Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

*Filière Informatique 2020-2021*

# Intégration des nombres complexes et des Unum en Java avec COJAC

Classe I3

---

## Cahier des charges

---

Travail de bachelor

Version : 1.2


Student : Cédric Tâche

Professor : Frédéric Bapst

Expert : Baptiste Wicht

## Table des versions

| Version | Date | Auteur | Description |
|---|---|---|---|
| 0.1 | 01.06.2021 | Cédric Tâche | Création |
| 1.0 | 07.06.2021 | Cédric Tâche | Description plus précise du projet et des objectifs |
| 1.1 | 11.06.2021 | Cédric Tâche | Précision des tâches<br>Correction de la planification<br>Ajout d'objectifs secondaires |
| 1.2 | 14.06.2021 | Cédric Tâche | Ajout de code pour les nombres complexes<br>Correction de la mise en page |

# Table des matières

# 1 Introduction

La plupart des langages de programmation offrent des capacités similaires pour stocker des nombres et effectuer des calculs. Ces langages permettent, entre autres, d'utiliser des nombres réels.

Ces nombres réels ont tout de même des limitations. Ils sont limités au domaine du réel. De plus, les nombres à virgule flottante sont souvent inexacts. Par conséquent, lorsque les erreurs s'accumulent, le résultat d'un calcul peut être très éloigné de la réponse exacte.

Pourtant, d'autres alternatives existent. Les nombres complexes sont couramment utilisés en mathématique et en physique. Ils peuvent être utilisés pour simplifier des réponses utilisant des racines de nombres négatifs ou pour combiner deux quantités réelles telles que la tension et l'intensité en électricité. Quant à eux, les nombres réels peuvent aussi être représentés différemment. Les **universal numbers** (**unums**) sont une représentation alternative possible dont la dernière version est expliquée dans l'article *Beating Floating Point at its Own Game : Posit Arithmetic* [2].

# 2 Contexte

En Java, aucun type ni aucune classe dans le JDK permet de gérer des nombres complexes ou des unum, mais il est possible de l'implémenter soi-même. Des librairies pouvant gérer ces éléments peuvent déjà exister.

La première solution pour ajouter ces fonctionnalités et de créer de nouvelles classes : une classe pour les nombres complexes et une classe pour les unums. Ensuite, il faut écrire le code en utilisant directement ces classes. Pour changer le type de calculs effectué (ex : nombre complexe $\rightarrow$ nombre réel), il faut changer le code source de l'application.

COJAC [1] est une librairie Java permettant de modifier les capacités arithmétiques d'un programme Java sans en modifier le code. Elle utilise l'API d'instrumentation et peut transformer les classes et méthodes au runtime pour changer le type de calcul effectué. Ainsi, pour changer le type de calculs effectué (ex : nombre réel $\rightarrow$ nombre complexe), il faut seulement changer l'argument donné à COJAC [1] lors du démarrage de l'application. Ceci ne demande aucune modification dans l'application de l'utilisateur.

# 3 Objectifs

Le but de ce projet est d'ajouter deux nouvelles fonctionnalités à COJAC [1]. COJAC devra permettre de remplacer automatiquement les nombres à virgules flottantes par deux nouveaux types numériques.

### 3.1 Intégration des nombres complexes

Une option de COJAC permettra de changer le comportement des calculs dans l'application de l'utilisateur. Les nombres à virgules flottantes (*float* et *double*) seront remplacés, au runtime, par des nombres complexes. Les opérations arithmétiques telles que l'addition, la soustration, etc. devront être adaptées. De plus, les méthodes souvent utilisées de la librairie standard devront également pouvoir fonctionner. Par exemple, la méthode *Math.sqrt* devra permettre de retourner la racine carrée d'un nombre négatif.

Voici un exemple sans les nombres complexes :

```
double val = Math.sqrt(-1); // = NaN
val = val * val; // = NaN;
```

Avec les nombres complexes, on obtient le résultat suivant :

```
double val = Math.sqrt(-1); // = i
val = val * val; // = -1;
```

### 3.2 Intégration des unums

Une option de COJAC permettra de changer le format de stockage et de calculs des nombres réels. Les unums seront utilisés à la place de la virgule flottante. Par conséquent, les opérations arithmétiques devront être redéfinies pour fonctionner avec ce nouveau format de stockage. Il faudra probablement utiliser JNI pour accéder à une librairie C/C++ permettant d'utiliser les unums, mais d'autres approches restent possibles.

### 3.3 Démonstration des deux fonctionnalités

Des programmes de démonstrations seront réalisés pour montrer ces deux fonctionnalités. Ces démonstrations doivent montrer l'utilité et les avantages de cette approche.

# 4 Objectifs secondaires

D'autres ajouts de fonctionnalités ou modifications permettraient d'améliorer ce projet.

## 4.1 Mise à jour des librairies

COJAC [1] utilise plusieurs librairies dont les versions sont désormais obsolètes. Il vaut mieux mettre à jour les versions avant de rencontrer des problèmes à cause de versions trop anciennes. Cependant, COJAC [1] devra garantir une compatibilité pour Java 8+.

## 4.2 Tests de performance

Lorsque les fonctionnalités de remplacement des nombres à virgule flottante par des nombres complexes et des unums, il restera encore un aspect inconnu qui est pourtant important pour décider de l'utilité de cette fonctionnalité : les performances. Pour cette raison, des tests de performance peuvent aussi être ajoutés pour tester l'efficacité de l'implémentation.

## 4.3 Documentation et promotion

COJAC possède une documentation pour l'utiliser et des vidéos pour expliquer l'utilité de certaines fonctionnalités. Il serait possible de documenter les nouvelles fonctionnalités ajoutées, de réaliser une vidéo pour montrer l'utilité de ces vidéos ou encore de compléter la documentation actuelle.

## 4.4 Comparaison des approches wrappers et behaviours

Deux approches sont possibles pour implémenter de nouvelles fonctionnalités dans COJAC :

- Behaviour : les opérations sur les floats et les doubles sont simplement remplacées par un appel de méthode. Ainsi, il est possible de changer le comportement de ceux-ci. Dans ce projet, il serait possible de mettre la partie réelle et la partie imaginaire dans un double et de modifier les opérations qui les utilisent.
- Wrapper : les floats et les doubles peuvent être remplacés par un objet (wrapper). Ce qui permet d'ajouter plus d'éléments dans le wrapper.

### 4.5 Améliorations diverses

Toute autre amélioration à la base de code existante est aussi le bienvenu. Voici quelques exemples d'améliorations qui pourraient être effectuées :

- Ajout d'un CI sur GitLab pour vérifier les tests et compiler le JAR.
- Ajout d'un logger pour améliorer la gestion des logs.
- Améliorer l'architecture du projet.
- Améliorer la documentation du code.
- Faire les TODO présents dans le code.

## 5 Tâches

Cette section détaille les tâches qui devront être effectuées pour réaliser chacun des objectifs définis précédemment.

### 5.1 Intégration des nombres complexes

Les tâches suivantes seront effectuées pour réaliser cet objectif :

- Analyse des implémentations existantes.
- Analyse de COJAC et de comment intégrer un nouveau type.
- Analyse des nombres complexes.
- Concevoir l'intégration des nombres complexes.
- Intégrer des nombres complexes.
- Tester l'intégration des nombres complexes.

### 5.2 Intégration des unums

Les tâches suivantes seront effectuées pour réaliser cet objectif :

- Analyse des implémentations existantes.
- Analyse de COJAC et de comment intégrer un nouveau type.

- Analyse des unums.
- Choix d'une variante d'unum à implémenter.
- Concevoir l'intégration des unums.
- Intégrer des unums.
- Tester l'intégration des unums.

### 5.3 Démonstration

Les tâches suivantes seront effectuées pour réaliser cet objectif :

- Concevoir les démonstrations.
- Réaliser les démonstrations.

### 5.4 Mise à jour des librairies

Les tâches suivantes seront effectuées pour réaliser cet objectif :

- Chercher les mises à jour de chaque librairie.
- Regarder les changements avec la nouvelle version.
- Modifier la version et adapter le code.
- Tester les modifications.

### 5.5 Tests de performance

Les tâches suivantes seront effectuées pour réaliser cet objectif :

- Analyser les tests de performance actuels.
- Réaliser les tests de performance.
- Synthétiser les tests de performance.

# 6 Planification

La figure 1 montre le diagramme de Gantt avec la liste des jalons et des tâches durant le projet. Ce travail dure du 31 mai au 16 juillet 2021.

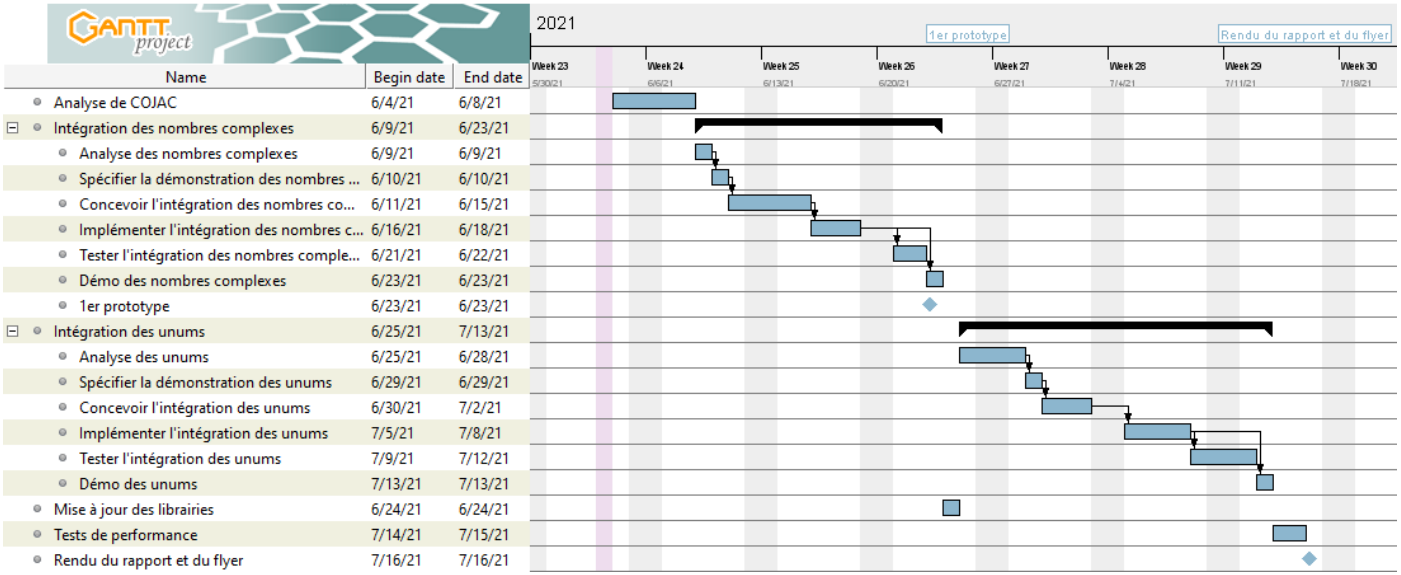


FIGURE 1 – Diagramme de Gantt

D'autres dates importantes sont également prévues plus tard dans l'année pour présenter ce projet.

| **Activité** | **Date limite** |
|---|---|
| Rendu du poster | 27.08.2021 |
| Exposition du travail de bachelor | 03.09.2021 |
| Défense orale | 06-08.09.2021 |

# 7 Références

[1] COJAC. Cojac - boosting arithmetic capabilities of java numbers. `https://github.com/Cojac/Cojac`, 2019.

[2] Gustafson and Yonemoto. Beating floating point at its own game : Posit arithmetic. *Supercomput. Front. Innov. : Int. J.*, 4(2) :71–86, June 2017. ISSN 2409-6008. doi : 10.14529/jsfi170206. URL `https://doi.org/10.14529/jsfi170206`.