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

Domaine Sciences et Technologies
Licence informatique

Programmation 2 : TP 4
Code UE : SIN3U02

Affichage de l'ensemble de Mandelbrot

1 Introduction

On va travailler sur ce TP sur l'affichage de l'ensemble de Mandelbrot. Pour cela, on va utiliser un code pré-existant. Malheureusement la classe `Complex` de ce code a été codée avec les pieds par un stagiaire et est donc bourrée d'erreurs. Le but du TP sera donc de corriger la classe `Complex` en s'aidant de tests unitaires.

1.1 Critères d'évaluation

Ce TP sera évalué et comptera pour 10% de la note finale de l'unité d'enseignement de programmation 2. Vous serez évalué sur :

— **La propreté du code :** comme indiqué dans le chapitre 2 du cours, il est important de programmer proprement. Des répétitions de code trop visibles ou des noms mal choisis vous pénaliseront.
— **La correction du code :** le but du TP étant de corriger du code incorrect, la correction du code sera un critère primordial lors de l'évaluation.
— **Les commit/push effectués :** il vous faudra travailler en continu avec git et faire des push/commit le plus régulièrement possible. Un projet ayant très peu de push/commit effectués juste avant la date limite sera considéré comme suspicieux et noté en conséquence.

1.2 Gestion de versions

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 la classroom est le suivant : lien.

Ce TP est à faire en deux séances de TP (celles ayant lieu les semaines du 5 au 9 novembre et du 12 au 16 novembre). Vous pouvez travailler (effectuer des push) sur ce TP jusqu'à la séance de TP de la semaine du 19 au 23 novembre (dimanche 18 novembre pour Aix et mardi 20 novembre pour Luminy dernier délai).

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`.

1.4 Maven

Ce projet est compilé grâce Maven qui est un moteur de production pour Java. Lorsque vous importez votre projet il faut donc choisir le *template* maven et accepter d'ajouter le projet en tant que projet Maven (*add as maven project*).

1.5 Comment exécuter le projet sur votre machine personnelle

La configuration conseillée pour faire le TP chez vous est :

— JDK 10 d'Oracle
— IntelliJ de Jetbrain
— maven
— git

1.6 Configuration pour exécuter le projet sur votre machine personnelle

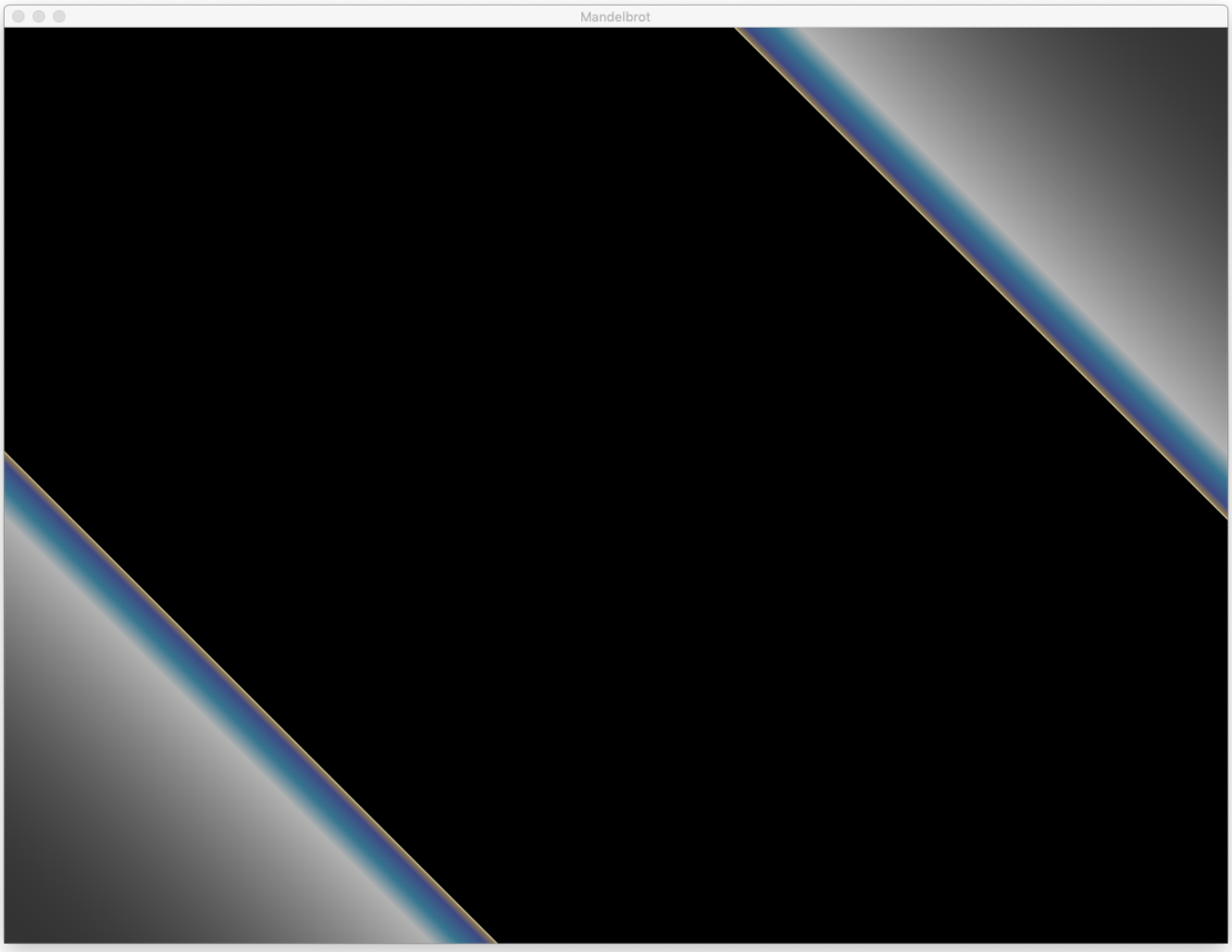
Si vous n'avez pas téléchargé le JDK 10 d'Oracle en début d'année, on vous conseille de :

— télécharger et installer (soit via dual boot ou par machine virtuelle via VirtualBox) une version récente (18.10 conseillée) d'ubuntu.
— installer *openjdk 11* avec la ligne de commande `sudo apt install openjdk-11-jdk`
— installer *git* avec la ligne de commande `sudo apt install git`
— installer *maven* avec la ligne de commande `sudo apt install maven`
— installer *snapd* avec la ligne de commande `sudo apt install snapd`
— installer *IntelliJ* avec la ligne de commande `sudo install intellij-idea-ultimate --classic`
— changer le fichier *pom.xml* du projet par le fichier pom.xml
— pour lancer l'interface graphique, il vous suffit d'utiliser la ligne de commande suivante : `mvn compile exec:java` dans le répertoire du projet.
— vous pouvez aussi compiler dans *IntelliJ* en éditant une configuration de compilation maven avec un maven goal `compile exec:java`
— vous pouvez lancer les tests directement dans l'IDE.

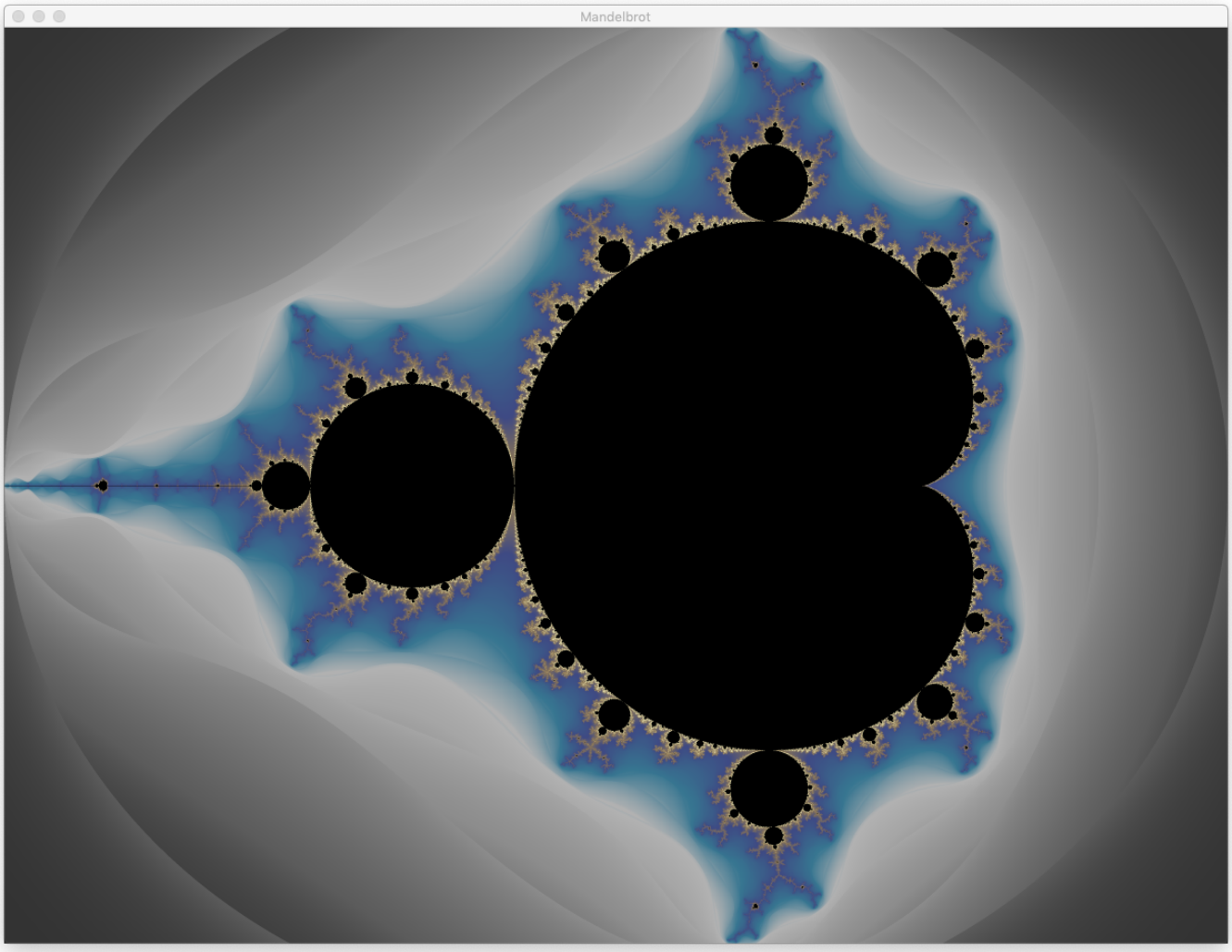
Ces consignes sont à suivre uniquement dans le cas où vous souhaiterez faire fonctionner le code sur votre machine et n'avez pas télécharger le JDK 10 d'oracle en début d'année.

2 Affichage de Mandelbrot

On cherche à corriger le code de 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). Le code de base du projet produit l'affichage suivant :



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



Toutes les méthodes/classes/constructeurs/initialisation des constantes sont erronées à l'exception du code 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, il vous devez corriger le code des 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)
— `String toString()` (méthode)

Pour trouver et corriger les erreurs dans le code, vous devrez lancer les tests de la classe `ComplexTest` en cliquant sur le triangle à gauche de la déclaration de la classe. Vous devez faire un commit à chaque fois que vous corrigez un élément du code : 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.

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

Pour cette tâche, vous devez corriger le code des méthodes suivantes :
— `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)`
— `public boolean equals(Object o)`

Pour trouver et corriger les erreurs dans le code 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 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.