Java & Algorithme – Corrigé du test final

1 QUESTIONS EN JAVA

Chaque question est sur un point, sauf certaines sur deux points. En tout, le QCM est noté sur 24 : la note est divisée par 6 pour revenir entre 0 et 4.

1. (/1) Comment afficher **Hello** à l'écran en Java ?

System.out.println("Hello"); ou System.out.print("Hello");

2. (/1) Quel est le mot clef pour **sinon** ?

else

3. (/1) Quelle paire de symboles caractérise les tableaux (cochez la case) ?

☐ <> **x** [] ☐ {} ☐ ()

4. (/1) Que dois-je écrire pour créer un tableau d'entiers possédant 8 cases ?

int tab[];

tab = new int[8];

5. (/1) Lequel/Lesquels de ces mots clefs permettent de déclarer un nombre à virgule (cochez la case) ?

☐ int **x** double ☐ short **x** float

6. (/1) Voici un extrait de code permettant de remplir toutes les cases du tableau **tab** avec le chiffre 4:

```
int compteur=0;
while(compteur<=30)
{
      tab[compteur]=4;
}
```

Combien de cases possède le tableau tab ? **31**

7. (/1) Que dois-je écrire pour déclarer un entier a ? **int a;**

8. (/1) Quel(s) mot(s) clef(s) peut-il y avoir avant le mot clef **else** ?

☐ while **x** if **x** else if ☐ for

9. (/2) Cet extrait de code calcule la moyenne des entiers a et b (déclarés précédemment), et place le résultat dans m. Est-ce que cela fonctionne ? Sinon, corrigez le programme (a et b sont considérés comme déjà déclarés)..

```
double m;
m = (a+b) / 2.0;
```

On pouvait aussi transformer A+B en double en plaçant (double) devant.

10. (/1) Que dois-je écrire pour déclarer un entier a sur le plus grand espace mémoire possible ?

long a;

11. (/1) J'ai un tableau **tab**. Quelle est sa taille ? **tab.length**

12. (/1) Quand on écrivait du code dans un fichier en TP de Java, quelle était l'extension du fichier (le mot situé après le point dans le nom de fichier) ?

java

13. (/1) J'ai trois variables a, b et c. Je souhaite permuter ces variables (placer a dans b, b dans c, et c dans a). De combien de variables supplémentaires ai-je besoin pour réaliser l'opération ?

☐ 0 **x** 1 ☐ 2 ☐ 3

14. (/2) Cet extrait de code est sensé placer 1 dans m si a est plus grand que b, et 0 sinon. Fonctionne-t-il ? Si non, corrigez-le (a, b et m sont considérés comme déjà déclarés).

```
if (a > b)
{
      m=1;
}
else ~~(a <= b)~~
{
      m=0;
}
```

on pouvait aussi transformer le else en else if.

15. (/1) J'ai fait un programme, et le compilateur me dit **"variable m might not be initialized"**. Quelle ligne de code, à placer au début du programme, me permettrait de ne pas avoir ce message ?

Il faut placer une valeur dans m, avec, par exemple, m=0;

16. (/2) Cet extrait de code est sensé inverser les valeurs de a et de b (précédemment déclarés). Fonctionne-t-il ? Si non, corrige-le (a et b sont considérés comme déjà déclarés).

```
int c;
c=a;
a=b;
b=c;
```

17. (/2) Le tableau **tab** possède 8 cases. Je souhaite remplir le tableau avec la valeur 3. Cet extrait de code fonctionne-t-il ? Si non, corrigez-le (tab est considéré comme déjà déclaré).

```
int compteur;
for(compteur=7; compteur>=0; compteur=compteur-1)
{
      tab[compteur]=3;
}
```

On pouvait aussi décider de faire un for, avec compteur allant de 0 à 7, ou bien un while.

18. (/2) Cet extrait de code doit placer dans m la somme de tous les entiers positifs plus petits ou égaux à n. Fonctionne-t-il ? Si non, corrigez-le (n est considéré comme déjà déclaré).

```
int compteur=0;
int m=0;
while(compteur <= n);
{
      m=m+compteur;
      compteur=compteur+1;
}
```

Ne pas oublier de retirer le point virgule juste après le while.

19. (/1) J'ai écrit un programme dans le fichier Main.java. Quelles sont les deux commandes à écrire (dans l'invite de commande Windows) afin de compiler mon programme puis de l'exécuter ?

javac Main.java

java Main

2 PETITE MISE EN ROUTE

1. **(/1)**Que fait cet algorithme (sachant que **n** est la taille du tableau tab)?

```
AlgoMystere(int tab[], int n)
{
        compteur=0
        m=0
        Tant que (compteur < n)
        {
                m = m + tab[compteur]
                compteur = compteur + 1
        }
        Afficher(m)
}
```

Cet algorithme calcule, dans m, la somme de toutes les cases du tableau tab.

2. **(/1)**Je veux afficher le plus grand nombre inférieur ou égal à **a** et divisible par **7**. Par exemple, si **a=24**, je devrais afficher **21**. Complétez l'algorithme ci-après afin d'effectuer cette tache (les points de suspension représentent les parties à compléter).

```
AlgoBizarre(int a)
{
        compteur=a
        Tant que (compteur % 7 != 0)
        {
                compteur = compteur -1
        }
        Afficher(compteur)
}
```

3. **(/1)**Je souhaite afficher la partie entière de la racine d'un nombre **a**. Par exemple, pour **9**, je souhaite afficher **3**. Pour **27** (racine de **27** vaut **5,196...**), je souhaite afficher **5**. En vérité, je souhaite calculer le plus grand nombre **b** possible tel que $b^2 \leq a$. Complétez l'algorithme ci-après afin d'effectuer cette tache (les points de suspension représentent les parties à compléter).

```
PartieEntiereRacine(int a)
{
        b = 0
        Tant que (b * b <= a)
        {
                b = b +1
        }

        b=b-1
        Afficher(b)
}
```

3 COLLISION DE CERCLES

Dans les jeux vidéos, il est très important de pouvoir calculer rapidement si deux objets sont en contact : ces algorithmes, dits de "détection de collision" permettent d'éviter, par exemple, que des personnages ne traversent des murs.

Je possède deux cercles : un cercle de centre **A** et de rayon **Ra**, et un cercle de centre **B** et de rayon **Rb**. Les coordonnées de **A** sont notées **(xa, ya)** et les coordonnées de **B** sont notées **(xb, yb)**.

On souhaite écrire un algorithme (très simple) permettant de dire si les deux cercles se rentrent dedans.

La distance entre les deux centres **A** et **B** est

$$\sqrt{(xb-xa)^2+(yb-ya)^2}$$

Sur la figure **a)**, les deux cercles ne se touchent pas, sur la **b)**, ils sont en contact (il y a collision), et sur la **c)**, ils se rentrent totalement dedans (il y a aussi collision).

1. **(/1)**Exprimez, grâce à une inégalité exprimée en fonction de **xa, ya, xb, yb, Ra** et **Rb**, une condition pour qu'il y ait collision entre les deux cercles. Votre formule ne doit pas contenir de racine carrée (une mise au carré peut aider à faire disparaître une racine carrée).

Indice : Sur la figure **b)**, les deux cercles se touchent en un seul point (c'est le début de la collision). Quelle relation y a-t-il à ce moment là entre **xa, ya, xb, yb, Ra** et **Rb ?** Une fois cette relation trouvée, vous pouvez facilement trouver l'inégalité demandée.

Il y a collision si et seulement si $\sqrt{(xb-xa)^2+(yb-ya)^2} \leqslant Ra+Rb$

Donc, sans la racine carrée, ça donne $(xb-xa)^2+(yb-ya)^2 \leqslant (Ra+Rb)^2$

2. **(/1,5)**Écrivez un très court programme, **DetectionCollision(xa, ya, Ra, xb, yb, Rb)**, qui renvoie **vrai** si les cercles **A** et **B** sont en collision, et **faux** sinon.

```
DetectionCollision(xa, ya, Ra, xb, yb, Rb)
{
    Si( (xb-xa)*(xb-xa) + (yb-ya)*(yb-ya) <= (Ra+Rb)*(Ra+Rb) )
    {
        retourner vrai;
    }
    Sinon
    {
        retourner faux;
    }
```

}

3. **(/0,5)** D'autres personnes ont réalisé une détection de collision entre deux carrés. Or, le programme de détection de collision des carrés est assez lent (car beaucoup de calculs compliqués). Voyez-vous un moyen d'utiliser votre programme pour accélérer la détection de collision des carrés (faîtes simplement une description écrite, aucun code n'est nécessaire) ?

!!Attention, ici, on ne demande pas de faire un programme de détection de collision entre carrés. On possède un programme qui fonctionne, mais qui est lent, et on veut savoir comment éviter de faire ces calculs lents en utilisant, telle quelle, notre fonction de détection de collision de cercles.

On peut utiliser notre programme pour éviter, dans certains cas, les calculs fastidieux de la détection de collision des carrés. On considère les cercles inscrits dans nos carrés : si ces deux cercles sont en collision, on est sûrs que les carrés sont en collision (pas besoin de faire d'autres calculs).

Sinon, on considère les cercles circonscrits aux carrés : s'ils ne se touchent pas, alors les deux carrés ne se touchent pas.

Sinon, alors on est dans une petite zone où les carrés sont proches, et il faut faire le calcul fastidieux pour décider s'il y a collision.

4 DIVISIBILITÉ

Dans tout cet exercice, vous n'avez pas le droit d'utiliser l'opération de modulo (%).

1. **(/2)** Écrivez une fonction **ExtraireChiffreUnite(a)**, qui renvoie le chiffre des unités du nombre entier **a** passé en paramètre.

Par exemple, **ExtraireChiffreUnite(476)** renvoie **6**.

```
ExtraireChiffreUnite(a)
{
        unite = a – (a/10)*10
        retourner unite
}
```

On pouvait aussi faire une boucle while qui retire 10 à a, tant que a reste supérieur ou égal à 10. Cependant, c'est plus fastidieux.

2. **(/1)**Écrivez une fonction **EstDivisiblePar5(a)** permettant de savoir si le nombre entier **a** passé en paramètre est divisible par **5**. Je vous rappelle qu'un nombre est divisible par **5** si et seulement si son chiffre des unités est **5** ou **0**.

Indice : Vous pouvez utiliser la fonction **ExtraireChiffreUnite**, même si vous n'avez pas su la faire à la question précédente. Je vous rappelle que si vous écrivez, dans votre code, **b=ExtraireChiffreUnite(476)**, alors **b** vaudra **6**.

```
EstDivisiblePar5(a)
{
        u = ExtraireChiffreUnite(a)
        Si( (u==5) || (u==0) )
        {
                afficher "oui"
        }
        Sinon
        {
                afficher "non"
        }
}
```

3. **(/3)** Un nombre est divisible par **7** si et seulement si le résultat de la soustraction du nombre des dizaines par le double du chiffre des unités est divisible par **7**. En itérant cette opération, le nombre sera divisible par **7** seulement si on obtient à la fin **7, 0** ou **-7**. Si on obtient un autre chiffre (**9, 8, 6, 5, 4, 3, 2, 1, -1, -2, -3, -4, -5, -6, -8, ou -9**), le nombre n'est pas divisible par **7**.

Écrivez une fonction **EstDivisiblePar7(a)**, qui renvoie **vrai** si l'entier **a** est divisible par **7**, et **faux** sinon.

Par exemple :

34 104 : son chiffre des unités vaut 4, son nombre des dizaines vaut 3 410 (on retire simplement le chiffre des unités du nombre, et on retire le signe s'il y en avait un). On calcule 3410 – 2x4 = 3402, et on recommence.

3 402 : son chiffre des unités vaut 2, son nombre des dizaines vaut 340, on calcule 340 – 2x2 = 336.

336 : on calcule 33 – 2x6 = 21.

21 : on calcule 2 – 2x1 = 0.

On obtient 0 : 34 104 est bien divisible par 7.

Autre exemple :

4 572 : on calcule 457 – 2x2 = 453.

453 : on calcule 45 – 2x3 = 39.

39 : on calcule 3 – 2x9 = -15.

-15 : son nombre des dizaines vaut 1 (et non pas -1), on calcule 1 – 2x5 = -9.

On obtient -9 : 4 572 n'est pas divisible par 7.

```
EstDivisiblePar7(a)
{
        b=a
        Tant que ( b>9 )
        {
                chiffre_unite = ExtraireChiffreUnite(b)
                nombre_dizaine = b/10
                b= nombre_dizaine - 2* chiffre_unite
                Si (b<0)
                {
                        b = -b
                }
        }

        Si ( (b==7) || (b==0) )
        {
                afficher "oui"
        }
        Sinon
        {
                afficher "non"
        }
}
```

5 LE CRIBLE D'ERATOSTHÈNE

Le crible d'Erathostène (inventé pendant l'antiquité) permet de trouver tous les nombres premiers entre 2 et un nombre choisi. Il fonctionne par élimination des multiples.

Voyons comment fonctionne le crible sur un exemple : on commence par choisir un nombre, disons **20** (on va donc chercher tous les nombres premiers entre **2** et **20**). Je place tous les nombres de **2** à **20** dans un tableau.

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

On commence par le nombre **2** : on va supprimer (ici, on place une croix à la place du nombre dans le tableau) tous les nombres multiples de deux. Pour cela, on se déplace de deux cases en deux cases, à partir de la case qui contient deux, et on supprime les chiffres correspondant. Le chiffre **2** étant dans la case d'indice ***0***, je vais donc supprimer les nombres dans les cases d'indices ***2***, ***4***, ***6***, ***8***, ...

2	3	x	5	x	7	x	9	x	11	x	13	x	15	x	17	x	19	x

On continue avec le prochain chiffre non supprimé du tableau, c'est à dire **3**, qui se situe dans la case d'indice ***1***. On va parcourir les cases du tableau de trois en trois, et supprimer les chiffres qui s'y trouvent. En partant de la case ***1***, on va donc aller à la case ***4*** (1+3), ***7***, ***10***, ***13***, ... et supprimer ce qui s'y trouve.

2	3	x	5	x	7	x	x	x	11	x	13	x	x	x	17	x	19	x

On continue avec le prochain chiffre non supprimé du tableau, c'est à dire **5** (à la case n°***3***). On parcourt les cases de cinq en cinq, à partir de la case d'indice ***3*** : on va donc à la case ***8*** (3+5), ***13***, ***18***, ... et on supprime ce qui s'y trouve.

2	3	x	5	x	7	x	x	x	11	x	13	x	x	x	17	x	19	x

Une fois cette opération répétée pour tout le monde, on obtient

2	3	x	5	x	7	x	x	x	11	x	13	x	x	x	17	x	19	x

Les nombres premiers entre **2** et **20** sont donc **2, 3, 5, 7, 11, 13, 17, et 19**.

1. A vous d'écrire une fonction **Eratosthene(a)**, qui affiche tous les nombres premiers entre **2** et **a**, grâce à l'utilisation du système décrit précédemment.

Indice : votre fonction devra construire un tableau qui contiendra les entiers entre **2** et **a**. Pour supprimer un entier du tableau, vous pouvez tout simplement remplacer l'entier par le chiffre **-1**.

Ici, le but n'était pas de refaire l'exemple, c'est à dire le cas où a=20... Il fallait traiter tous les cas possibles (beaucoup ont pensé que a=20, et n'ont donc traité qu'un tableau de 20 cases, et les multiples de 2, 3 et 5).

```
Eratosthene(a)
{
	int tab[]
	tab = new int[a-1]

	//Construction du tableau
	compteur=0
	Tant que (compteur < a-1 )
	{
		tab[compteur] = compteur +2
		compteur = compteur+1
	}

	//Elimination des nombres non premiers
	compteur = 0
	Tant que (compteur < a-1)
	{
		nombre = tab[compteur];
		compteur2 = compteur+nombre;
		Tant que (compteur2 < a-1)
		{
			tab[compteur2] = -1
			compteur2 = compteur2 + nombre
		}
	}

	//Affichage des résultats
	compteur = 0
	Tant que (compteur < a-1 )
	{
		Si (tab[compteur] != -1)
		{
			afficher tab[compteur]
		}
		compteur=compteur+1
	}
}
```