Aix*Marseille université
Initiative d'excellence
Année 2018-19

Domaine Sciences et Technologies
Licence informatique

Programmation 2 : TP 2 (informatique)
Code UE : SIN3U02

Affichage de l'ensemble de Mandelbrot

1 Affichage de l'ensemble de Mandelbrot

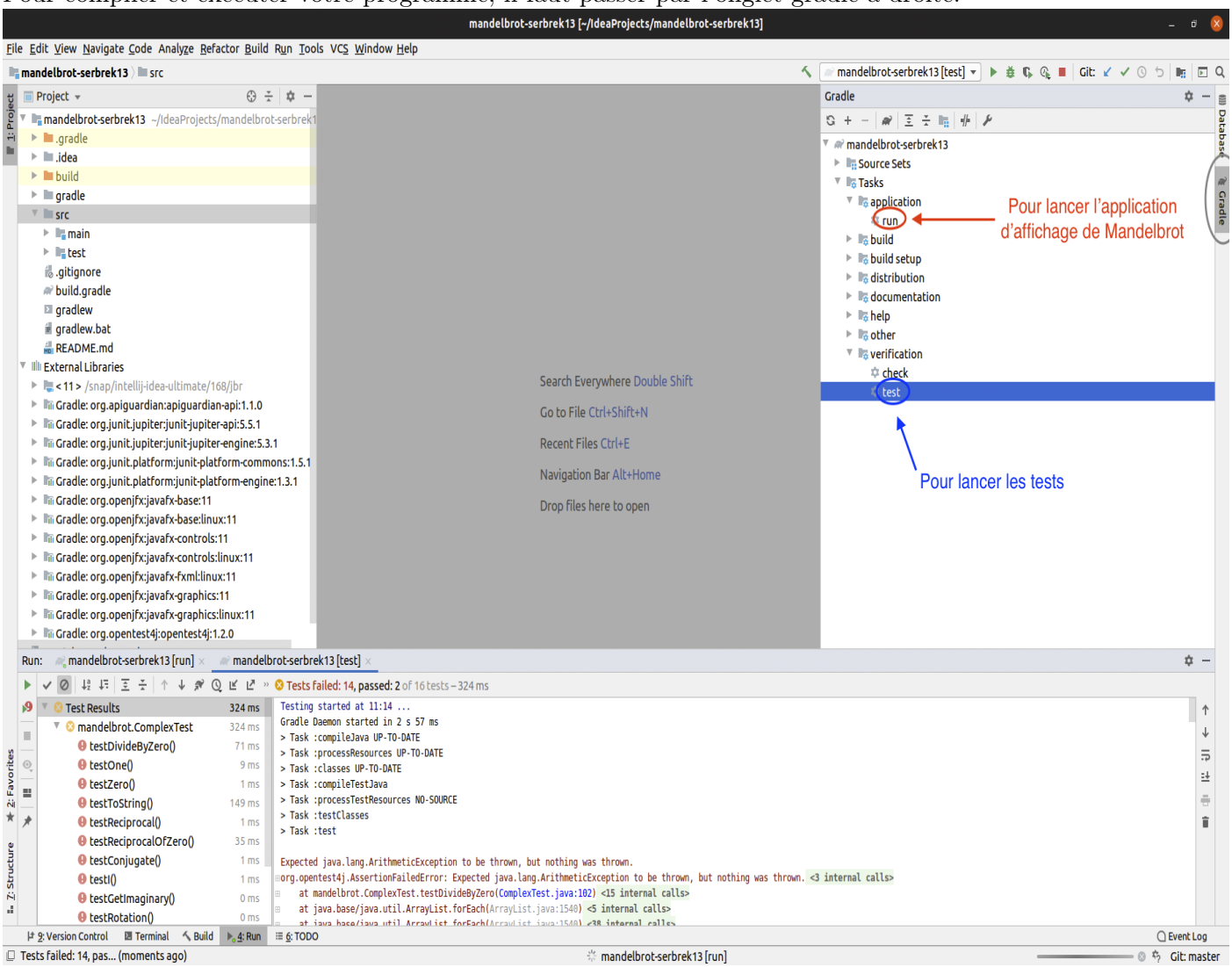
On va travailler sur ce TP sur l'affichage de l'ensemble de Mandelbrot. Pour cela, on va utiliser un projet pré-existant. Malheureusement la classe `Complex` de ce projet est bourrée d'erreurs. Le but du TP sera donc de corriger la classe `Complex` en s'aidant de tests unitaires.

1.1 Récupérer le dépôt

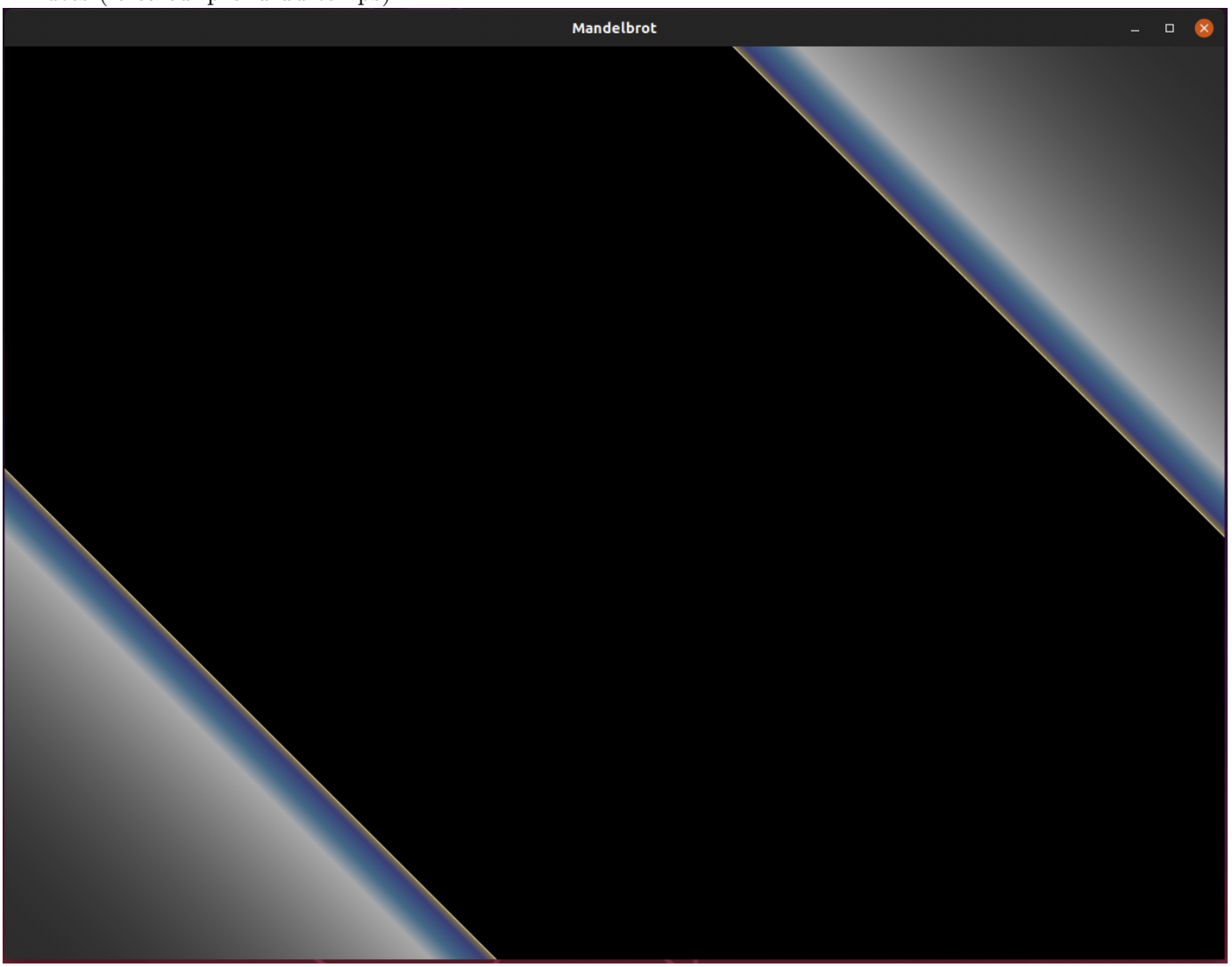
Comme pour le TP précédent, on va utiliser git pour la gestion de versions. Il vous faut donc vous reporter aux consignes du précédent TP. Le lien vers le devoir sur la classroom est le suivant : lien. Il vous faut recréer une équipe et demander la création d'un dépôt.

1.2 Exécuter le projet du dépôt

Pour compiler et exécuter votre programme, il faut passer par l'onglet gradle à droite.



— pour les tests il faut cliquer deux fois sur `mandelbrot-***` -> `Tasks` -> `verification` -> `test`.
— pour exécuter l'application qui est censé afficher la fractale de Mandelbrot, il faut cliquer deux fois sur `mandelbrot-***` -> `application` -> `run`. Vous devriez obtenir l'affichage suivant au bout de quelques minutes (le calcul prend du temps).

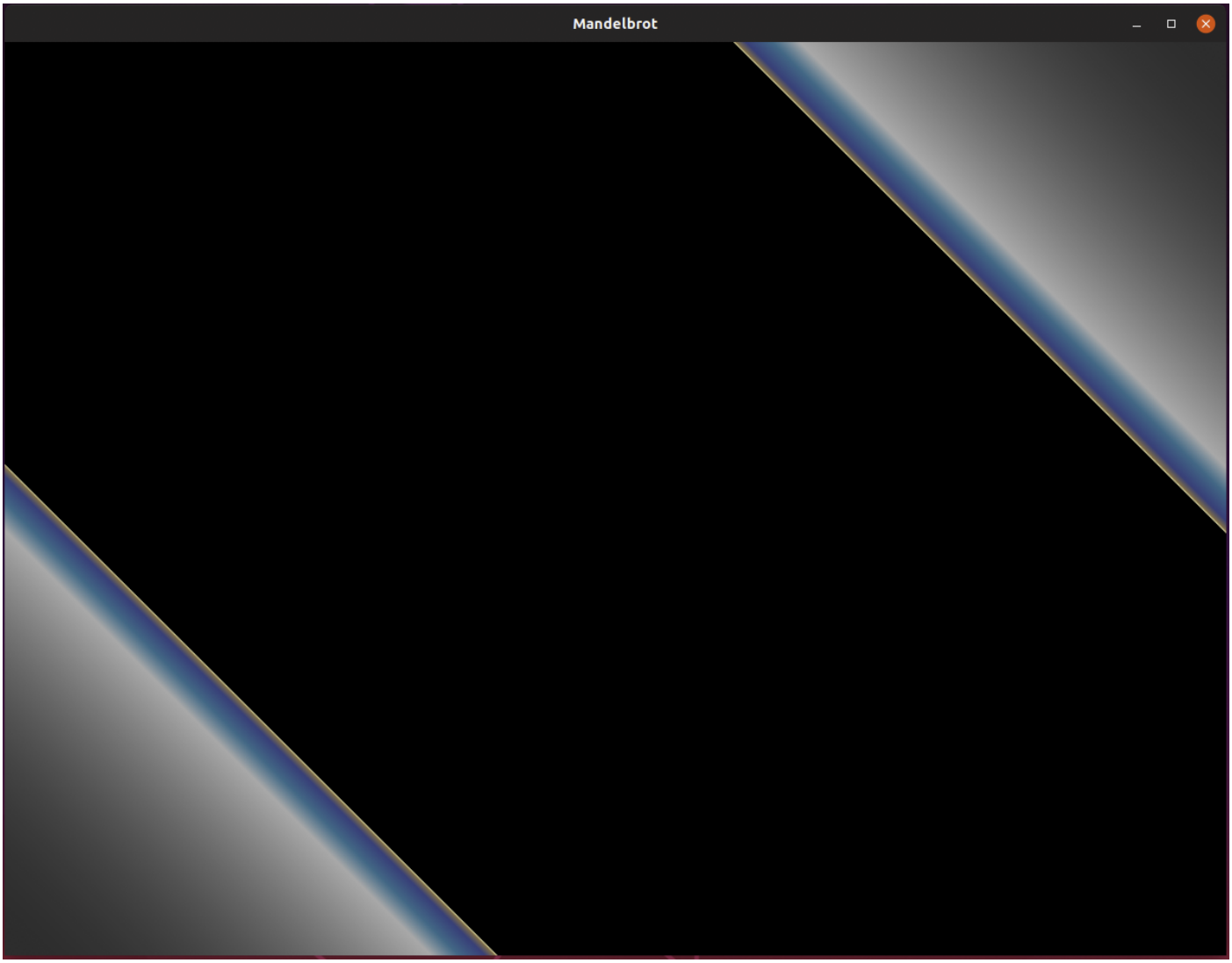


1.3 Consignes pour le début du TP

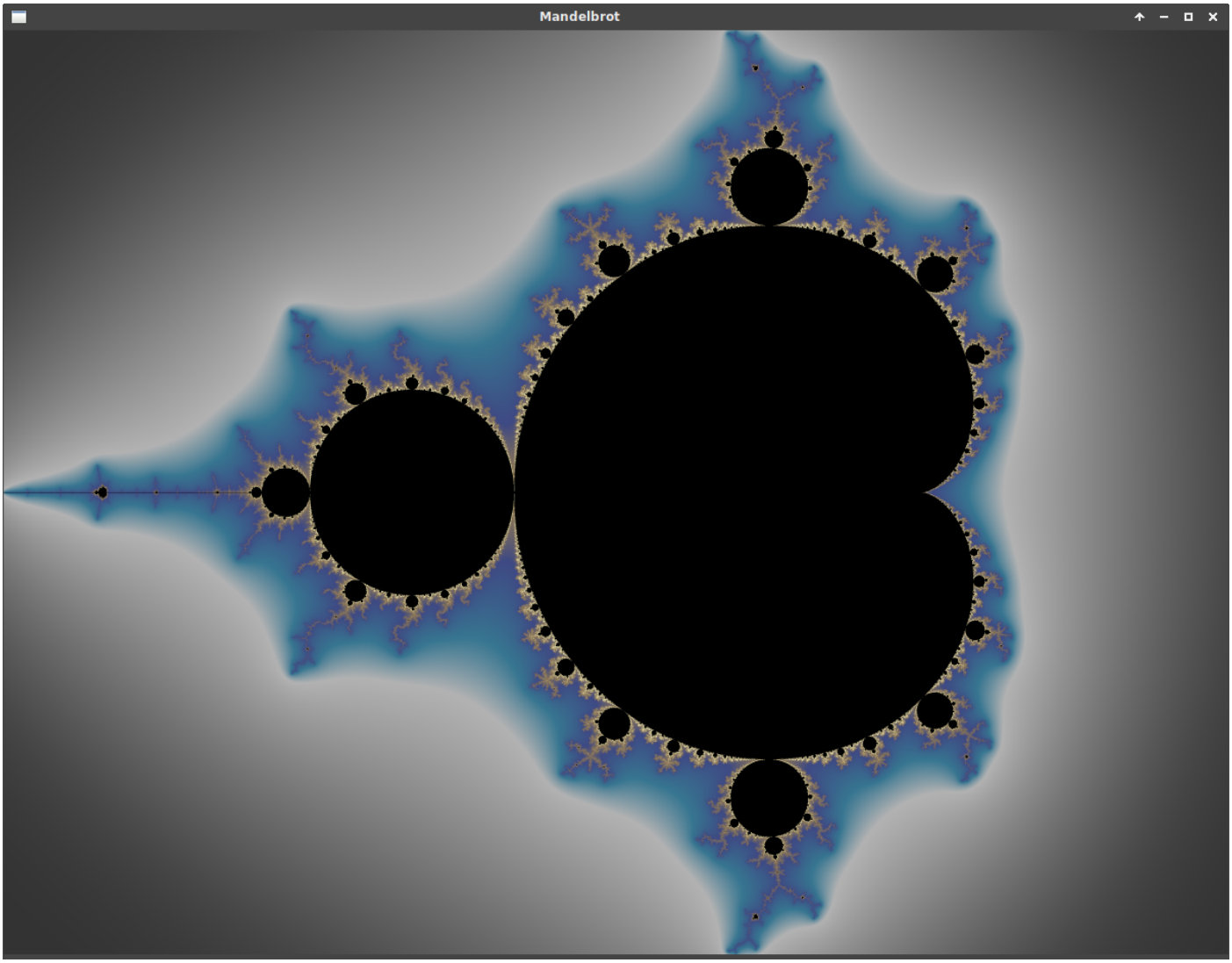
Modifiez le fichier `README.md`. Mettez votre nom, votre **numéro de groupe** ainsi que le nom et le **numéro de groupe** de votre éventuel co-équipier. Faites un `commit` avec pour message "inscription d'un membre de l'équipe", puis un `push`.

2 Affichage de Mandelbrot

On cherche à corriger la classe `Complex.java` afin d'afficher correctement l'ensemble de Mandelbrot. Vous ne devez modifier que la classe `Complex.java` (sauf pour les tâches optionnelles). La version de base du projet produit l'affichage suivant :



L'objectif du TP est d'obtenir l'affichage correct suivant :



Toutes les méthodes/classes/constructeurs/initialisation des constantes sont erronées à l'exception de la méthode `int hashCode()`.

2.1 Pourquoi des tests ?

Les calculs nécessaires à l'affichage de l'ensemble de Mandelbrot prennent plusieurs minutes alors que des tests unitaires peuvent se lancer en quelques secondes. Il est donc bien plus efficace de tester les méthodes et de ne lancer le calcul pour l'affichage que lorsque tous les tests sont passés avec succès. De plus, le résultats des tests avec les valeurs attendues donne une bien meilleure indication des erreurs que l'affichage de l'ensemble de Mandelbrot qui est dur à analyser.

2.2 Tâche 1 : corriger les méthodes pour lesquelles les tests existent

Pour cette tâche, vous devez corriger les constructeurs, méthodes et des initialisations des constantes afin qu'ils passent les tests déjà écrits dans la classe `ComplexTest` se trouvant dans le répertoire `src/test/java/mandelbrot/` :

— `Complex(double real, double imaginary)` (constructeur)
— `double getImaginary()` (méthode)
— `double getReal()` (méthode)

— `static Complex ZERO` (constante)
— `static Complex ONE` (constante)
— `static Complex I` (constante)
— `Complex negate()` (méthode)
— `Complex reciprocal()` (méthode)
— `Complex divide(Complex divisor)` (méthode)
— `Complex conjugate()` (méthode)
— `static Complex rotation(double radians)` (méthode)

Pour trouver et corriger les erreurs dans le programme, vous devrez lancer les tests. Vous devez faire un commit à chaque fois que vous corrigez un élément du programme : constructeur, méthode ou initialisation d'une constante. Si vous souhaitez utiliser dans vos test une fonction qui n'est pas testée, il est plus que conseillé de la tester au préalable. La documentation des assertions de Junit est disponible au lien suivant : org.junit.jupiter.api.Assertions.

En ce qui concerne les opérations sur les complexes, vous vous fierez à la section sur les opérations élémentaires de la page wikipedia sur les nombres complexes.

Vous pouvez vous référez à la documentation de la classe Complex pour la corriger.

2.2.1 Attention

Pour corriger ces méthodes, vous avez peut-être besoin de corriger d'autres méthodes.

2.3 Tâche 2 : écrire les tests puis corriger les méthodes pour lesquelles les tests n'existent pas

Pour cette tâche, vous devez corriger les méthodes suivantes (si vous ne l'avez pas déjà fait) :
— `public static Complex real(double real)`
— `public Complex add(Complex addend)`
— `Complex subtract(Complex subtrahend)`
— `Complex multiply(Complex factor)`
— `double squaredModulus()`
— `double modulus()`
— `Complex pow(int p)`
— `Complex scale(double lambda)`

Pour trouver et corriger les erreurs dans le programme vous devrez écrire des méthodes de test dans la classe `ComplexTest` en vous inspirant des tests déjà écrits pour les autres méthodes. Vous devez faire un commit à chaque fois que vous corrigez un méthode.

3 Petit rappel sur les tests

3.1 Tests unitaires

— Tester seulement une unité du programme : classe, méthodes, ...
— Vérifier un comportement :
  — cas normaux
  — cas limites
  — cas anormaux

3.2 Assertions utiles en JUnit

— `assertTrue(boolean condition)` : vérifie que `condition` est vraie.
— `assertFalse(boolean condition)` : vérifie que `condition` est faux.

— `assertEquals(expected, actual)` : vérifie que `expected` est égal à `actual`
égal : `equals` pour les objets et `==` pour les types primitifs.
— `assertEquals(double expected, double actual, double delta)` : vérifie que $|expected - actual| \leq delta$
— `assertNull(Object object)` : vérifie que la référence est `null`
— `assertNotNull(Object object)` : vérifie que la référence **n'**est **pas** `null`
— `assertSame(Object expected, Object actual)` : vérifie que les deux objets sont les mêmes (même référence).
— `assertArrayEquals(Object[] expected, Object[] actual)` : vérifie si les deux tableaux contiennent les même éléments dans le même ordre.
— `fail()` : échoue toujours

3.2.1 Message

Pour toutes les assertions, il est possible d'ajouter un `message` en premier argument qui permet d'identifier l'assertion.

4 Tâches optionnelles

Quelques tâches possibles mais optionnelles (pouvant entraîner un bonus à la note) :
— Ajouter dans l'interface graphique une manière pour l'utilisateur de définir la région à afficher.
— Ajouter dans l'interface graphique une manière pour l'utilisateur de personnaliser les couleurs d'affichage de l'ensemble.