MOOC Init Prog Java
Exercices semaine 3

Exercice 9 : tables de multiplication (itération `for`)

Écrivez un programme `Tables.java` affichant les tables de multiplication de 2 à 10.

Votre programme devra produire la sortie suivante à l'écran :

```
        Tables de multiplication

Table de 2 :
  1 * 2 = 2
  ...
  10 * 2 = 20
...
Table de 5 :
  1 * 5 = 5
  2 * 5 = 10
  ...
...
Table de 10 :
  1 * 10 = 10
  ...
```

Méthode :

Utilisez deux structures d'itération `for` imbriquées l'une dans l'autre.

Exercice 10 : Plus grand diviseurs commun (structures de contrôle)

Ecrivez un programme `PGDC.java` qui calcule et affiche le plus grand diviseur commun de deux nombres entiers positifs entrés au clavier. Exemples d'exécution du programme:

```
Entrez un nombre positif :  9
Entrez un nombre positif :  6
Le plus grand diviseur commun de 9 et 6 est 3

Entrez un nombre positif :  9
Entrez un nombre positif :  4
Le plus grand diviseur commun de 9 et 4 est 1
```

Utilisez l'algorithme d'Euclide pour déterminer le plus grand diviseur. Cette formule se résume comme suit:

> Soient deux nombres entiers positifs `a` et `b`. Si `a` est plus grand que `b`, le plus grand diviseur commun de `a` et `b` est le même que pour `a-b` et `b`. Vice versa si `b` est plus grand que `a`.

Les équivalences mathématiques utiles sont:

1. Si `a > b`, alors `PGDC(a, b) = PGDC(a-b, b)`
2. `PGDC(a, a) = a`

Exemple de calcul de PGDC(42, 24):

1. 42 > 24, alors PGDC(42, 24) = PGDC(42--24, 24) = PGDC(18, 24) = PGDC(24,18)
2. 24 > 18, alors PGDC(24, 18) = PGDC(24--18, 18) = PGDC(6, 18) = PGDC(18, 6)
3. 18 > 6, alors PGDC(18, 6) = PGDC(18--6, 6) = PGDC(12, 6)
4. 12 > 6, alors PGDC(12, 6) = PGDC(12--6, 6) = PGDC(6, 6)
5. Résultat: PGDC(42, 24) = PGDC(6, 6) = 6

Indication: utilisez une boucle (par exemple `while`) qui s'occupe de modifier et de tester les valeurs de `a` et `b` jusqu'à ce qu'une solution soit trouvée.

Exercice 11 : rebonds de balles (itération `for`)

Première partie

L'objectif de cet exercice est de résoudre le problème suivant :

Lorsqu'une balle tombe d'une hauteur initiale ***h***, sa vitesse à l'arrivée au sol est $v = \sqrt{2 \times h \times g}$. Immédiatement après le rebond, sa vitesse est $v1 = eps \times v$ (où ***eps*** est une constante et ***v*** la vitesse avant le rebond). Elle remonte alors à la hauteur $h = \frac{{v_1}^2}{2g}$.

Le but est d'écrire un programme (`Rebonds1.java`) qui calcule la hauteur à laquelle la balle remonte après un nombre `nbr` de rebonds.

Méthode :

On veut résoudre ce problème, non pas du point de vue formel (équations) mais par **simulation** du système physique (la balle).

Utilisez une itération `for` et des variables `v`, `v1`, (les vitesses avant et après le rebond), et `h`, `h1` (les hauteurs au début de la chute et à la fin de la remontée).

Tâches :

Écrivez le programme `Rebonds1.java` qui affiche la hauteur après le nombre de rebonds spécifié.

Votre programme devra utiliser la **constante** `g`, de valeur 9,81 et demander à l'utilisateur d'entrer les valeurs de :

- **H0** (hauteur initiale, contrainte : H0 >= 0),
- **eps** (coefficient de rebond, contrainte 0 <= eps < 1)
- **nbr** (nombre de rebonds, contrainte : 0 <= NBR).

Essayez les valeurs H0 = 25, eps = 0.9, NBR = 10. **La hauteur obtenue devrait être environ 3.04**.

Deuxième partie

On se demande maintenant combien de rebonds fait cette balle avant que la hauteur à laquelle elle rebondit soit plus petite que (ou égale à) une hauteur donnée `h_fin`.

Écrivez le programme `Rebonds2.java` qui affiche le nombre de rebonds à l'écran.

Il devra utiliser une boucle `do...while`, et demander à l'utilisateur d'entrer les valeurs de :

- **H0** (hauteur initiale, contrainte : H0 >= 0),
- **eps** (coefficient de rebond, contrainte 0 <= eps < 1)
- **h_fin** (hauteur finale désirée, contrainte : 0 < h_fin < H0).