`).

Sinon le répertoire est conforme.

La seconde vérification : Cette fois-ci le programme pardonne une erreur dans les niveaux de répertoires. Si le sous-répertoire examiné contient un sous-répertoire unique, le programme va chercher récursivement si un des sous-répertoires contient les deux fichiers `.py`.

Les critères qui ne changent pas sont quand même répétés pour que vous puissiez voir ce qu'il y a à modifier par rapport à la précédente vérification.

Le programme va rejeter comme non conforme avec le message indiqué dans les sorties produites :

- Les sous-répertoires qui ne contiennent rien (p. ex. `Rdugheb`)
- Les sous-répertoires qui contiennent plus de 2 éléments (p. ex. `Lapcyx`)
- Les sous-répertoires qui contiennent exactement 2 éléments mais l'un des deux (au moins) est un répertoire (p. ex. `Hedtaah`)
- Les sous-répertoires qui contiennent exactement 2 fichiers mais l'un des deux (au moins) n'a pas l'extension `.py` (p. ex. `Ijwewasu`).
- Si le sous-répertoire contient exactement un sous-sous-répertoire, on recommence la vérification à partir de ce sous-sous répertoire. Si on trouve un sous-répertoire qui vérifie les conditions, le répertoire est jugé conforme (p. ex. `Rendus\Zineum\MonRendu\Delepreuve` et `Rendus\Hyzxesov\ALP` et `Rendus\Fgearag\SEPTEMBRE`)

Sinon le répertoire est conforme.

Sorties produites

```
PREMIERE VERIFICATION
Rendus\Cakdocao conforme !
Rendus\Dintyo conforme !
Rendus\Fgearag non conforme : 1 élément(s) au lieu de 2
Rendus\Hedtaah non conforme : au moins un répertoire au lieu d'un fichier
Rendus\Heonevy conforme !
Rendus\Hyzxesov non conforme : 1 élément(s) au lieu de 2
Rendus\Ijwewasu non conforme : au moins un fichier n'a pas l'extension .py
Rendus\Jawso conforme !
Rendus\Lapcyx non conforme : 3 élément(s) au lieu de 2
Rendus\Quzik non conforme : 5 élément(s) au lieu de 2
Rendus\Rdugheb non conforme : aucun fichier rendu !
Rendus\Ugi conforme !
Rendus\Wemey conforme !
Rendus\Wotyq non conforme : 3 élément(s) au lieu de 2
Rendus\Zineum non conforme : 1 élément(s) au lieu de 2
Rendus\Zivad non conforme : 1 élément(s) au lieu de 2

-------------------------

SECONDE VERIFICATION
Rendus\Cakdocao conforme !
Rendus\Dintyo conforme !
Rendus\Fgearag\SEPTEMBRE conforme !
Rendus\Hedtaah non conforme : au moins un répertoire au lieu d'un fichier
Rendus\Heonevy conforme !
Rendus\Hyzxesov\ALP conforme !
Rendus\Ijwewasu non conforme : au moins un fichier n'a pas l'extension .py
Rendus\Jawso conforme !
Rendus\Lapcyx non conforme : 3 fichiers au lieu de 2
Rendus\Quzik non conforme : 5 fichiers au lieu de 2
Rendus\Rdugheb non conforme : aucun fichier rendu !
Rendus\Ugi conforme !
Rendus\Wemey conforme !
Rendus\Wotyq non conforme : 3 fichiers au lieu de 2
Rendus\Zineum\MonRendu\Delepreuve conforme !
Rendus\Zivad non conforme : un seul fichier rendu
```