

NADÉ **TP 4** Université de Bretagne Sud

Alexandre **Méthodes** IUT de Vannes

D2 28/09/2022 BUT Info 1D

# Exercice 1

Écrire les deux méthodes suivantes :

* int factoriel (int n)
* int combinaison (int n, int k)

Tester la méthode factoriel() en utilisant la méthode testFactoriel() :

**Code**

1. /\*\*
2. \* calcul de la factoriel du paramètre
3. \* @param n valeur de la factoriel à calculer
4. \* @return factoriel de n
5. \*\*/
6. int factoriel (int n) {
7. int somme = 1;
8. int i = 1;
10. while (i <= n) {
11. somme = somme \* i;
12. i = i + 1;
13. }
15. return somme;
16. }
18. /\*\*
19. \* calcul de la combinaison k parmi n
20. \* @param n cardinalité de l’ensemble
21. \* @param k nombre d’éléments dans n avec k<=n
22. \* @return nombre de combinaisons de k parmi n
23. \*\*/
24. int combinaison (int n, int k) {
25. int value = factoriel(n) / (factoriel(k) \* factoriel(n -k));
26. return value;
27. }

# Réponse

\*\*\* testFactoriel()

factoriel (5) = 120 : OK

factoriel (0) = 1 : OK

factoriel (1) = 1 : OK

factoriel (2) = 2 : OK

# Exercice 1.2

Sur le modèle de testFactoriel(), écrire la méthode testCombinaison().

# Code

1. /\*\*
2. \* Teste la méthode combinaison()
3. \*\*/
4. void testCombinaison () {
5. System.out.println ();
6. System.out.println ("\*\*\* testCombinaison()");
7. testCasCombinaison (5, 3, 10);
8. testCasCombinaison (0, 0, 1);
9. testCasCombinaison (1, 0, 1);
10. testCasCombinaison (1, 1, 1);
11. testCasCombinaison (2, 1, 2);
12. testCasCombinaison (2, 2, 1);
13. }
15. /\*\*
16. \* teste un appel de combinaison
17. \* @param n valeur de la combinaison à calculer
18. \* @param k valeur de la combinaison à calculer
19. \* @param result resultat attendu
20. \*\*/
21. void testCasCombinaison (int n, int k, int result) {
22. // Arrange
23. System.out.print ("combinaison (" + n + ", " + k + ") \t= " + result + "\t : ");
24. // Act
25. int resExec = combinaison(n, k);
26. // Assert
27. if (resExec == result){
28. System.out.println ("OK");
29. } else {
30. System.err.println ("ERREUR");
31. }
32. }

# Réponse

\*\*\* testCombinaison()

combinaison (5, 3) = 10 : OK

combinaison (0, 0) = 1 : OK

combinaison (1, 0) = 1 : OK

combinaison (1, 1) = 1 : OK

combinaison (2, 1) = 2 : OK

combinaison (2, 2) = 1 : OK

# Exercice 1 (BONUS)

Pour combinaison(25,24) obtient -2 car lors du calcul de des différents factorielles, le résultat dépasse le nombre maximum d’un type Entier (INT).

Pour régler ce problème, si k == n-1 on return n.

# Code

1. long combinaison (long n, long k) {
2. if (k == n-1) {
3. return n;
4. } else {
5. return factoriel(n) / (factoriel(k) \* factoriel(n-k));
6. }
7. }

# Réponse

\*\*\* testCombinaison()

combinaison (5, 3) = 10 : OK

combinaison (0, 0) = 1 : OK

combinaison (1, 0) = 1 : OK

combinaison (1, 1) = 1 : OK

combinaison (2, 1) = 2 : OK

combinaison (2, 2) = 1 : OK

combinaison (25, 24) = 25 : OK

combinaison (13, 12) = 13 : OK

combinaison (25, 25) = 1 : OK

# Exercice 2

Écrire une méthode estDiviseur() qui rend vrai si le deuxième entier divise le premier, faux sinon.

Écrire la méthode testEstDiviseur() qui teste la méthode estDiviseur()

# Code

1. /\*\*
2. \* teste la divisibilité de deux entiers
3. \* @param p entier positif à tester pour la divisibilité
4. \* @param q diviseur strictement positif
5. \* @return vrai ssi q divise p
6. \*/
7. boolean estDiviseur(int p, int q) {
8. if (q == 0) {
9. if (p == 0) {
10. return true;
11. } else {
12. return false;
13. }
14. }
15. if (p % q == 0) {
16. return true;
17. } else {
18. return false;
19. }
20. }
22. /\*\*
23. \* test de la méthode estDiviseur
24. \*\*/
25. void testEstDiviseur() {
26. System.out.println();
27. System.out.println("\*\*\* testEstDiviseur()");
28. testCasEstDiviseur(5, 10, false);
29. testCasEstDiviseur(10, 5, true);
30. testCasEstDiviseur(5, 11, false);
31. testCasEstDiviseur(5, 0, false);
32. testCasEstDiviseur(0, 5, true);
33. testCasEstDiviseur(0, 0, true);
34. }
36. /\*\*
37. \* teste un appel de estDiviseur
38. \* @param p entier positif à tester pour la divisibilité
39. \* @param q diviseur strictement positif
40. \* @param result resultat attendu
41. \*\*/
42. void testCasEstDiviseur(int p, int q, boolean result) {
43. // Arrange
44. System.out.print("estDiviseur (" + p + ", " + q + ") \t= " + result + "\t : ");
45. // Act
46. boolean resExec = estDiviseur(p, q);
47. // Assert
48. if (resExec == result) {
49. System.out.println("OK");
50. } else {
51. System.err.println("ERREUR");
52. }
53. }

# Réponse

\*\*\* testEstDiviseur()

estDiviseur (5, 10) = false : OK

estDiviseur (10, 5) = true : OK

estDiviseur (5, 11) = false : OK

estDiviseur (5, 0) = false : OK

estDiviseur (0, 5) = true : OK

estDiviseur (0, 0) = true : OK

# Exercice 3

En utilisant la méthode estDiviseur(), écrire la méthode estParfait()

Écrire la méthode TestEstParfait() qui teste la méthode estParfait()

# Code

1. /\*\*
2. \* teste si un nombre est parfait
3. \* @param a entier positif
4. \* @return vrai ssi a est un nombre parfait
5. \*/
6. boolean estParfait(int a) {
7. int somme = 0;
8. boolean result;
10. for (int i = 1; i < a; i++) {
11. if (estDiviseur(a, i)) {
12. somme = somme + i;
13. }
14. }
15. if (somme == a) {
16. result = true;
17. } else {
18. result = false;
19. }
21. return result;
22. }
24. /\*\*
25. \* test de la méthode estParfait
26. \*\*/
27. void testEstParfait() {
28. System.out.println();
29. System.out.println("\*\*\* testEstParfait()");
30. testCasEstParfait(6, true);
31. testCasEstParfait(28, true);
32. testCasEstParfait(496, true);
33. testCasEstParfait(8128, true);
34. testCasEstParfait(33550336, true);
35. testCasEstParfait(3, false);
36. testCasEstParfait(4, false);
37. testCasEstParfait(5, false);
38. testCasEstParfait(7, false);
39. testCasEstParfait(8, false);
40. testCasEstParfait(9, false);
41. testCasEstParfait(10, false);
42. }
44. /\*\*
45. \* teste un appel de estParfait
46. \* @param a entier positif
47. \* @param result resultat attendu
48. \*\*/
49. void testCasEstParfait(int a, boolean result) {
50. // Arrange
51. System.out.print("estParfait (" + a + ") \t= " + result + "\t : ");
52. // Act
53. boolean resExec = estParfait(a);
54. // Assert
55. if (resExec == result) {
56. System.out.println("OK");
57. } else {
58. System.err.println("ERREUR");
59. }
60. }

# Réponse

\*\*\* testEstParfait()

estParfait (6) = true : OK

estParfait (28) = true : OK

estParfait (496) = true : OK

estParfait (8128) = true : OK

estParfait (33550336) = true : OK

estParfait (3) = false : OK

estParfait (4) = false : OK

estParfait (5) = false : OK

estParfait (7) = false : OK

estParfait (8) = false : OK

estParfait (9) = false : OK

estParfait (10) = false : OK

# Exercice 4

Écrire une méthode QuatreNbParfaits() qui affiche les quatre premiers nombres parfaits

# Code

1. /\*\*
2. \* affiche les quatre premiers nombres parfaits
3. \*\*/
4. void quatreNbParfaits() {
5. int i = 1;
6. int nbParfait = 0;
7. while (nbParfait < 4) {
8. if (estParfait(i)) {
9. System.out.println(i);
10. nbParfait = nbParfait + 1;
11. }
12. i = i + 1;
13. }
14. }

# Réponse

6

28

496

8128

# Exercice 5

Écrire une méthode testEstCroissant() qui teste la méthode

Écrire le code de la méthode estCroissant()

# Code

1. /\*\*
2. \* teste si les valeurs d’un tableau sont triées par ordre croissant
3. \* @param t tableau d’entiers
4. \* @return vrai ssi les valeurs du tableau sont en ordre croissant
5. \*/
6. boolean estCroissant(int[] t) {
7. boolean result = true;
8. int i = 0;
9. while (i < t.length - 1 && result) {
10. if (t[i] > t[i + 1]) {
11. result = false;
12. }
13. i = i + 1;
14. }
15. return result;
16. }
18. /\*\*
19. \* teste la méthode estCroissant
20. \*\*/
21. void testEstCroissant() {
22. System.out.println();
23. System.out.println("\*\*\* testEstCroissant()");
24. testCasEstCroissant(new int[] { 1, 2, 3, 4, 5 }, true);
25. testCasEstCroissant(new int[] { 1, 2, 3, 5, 4 }, false);
26. testCasEstCroissant(new int[] { 1, 2, 3, 3, 4 }, true);
27. testCasEstCroissant(new int[] { 1, 2, 3, 3, 3 }, true);
28. testCasEstCroissant(new int[] { 1, 2, 3, 2, 4 }, false);
29. testCasEstCroissant(new int[] { 1, 2, 3, 1, 4 }, false);
30. testCasEstCroissant(new int[] { 1, 2, 3, 0, 4 }, false);
31. testCasEstCroissant(new int[] { 1, 2, 3, -1, 4 }, false);
32. testCasEstCroissant(new int[] { 1, 2, 3, -2, 4 }, false);
33. testCasEstCroissant(new int[] { 1, 2, 3, -3, 4 }, false);
34. testCasEstCroissant(new int[] { 1, 2, 3, -4, 4 }, false);
35. testCasEstCroissant(new int[] { 1, 2, 3, -5, 4 }, false);
36. testCasEstCroissant(new int[] { 1, 2, 3, -6, 4 }, false);
37. testCasEstCroissant(new int[] { 1, 2, 3, -7, 4 }, false);
38. testCasEstCroissant(new int[] { 1, 2, 3, -8, 4 }, false);
39. testCasEstCroissant(new int[] { 1, 2, 3, -9, 4 }, false);
40. testCasEstCroissant(new int[] { 1, 2, 3, -10, 4 }, false);
41. }
43. /\*\*
44. \* teste un appel de estCroissant
45. \* @param t tableau d’entiers
46. \* @param result résultat attendu
47. \*/
48. void testCasEstCroissant(int[] t, boolean result) {
49. // Arrange
50. System.out.print("estCroissant (");
51. afficherTab(t);
52. System.out.print(") \t= " + result + "\t : ");
53. // Act
54. boolean resExec = estCroissant(t);
55. // Assert
56. if (resExec == result) {
57. System.out.println("OK");
58. } else {
59. System.err.println("ERREUR");
60. }
61. }

# Réponse

\*\*\* testEstCroissant()

estCroissant ([1, 2, 3, 4, 5]) = true : OK

estCroissant ([1, 2, 3, 5, 4]) = false : OK

estCroissant ([1, 2, 3, 3, 4]) = true : OK

estCroissant ([1, 2, 3, 3, 3]) = true : OK

estCroissant ([1, 2, 3, 2, 4]) = false : OK

estCroissant ([1, 2, 3, 1, 4]) = false : OK

estCroissant ([1, 2, 3, 0, 4]) = false : OK

estCroissant ([1, 2, 3, -1, 4]) = false : OK

estCroissant ([1, 2, 3, -2, 4]) = false : OK

estCroissant ([1, 2, 3, -3, 4]) = false : OK

estCroissant ([1, 2, 3, -4, 4]) = false : OK

estCroissant ([1, 2, 3, -5, 4]) = false : OK

estCroissant ([1, 2, 3, -6, 4]) = false : OK

estCroissant ([1, 2, 3, -7, 4]) = false : OK

estCroissant ([1, 2, 3, -8, 4]) = false : OK

estCroissant ([1, 2, 3, -9, 4]) = false : OK

estCroissant ([1, 2, 3, -10, 4]) = false : OK