Mathématiques appliquées

Examen pratique – Juin 2025 – Durée 3H30

1. Consignes

1.1. Consignes générales

Les exercices indiqués dans cet énoncé doivent être résolus au moyen du langage Java et de l'environnement de développement Eclipse, en complétant le projet de départ « 2425_B1MATH_Examen_Juin_2025_2_NomPrenom.zip » mis à votre disposition sur HELMo Learn.

Le projet de départ contient toutes les classes citées dans cet énoncé avec, dans celles-ci, les prototypes des méthodes obligatoires que vous devrez compléter pour la réalisation des différentes tâches.
Si nécessaire, vous pouvez enrichir le projet avec vos propres packages, classes et méthodes.

Chaque méthode est précédée d'une Javadoc. Lisez attentivement les informations qui y sont données en complément de l'énoncé.

Le projet de départ contient un plan de test pour la plupart des classes à compléter. Ce plan de test valide les éléments essentiels de votre solution, mais il n'a pas la prétention d'être exhaustif !

1.2. Fichiers de départ

Avant de commencer l'examen, vous devez télécharger les fichiers suivants depuis HELMo Learn :

- énoncé de l'examen (= ce document) ;
- projet Eclipse de départ « 2425_B1MATH_Examen_Juin_2025_2_NomPrenom.zip » ;

Projet Eclipse de départ :

- extraire les fichiers de l'archive Zip, puis importer le projet dans Eclipse,
- **renommer le projet dans Eclipse (Refactor > Rename…) afin de remplacer « NomPrenom » par votre nom et votre prénom**.

1.3. Dépôt de vos solutions

Lorsque vous avez réalisé les différents exercices, vous devez déposer sur HELMo Learn :

- une archive Zip (Export…) « 2425_B1MATH_Examen_Juin_2025_2_NomPrenom.zip » contenant votre projet Eclipse, où « NomPrenom » est remplacé par votre nom et votre prénom.
- un fichier « REGGRM_NomPrenom.jff » contenant votre grammaire, solution de l'exercice 4.1 ;
- un fichier « DFA_NomPrenom.jff » contenant votre automate fini déterministe, solution de l'exercice 4.2.

1.4. Evaluation

- Dépôt tardif, plagiat, projet mal nommé (pas de refactor), erreur(s) de compilation → **pénalité.**
- Les règles de bonne pratique Java sont respectées (identifiants, indentation, etc.) → pénalités en cas de non-respect.
- L'accent est principalement mis sur la qualité algorithmique de la solution proposée.
  - Le code doit être fonctionnel, robuste, efficace.
  - Le code doit être bien structuré, simple, **non redondant**.
- Vos méthodes doivent être correctement **commentées** et **expliquées**.

2. Récursivité

2.1. Nombre d'occurrences d'un caractère dans une chaîne

Dans cet exercice, nous souhaitons déterminer le nombre d'occurrences d'un caractère donné dans une chaîne de caractères, d'abord de manière itérative et ensuite de manière récursive.

Remarque : cette analyse ne sera pas sensible à la casse.

Exemples : `nombreOccurrences("Mon petit chat est blanc", 't')` → 4

`nombreOccurrences("Mon petit chat est blanc", 'z')` → 0

Exercice 2.1 – Nombre d'occurrences d'un caractère dans une chaîne

- Dans la classe « `NombreOccurrences` » du package « `recursivite` », complétez les méthodes suivantes :

  `public static int` **`nombreOccurrencesIteratif`**`(String chaine, char c)`

  → Détermine le nombre d'occurrences du caractère `c` dans la chaîne `chaine`, de manière itérative.

  `public static int` **`nombreOccurrencesRecursif`**`(String chaine, char c)`

  → Détermine le nombre d'occurrences du caractère `c` dans la chaîne `chaine`, de manière récursive.

- Note : on considère que les paramètres `chaine` et `c` fournis lors de l'appel sont valides.
- Vérifiez que vos méthodes fonctionnent correctement en exécutant les plans de test JUnit « `NombresOccurrencesIteratifTests` » et « `NombresOccurrences RecursifTests` ».
- Rappel : vos méthodes doivent être commentées.

2.2. Elément manquant

Soit `tab` un tableau de nombres entiers positifs, trié, sans répétition, à 1 ou à 2 dimensions. Concevez une méthode récursive capable de déterminer le plus petit entier manquant.

Exemples :

$$[0 \quad 1 \quad 2 \quad 7 \quad 9] \rightarrow 3$$

$$[4 \quad 5 \quad 10 \quad 11 \quad 12] \rightarrow 0$$

$$\begin{bmatrix} 0 & 1 & 2 & 6 & 8 \\ 9 & 11 & 13 & 14 & 15 \end{bmatrix} \rightarrow 3$$

$$\begin{bmatrix} 4 & 5 & 10 & 11 & 12 \\ 14 & 15 & 20 & 21 & 22 \end{bmatrix} \rightarrow 0$$

Attention, les signatures des méthodes récursives sont imposées !

Exercice 2.2 – Elément manquant

- La classe « `ElementManquant` » du package « `recursivite` », contient les deux méthodes itératives suivantes déjà complétées :

  `public static int` **`elementManquantIteratif`**`(int[] tab)`

  → Recherche le plus petit entier manquant dans un tableau 1D d'entiers positifs, de manière itérative.

  `public static int` **`elementManquantIteratif`** `(int[][] tab)`

  → Recherche le plus petit entier manquant dans un tableau 2D d'entiers positifs, de manière itérative.

- Complétez les deux méthodes ***récursives*** correspondantes en respectant les signatures des fonctions auxiliaires :

  `public static int` **`elementManquantRecursif`**`(int[] tab)`

  `private static int` **`elementManquantRecursif`**`(int[] tab, int colonne)`

  → Recherche le plus petit entier manquant dans un tableau 1D d'entiers positifs, de manière récursive.

  `public static int` **`elementManquantRecursif`**`(int[][] tab)`

  `private static int` **`elementManquantRecursif`**`(int[][] tab, int ligne, int colonne)`

  → Recherche le plus petit entier manquant dans un tableau 2D d'entiers positifs, de manière récursive.

- Testez que vos méthodes fonctionnent correctement au moyen des plans de test JUnit « ElementManquantRecursif1DTest », « ElementManquantRecursif2DTest ».

- Rappel : vos méthodes doivent être commentées.

3. Graphisme 2D et récursivité – L'étoile fractale

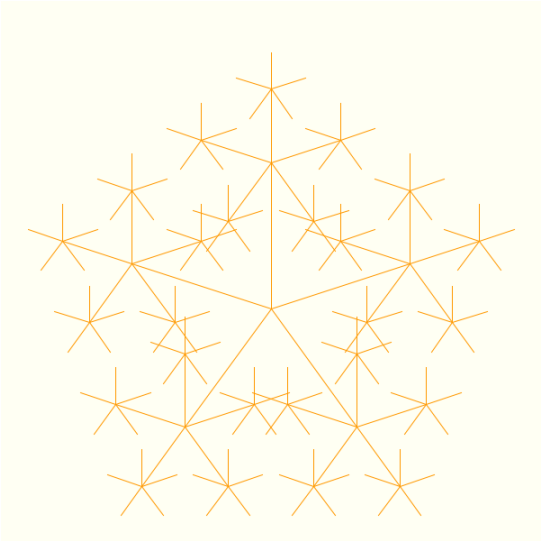
L'objectif de cet exercice est de dessiner **de manière récursive** une étoile fractale.

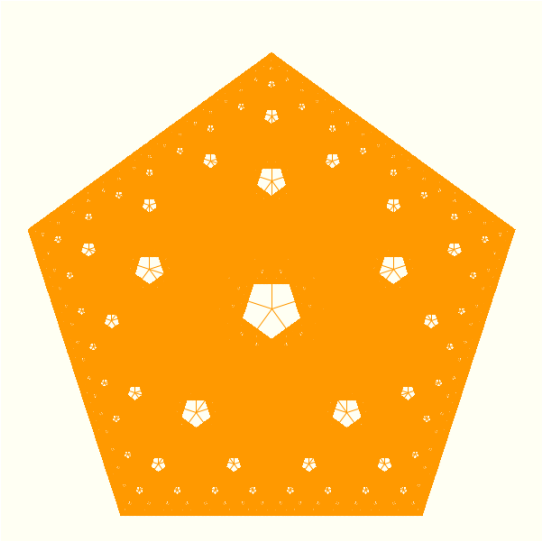
Plus précisément, on demande d'écrire une fonction `dessinerEtoile()` qui dessine des segments de droite (« ligne ») récursivement de manière à former un nombre d'étoiles de plus en plus grand en fonction de l'ordre n de récursion choisi.

Remarque : dans cet exercice, on utilisera obligatoirement les objets définis lors du laboratoire 3 ; à savoir, Point2D, Ligne2D et Dessin2D.

Voici le résultat attendu

si $n = 2$ et $n = 8$:



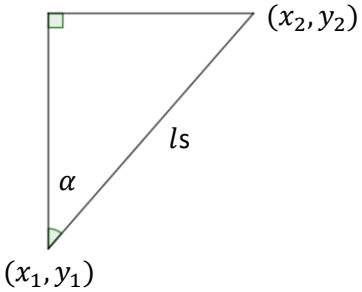


La méthodologie à suivre est la suivante :

Pour chacune des branches de l'étoile (ici, 5 branches) :

- on calcule l'angle qui sépare la branche de l'étoile à dessiner de la verticale ;
  NB : les branches sont régulièrement espacées.
- on définit les deux points délimitant le segment (la branche) à dessiner :
  - les coordonnées (x_1, y_1) du premier point sont fournies en argument ;
  - la longueur ls du segment et l'angle α entre la verticale et le segment sont également fournies en argument ; ces informations permettent de calculer les coordonnées du second point (voir schéma ci-dessous) :

$$x_2 = x_1 + ls \,.\sin\alpha$$
$$y_2 = y_1 + ls \,.\cos\alpha$$



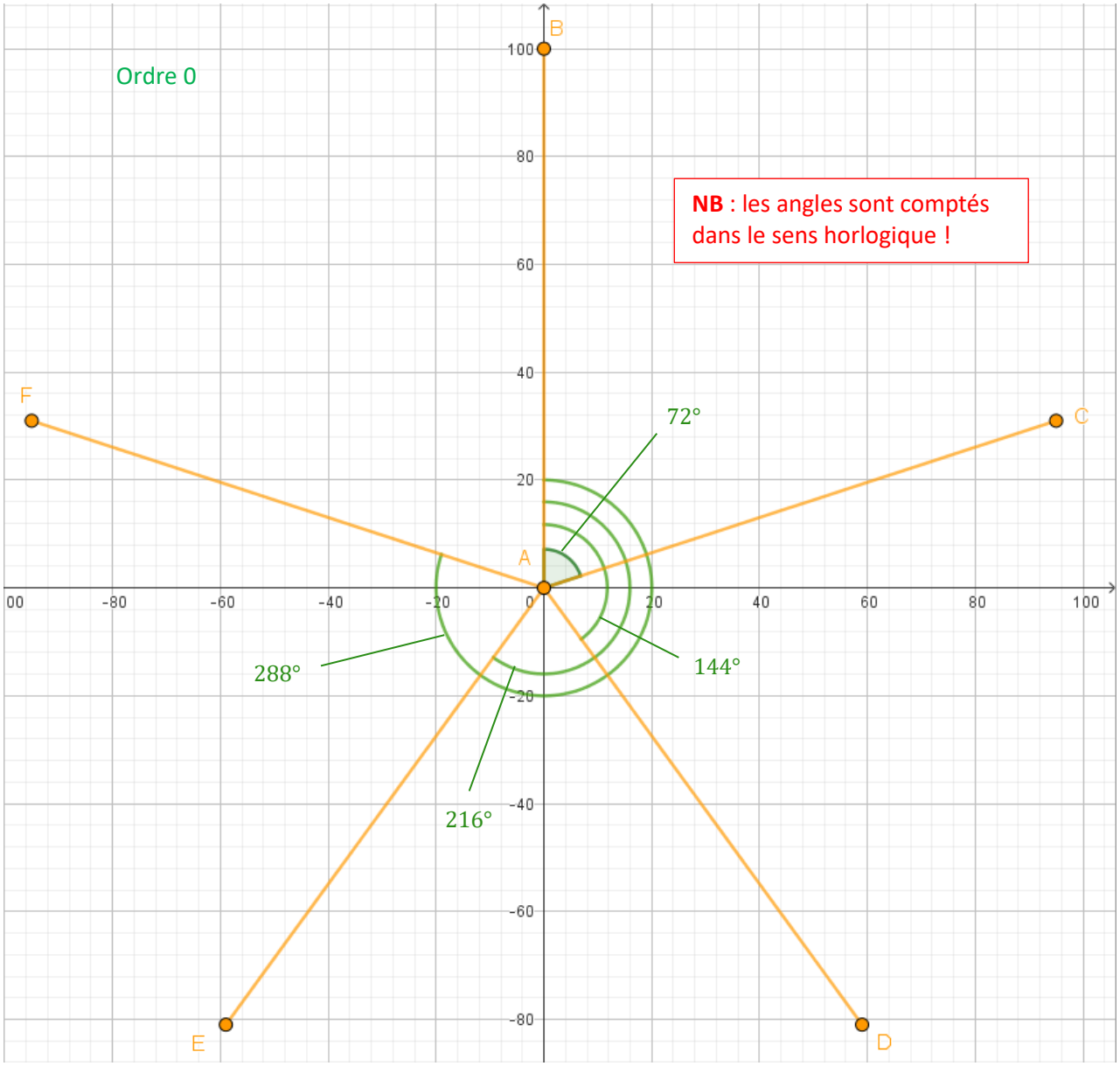
- on ajoute la ligne obtenue au dessin ;

- si l'ordre n > 0, on dessine récursivement les autres étoiles.

  Chaque segment courant donne naissance à une nouvelle étoile, pour laquelle le centre est le second point du segment courant et la longueur est réduite d'un facteur constant (constante *REDUCTION* = 0.5).

Remarque : lorsque $n = 0$, on dessine la première étoile, c'est-à-dire 5 segments de longueur initiale !

Voici un schéma reprenant la construction pour l'ordre 0 :



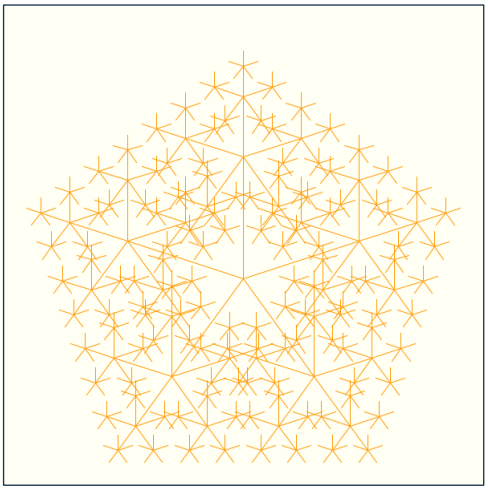
Exercice 3 – L'étoile fractale

- Dans la classe « `EtoileFractale` » du package « `dessin` », complétez la méthode suivante :

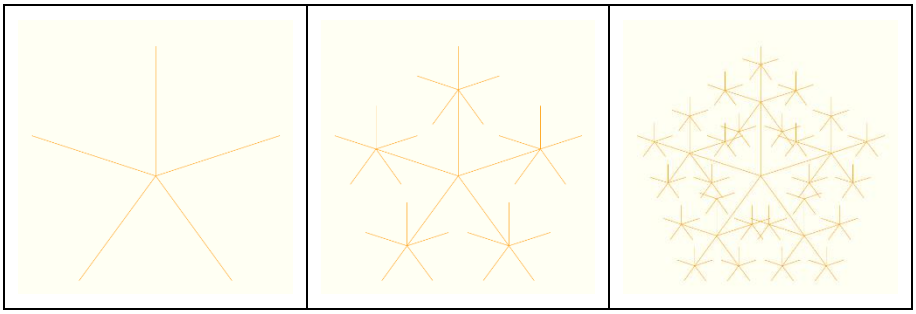
  `public static void` **`dessinerEtoile`**`(Dessin2D dessin, double x, double y, double longueur, int n)`

  → Dessine une étoile fractale, de manière <u>récursive</u>.

- Consultez la documentation du code pour de plus amples informations sur les différentes méthodes et sur les classes « Dessin2D », « `Ligne2D` » et « `Point2D` ».
- Testez que votre méthode fonctionne correctement en exécutant la méthode « `main()` » de la classe.
  Le test de base est configuré avec $n = 3$; ce qui doit conduire au résultat suivant :



Voici les premières figures que nous souhaitons obtenir :



Ordre 0 Ordre 1 Ordre 2

4. Langages formels et automates finis

4.1. Grammaire génératrice d'un langage

Exercice 4.1 – Grammaire génératrice d'un langage

- Construisez dans JFLAP une grammaire régulière génératrice du langage décrit par :

$$L = \{w \in \{a, b, c\}^* \mid w \text{ contient au moins une occurrence de la lettre } a\}.$$

Sauvegardez votre grammaire dans un fichier JFF nommé « REGGRM_NomPrenom.jff » que vous déposerez sur HELMo Learn.

4.2. Automate fini déterministe pour reconnaître un langage

Exercice 4.2 – Automate fini déterministe complet

Soit l'expression régulière suivante : $\left((abb^*)\middle|(b^*c)\right)$ définie sur l'alphabet $\Sigma = \{a, b, c\}$.

- Construisez dans JFLAP un automate d'états finis déterministe complet qui reconnaît le langage décrit par cette expression.

  Remarque : les étiquettes des arêtes seront constituées d'un seul input (Exemple : a ou a, b mais pas aa ou ab)

- Validez que votre automate est correct en simulant son fonctionnement pour un ensemble de mots d'entrée. Vérifiez que les mots qui font partie du langage sont acceptés et que ceux qui n'en font pas partie sont rejetés.

Sauvegardez votre automate dans un fichier JFF nommé « DFA_NomPrenom.jff » que vous déposerez sur HELMo Learn.

5. Opérations de décalage et logiques « bit à bit »

5.1. Chiffrement

Le programme « Chiffrement » permet d'encoder et de décoder un entier, en utilisant uniquement des opérations de décalage et logiques bit à bit.

La méthode de codage est illustrée ci-dessous pour l'entier x = 0xFADE68B1.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	1	0	1	1	0	1	1	1	1	0	0	1	1	0	1	0	0	0	1	0	1	1	0	0	0	1

Inverser :

0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	1	1	0	0	1	0	1	1	1	0	1	0	0	1	1	1	0

Décaler à gauche de 3 positions :

0	0	1	0	1	0	0	1	0	0	0	0	1	1	0	0	1	0	1	1	1	0	1	0	0	1	1	1	0	0	0	0

Appliquer un masque secret (par exemple, 0xA5A5A5A5) de manière à inverser la valeur des bits correspondant aux "1" du masque :

0	0	1	0	1	0	0	1	0	0	0	0	1	1	0	0	1	0	1	1	1	0	1	0	0	1	1	1	0	0	0	0
1	0	1	0	0	1	0	1	1	0	1	0	0	1	0	1	1	0	1	0	0	1	0	1	1	0	1	0	0	1	0	1
1	0	0	0	1	1	0	0	1	0	1	0	1	0	0	1	0	0	0	1	1	1	1	1	1	1	0	1	0	1	0	1

Exercice 5 – Chiffrement

- Dans la classe « Chiffrement » du package « `bitwise` », complétez les méthodes suivantes :

  `public static int` **`encoder`**`(int x)`

  → Encode un entier en appliquant une série de transformations bit à bit.

  `public static int` **`decoder`**`(int y)`

  → Décode un entier (encodé avec la méthode précédente) en appliquant une série de transformations bit à bit ; `x = decoder(encoder(x))`.

  Remarque : vous devez utiliser exclusivement les opérateurs logiques bit à bit et de décalage !

- Vérifiez que votre méthode **`encoder()`** fonctionne correctement en exécutant le plan de test JUnit « `ChiffrementTest` » et testez que votre méthode **`decoder()`** fonctionne correctement en exécutant la méthode « `main()` » de la classe.

- Rappel : vos méthodes doivent être commentées.

6. Matrices

6.1. Matrice antisymétrique

Définition : $A = [a_{ij}]$ est antisymétrique si $\forall i, \forall j : a_{ij} = -a_{ji} \ (i \neq j)$.

Exemples : $[0]$, $\begin{bmatrix} 0 & 3 \\ -3 & 0 \end{bmatrix}$, $\begin{bmatrix} 0 & 4 & 1 \\ -4 & 0 & 2 \\ -1 & -2 & 0 \end{bmatrix}$

Contre-exemples : $[9]$, $\begin{bmatrix} 0 & 3 \\ 3 & 0 \end{bmatrix}$, $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$

Exercice 6.1 – Test de matrice antisymétrique

- Dans la classe « `Matrix` » du package « `matrix` », complétez la méthode

  `public boolean` **`isAntisymmetric`**`()`

  qui détermine si une matrice est antisymétrique ou pas.
- Pensez à optimiser votre méthode de manière à ne pas effectuer d'opérations inutiles !
- Testez que votre méthode fonctionne correctement au moyen du plan de test JUnit.
- Rappel : vos méthodes doivent être commentées.

6.2. Matrices et images

Dans cet exercice, vous allez générer l'image « zoom x2 » de l'image suivante :

Les deux étapes principales sont :

1) L'ajout, dans la classe Matrix, de la méthode extractFromThis() qui permettra de sélectionner dans l'image de départ la zone à redimensionner ;

2) L'implémentation, dans la classe Zoom, de la méthode zoom2Image() qui renvoie une nouvelle image de mêmes dimensions que l'image originale et correspondant à l'application d'un zoom de facteur 2 sur la sous-région centrale de l'image origine.

Exercice 6.2 – « Zoom »

- Dans la classe « `Matrix` » du package « `matrix` », complétez la méthode suivante :

  `public Matrix` **`extractFromThis`**`(int extractRow, int extractCol, int height, int width)`

  → Extrait une sous-matrice de la matrice courante, de dimensions height x width, à partir de la position spécifiée.

  Remarque : les méthodes de la classe Matrix ne modifient pas la matrice courante (`this`) ou une matrice passée en argument, mais renvoient une nouvelle instance contenant le résultat attendu.

- Pour la méthode `extractFromThis()`, vérifiez que les paramètres `extractRow` et `extractCol` fournis lors de l'appel permettent de réellement extraire la sous-matrice prévue. A défaut, la méthode doit lever une exception du type « `IllegalArgumentException` » au moyen de l'instruction « `throw new IllegalArgumentException(`*`<message>`*`)` ».
  Idée : pensez à faire un petit schéma des différentes situations possibles.
- Testez que votre méthode fonctionne correctement en exécutant le plan de test JUnit « `ExtractFromThisTest` ».
- Dans la classe « `Zoom` » du package « `imageProcessing` », complétez la méthode suivante :

  `public Matrix` **`zoom2Image`**`(Matrix image)`

  → Effectue un zoom de facteur 2 sur une image en extrayant une sous-région centrale et en l'agrandissant de manière à obtenir une nouvelle image de mêmes dimensions que l'image origine (voir image ci-dessous et détails dans la documentation Java de la méthode).

  Remarque : l'utilisation de boucles n'est autorisée que pour la création de l'image zoomée ; les autres manipulations feront appel à des méthodes de la classe Matrix.

- Pour la méthode `zoom2Image()`, vérifiez que les dimensions de l'image originale sont bien des nombres pairs.
  A défaut, la méthode doit lever une exception du type « `IllegalArgumentException` » au moyen de l'instruction « `throw new IllegalArgumentException(`*`<message>`*`)` ».
- La méthode `main()` de la classe « Zoom » du package « `imageProcessing` » est déjà complétée et vous permettra de tester votre solution. Le résultat attendu est le suivant :

Bon travail !