CAHIER DE CONCEPTION GÉNÉRALE

Projet Maths-Infos 2017-2018

Version: 1.4
Auteur: Romain JACQUIEZ, Antoine BERENGUER et Théo Hipault

ISEN Toulon - Yncrea
Maison du Numérique et de l'Innovation
Place Georges Pompidou
83000 Toulon

Description du document

Type	Version	Confidentialité
Cahier de conception générale	1.4	Usage externe

	Nom	Fonction	Date	Visa
Rédacteur	Romain JACQUIEZ Antoine BERENGUER Théo HIPAULT	Rédaction	19/12/17 au 21/12/17	
Vérificateur	Romain JACQUIEZ Théo HIPAULT	Corrections	19/12/17 au 21/12/17	
Approbateur	M.THELOT et M.FRESSEL	Validation	22/12/17	

Destinataire	Fonction	Organisme
Client	Lecture	ISEN

Révisions du document

Version	Date	Rédacteur	Modifications
1.0	18/07/2016	FMC	Mise en forme
1.1	19/12/17	RJ, TH	Modification de la forme
1.2	19/12/17	RJ, AB	Début écriture
1.3	20/12/17	RJ	Modification schémas
1.4	20/12/17	TH	Révision finale

Sommaire

Index des illustrations

Index des tables

REFERENCES

Référence	Description	Nom
[1]		
[2]		

DEFINITIONS

Sans objet

ABBREVIATIONS

ISEN : Institut Supérieur de l'Electronique et du Numérique

1. INTRODUCTION

Ce programme sera réalisé en langage Java dans le cadre du projet Maths-Info de l'année 2017. Il aura pour but de calculer facilement la transformée de Fourier rapide (FFT) d'un signal donné et d'afficher ces résultats. Le signal traité par le programme pourra être réel ou complexe selon le choix de l'utilisateur, et il pourra être entré manuellement ou choisi parmi des signaux de test selon le besoin. Il permettra également d'effectuer la transformée inverse d'un signal afin de retrouver la donnée d'origine.

Exemple de fonctionnement : Le programme se lance en console. Au démarrage, un menu demandera à l'utilisateur de choisir parmi 5 signaux types ou bien d'importer les valeurs souhaitées à partir d'un fichier .csv.
La sélection se fait en entrant le numéro correspondant à son choix dans la console.

Signaux types : Après avoir choisi un signal type, le calcul sera automatiquement effectué et affiché dans la console. Trois autres fenêtres s'affichent, ce sont les graphes représentant le signal obtenu (module en fonction des valeurs échantillonnées) après la FFT et l'iFFT. Le programme se termine une fois que ces fenêtres sont fermées.

Signal personnalisé : Pour importer le signal désiré, une fenêtre s'affiche (sélecteur de fichier classique). Naviguer jusqu'au fichier.csv souhaité et valider, les résultats du calcul s'affichent en console et les graphes dans des fenêtres séparées.

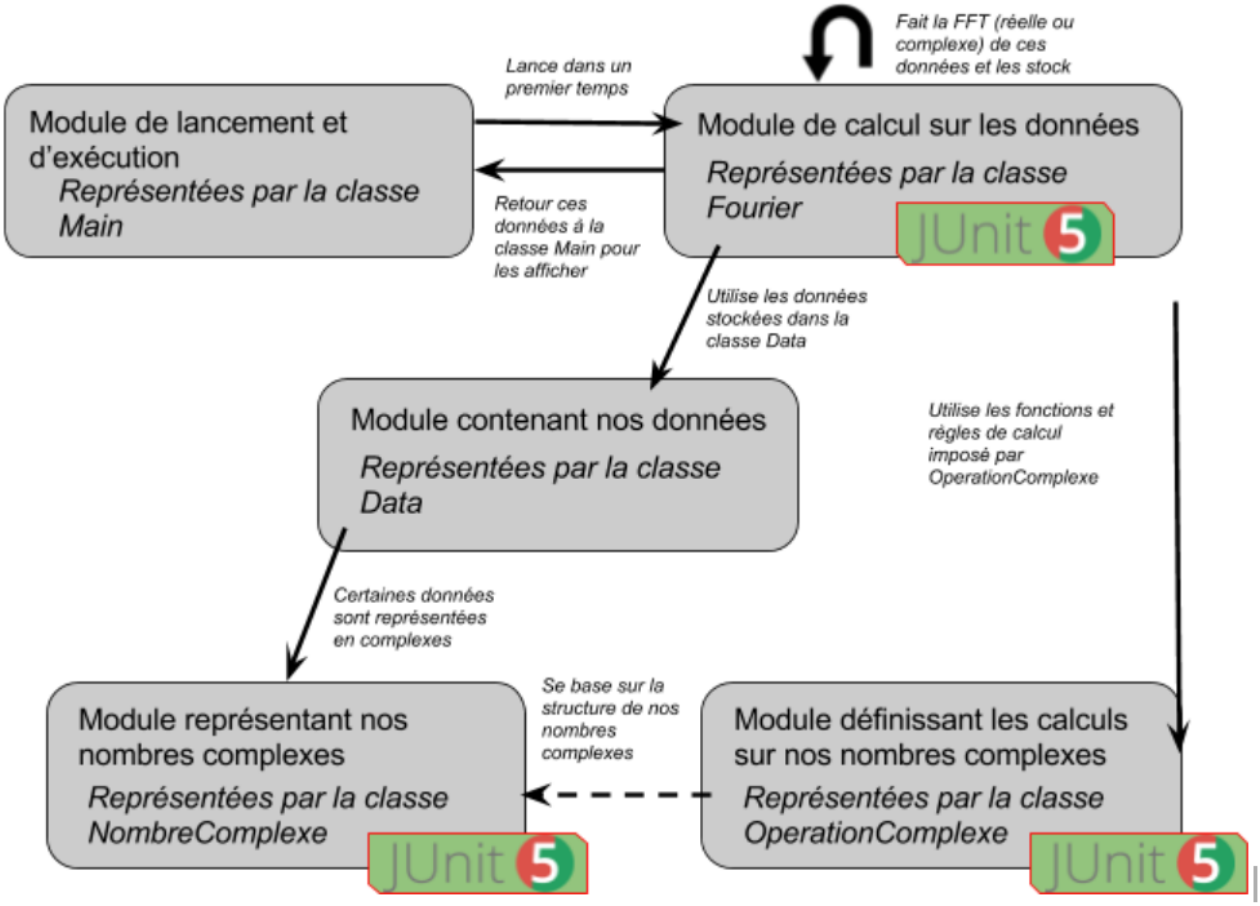
Note : Le fichier .csv doit être mis en forme de la façon suivante :
- Les nombres décimaux doivent être écrits avec un point "."
- Première colonne du fichier : partie réelle du signal à traiter.
- Deuxième colonne : partie imaginaire du signal à traiter (sans "i").
- La taille des colonnes doit être conforme à la taille du signal indiquée dans le programme. Si le signal dépasse la taille, les valeurs en trop ne seront pas traitées.

Ce fichier .csv sera modifié par le programme de sorte à enregistrer les résultats obtenus. La troisième colonne du fichier est laissée vide, les colonnes 4 et 5 contiennent respectivement la partie réelle et imaginaire du signal obtenu par la FFT. La sixième colonne est laissée vide, les colonnes 7 et 8 contiennent respectivement la partie réelle et imaginaire du signal obtenu par l'iFFT.

2. MODULES FONCTIONNELS

A) Architecture des modules

Architecture du programme et utilité de chaque module.



B) Données utilisées par chaque module

Classe Main :

Cette classe initialise un objet Fourier, nécessaire pour effectuer nos transformées.

```
Fourier monFourier = new Fourier(16);
```

Ici, notre objet Fourier est créé et initialisé avec la valeur 16. Celle-ci représente la taille de notre tableau de données (ainsi que celui de retour où sont stockées les valeurs transformées).

La classe Main manipule aussi des tableaux de type NombreComplexe

```
NombreComplexe monRetour[] = new NombreComplexe[monFourier.getTaille()];
NombreComplexe monRetourInverse[] = new NombreComplexe[monFourier.getTaille()];
```

Ce sont ces tableaux de retour, que nous affichons à l'utilisateur du programme.

Classe Fourier :

Cette classe contient deux attributs qu'elle manipule :

```
private int taille;
public Data donnee;
```

Elle manipule un entier de type *int* qui représente, comme dit plus haut, la taille de nos tableaux de retours et de données. Mais aussi un objet de type Data, nommé ici donnee, qui contient nos données représentatives de notre signal.

Classe Data :

Cette classe manipule deux tableaux de données :

```
private double Sreel[];
private NombreComplexe Scomplexe[];
```

Le premier tableau est la représentation de notre signal réel. C'est à dire les valeurs de ce signal échantillonné à une certaine fréquence.
Si le signal est complexe, on l'échantillonne et on stock ses valeurs dans le tableau de nombres complexes.

La classe a possibilité de manipuler un chemin d'accès où se trouve un un fichier (.csv).
Effectivement, le programme peut prendre les données d'un signal d'un fichier .csv, les traiter, et les retourner dans ce même fichier.

Classe NombreComplexe :

Cette classe manipule une partie réel et une partie imaginaire, dans le but de définir l'architecture et les transformation que peut subir un nombre complexe.

```
private double partieReelle;
private double partieImaginaire;
```

Elle manipule aussi une valeur :

```
private double margeErreur;
```

Cette valeur permet de définir un seuil, à partir duquel, un *double* est considéré comme étant égal à 0. Cela permet de simplifier les calculs ainsi que l'affichage des retours et des données pour l'utilisateur.

Classe OperationComplexe :

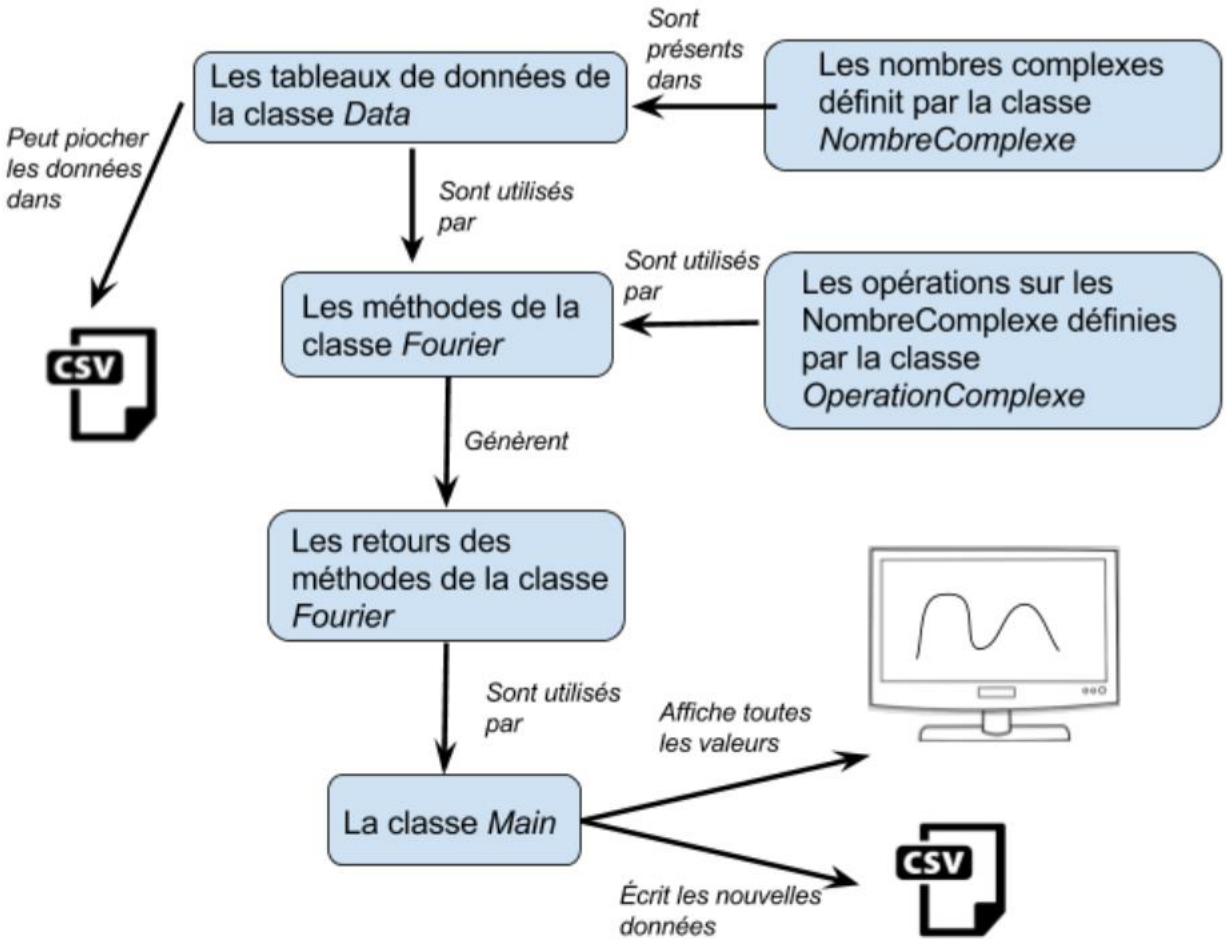
Cette classe définit les opérations possibles que notre programme peut effectuer sur des nombres complexes de type *NombreComplexe*.

Elle manipule donc des nombres complexes de types *NombreComplexe* mais aussi des nombres de type *double*. Effectivement, il est important de fixer ce que doit faire programme lorsque l'on doit, par exemple multiplier des nombres complexes entre eux, mais il est tout aussi important de le faire pour des opérations entre des double et des nombres complexes. C'est le cas quand on veut faire une FFT sur des nombres réels de type *double*.

La classe Fourier utilise les opérations définies dans cette classe

C) Échange de données entre modules

Détail des transferts de données entre chaque module



3. STRUCTURES DE DONNÉES

A) Définition des structures de données

Déclaration de chaque structure de donnée + proto

Classe Fourier :

```
private int taille

public Data donnee

public Fourier(int taille)

public NombreComplexe[] FFTr(int signal)

private NombreComplexe[] recursiveFFTr(double tab[], int taille)

public NombreComplexe[] iFFT(NombreComplexe transformee[], int taille)

public NombreComplexe[] FFTc(int signal)

private NombreComplexe[] recursiveFFTc(NombreComplexe tab[], int taille)

public int getTaille()

public void setTaille(int taille)
```

Classe Data :

```
public static final int UN = 1

public static final int SIN = 2

public static final int COS = 3

public static final int DIRAC = 4

public static final int CSV = 5;

private double Sreel[]

private NombreComplexe Scomplexe[]

private String csvFile;

public Data(int signal, int taille)
```

```
private void initUn(int taille)

private void initSin(int taille)

private void initCos(int taille)

private void initDirac(int taille)

private boolean initCSV(int taille)

public double[] getSreel()

public NombreComplexe[] getScomplexe()

private String[][] CSVparser(String path, int taille)

public boolean CSVWrite(NombreComplexe donnees[])

private String menuOuverture(String type)
```

Classe NombreComplexe :

```
private double partieReelle

private double partieImaginaire

private double margeErreur

public NombreComplexe(double partieReelle, double
partieImaginaire, double margeErreur)

public NombreComplexe(double partieReelle, double
partieImaginaire)

public NombreComplexe()

public double getPartieReelle()

public void setPartieReelle(double partieReelle)

public double getPartieImaginaire()

public void setPartieImaginaire(double partieImaginaire)

public void setNombreComplexe(NombreComplexe z)

public NombreComplexe conjugue()
```

```
public double module()
```

```
public double argument()
```

```
public String toString()
```

```
public String formeExponentielle()
```

Classe OperationComplexe :

```
private OperationComplexe()
```

```
public static NombreComplexe inverse(NombreComplexe z)
```

```
public static NombreComplexe additionner(NombreComplexe z1,
NombreComplexe z2)
```

```
public static NombreComplexe additionner(NombreComplexe z1, double
x)
```

```
public static NombreComplexe soustraire(NombreComplexe z1,
NombreComplexe z2)
```

```
public static NombreComplexe soustraire(NombreComplexe z, double
x)
```

```
public static NombreComplexe soustraire(double x, NombreComplexe
z)
```

```
public static NombreComplexe multiplier(NombreComplexe z1,
NombreComplexe z2)
```

```
public static NombreComplexe multiplier(NombreComplexe z, double
x)
```

```
public static NombreComplexe diviser(NombreComplexe z1,
NombreComplexe z2)
```

```
public static NombreComplexe diviser(NombreComplexe z, double x)
```

```
public static NombreComplexe expoVersAlgebrique(double module,
double argument)
```

```
public static NombreComplexe puissance(NombreComplexe z, double x)
```

Classe Main :

```
public static void main(String args[])
```

```
public static void genereChartFourier(NombreComplexe retour[],
boolean inverse)
```

```
public static void genereChartSignal(Fourier monFourier)
```

B) Visibilité des données

En Java, les attributs sont la plupart du temps accessibles via les accesseurs et mutateurs (ou en anglais, getters et setters) et sont donc le plus souvent privés ou protégés.

Dans la classe Fourier, l'attribut Data est public afin de pouvoir accéder directement à ses méthodes depuis l'objet Fourier.

Dans la classe Data, les attributs UN, SIN, COS et DIRAC sont déclarées comme publiques, statiques et finaux afin de pouvoir les utiliser comme des constantes.

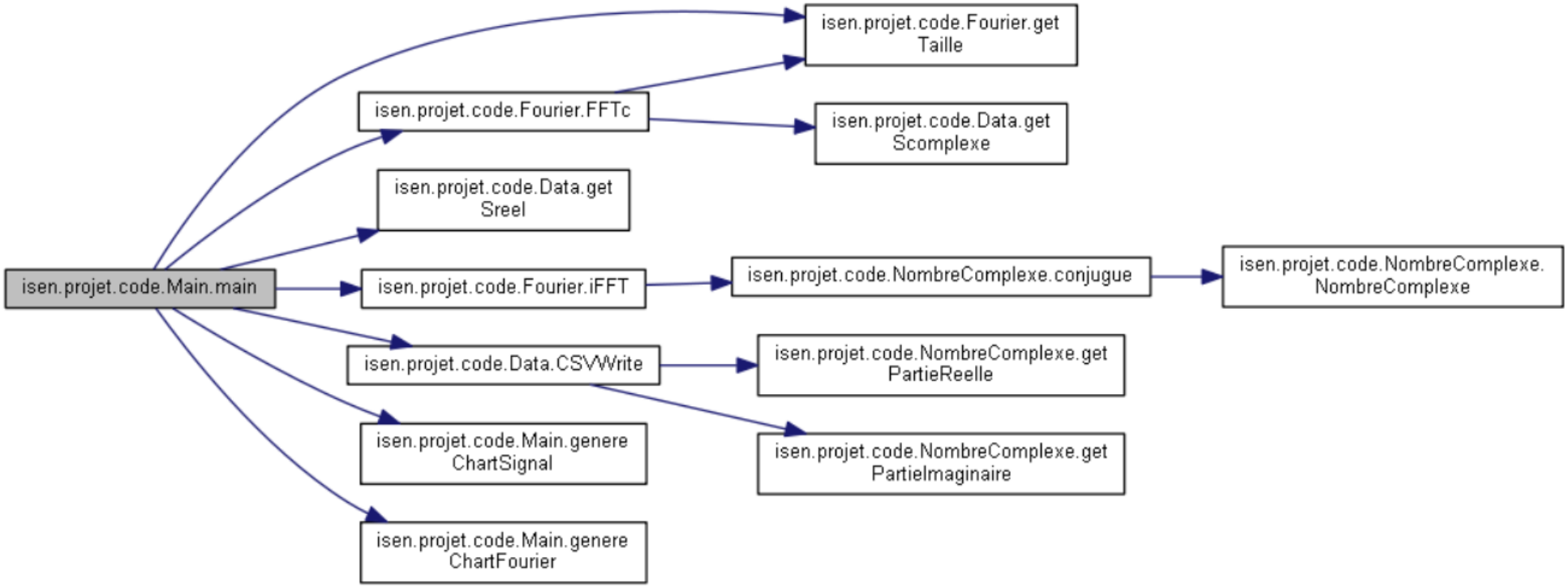
La classe OperationComplexe sert uniquement à fournir des méthodes et n'a pas besoin d'être initialisée (similaire à la classe Math fournie avec Java). Java n'autorise pas de déclarer une classe comme statique (sinon, c'est ce qui aurait été fait), elle est donc déclarée de façon à ressembler au mieux à une classe statique :

- La classe est déclarée comme finale
- Le constructeur est déclaré comme privé pour éviter l'instanciation de la classe
- Toutes les méthodes sont déclarées comme statiques

4. ARBRE DES CLASSES ET MÉTHODES ET FLUX DES DONNÉES

A) Arbre d'appel et flux de données

(Schéma des différents blocs et interactions entre ces blocs + arbre d'appel des méthodes)



B) Description des classes et méthodes

Prototypes des classes et méthodes des modules :

Classe Fourier :

```
public Fourier(int taille)
Ce constructeur modifie l'attribut taille de la classe
```

```
public NombreComplexe[] FFTr(int signal)
Cette fonction a pour but de calculer la FFT (Fast Fourier Transformation) de notre échantillon de valeurs réelles de notre signal, passé en paramètre.
```

```
private NombreComplexe[] recursiveFFTr(double tab[], int taille)
Cette fonction a pour but de calculer la FFT (Fast Fourier Transformation) de notre échantillon de valeurs réelles de notre signal, représenté par un tableau double. Elle est appelée par la fonction FFTr et permet de "descendre et remonter" dans la récursion, nécessaire pour traiter notre tableau de données.
```

```
public NombreComplexe[] iFFT(NombreComplexe transformee[], int taille)
Cette fonction a pour but de calculer la FFT (Fast Fourier Transformation) inverse de nos valeurs, stockés dans un tableau de nombres NombreComplexe. Elle est appelée sur un tableau de retour de la fonction FFTr ou FFTc.
```

```
public NombreComplexe[] FFTc(int signal)
Cette fonction a pour but de calculer la FFT (Fast Fourier Transformation) de notre échantillon de valeurs complexes de notre signal, passé en paramètre.
```

```
private NombreComplexe[] recursiveFFTc(NombreComplexe tab[], int taille)
Cette fonction a pour but de calculer la FFT (Fast Fourier Transformation) de notre échantillon de valeurs complexes de notre signal, représenté par un tableau de NombreComplexe. Elle est appelée par la fonction FFTc et permet de "descendre et remonter" dans la récursion, nécessaire pour traiter notre tableau de données.
```

```
public int getTaille()
```

```
Une fois appelée, cette fonction renvoie l'attribut taille de la classe Fourier. La fonction est public et permet de retourner taille qui est private
```

```
public void setTaille(int taille)
Une fois appelée, cette fonction modifie l'attribut taille de la classe Fourier. La fonction est public et permet de modifier l'attribut taille qui est private
```

Classe Data :

```
public Data(int signal, int taille)
Ce constructeur modifie les attributs de la classe (les tableaux) en fonction du signal passé en paramètre ainsi que de la taille.
/!\ Cette taille doit être une puissance de 2.
```

```
private void initUn(int taille)
Appelée par le constructeur de la classe, cette fonction initialise les tableaux de la classe par des 1
```

```
private void initSin(int taille)
Appelée par le constructeur de la classe, cette fonction initialise les tableaux de la classe par un échantillonnage des valeurs de sin().
```

```
private void initCos(int taille)
Appelée par le constructeur de la classe, cette fonction initialise les tableaux de la classe par un échantillonnage des valeurs de cos().
```

```
private void initDirac(int taille)
Appelée par le constructeur de la classe, cette fonction initialise les tableaux de la classe par une impulsion de Dirac
```

```
private boolean initCSV(int taille)
Appelée par le constructeur de la classe, cette fonction initialise les tableaux de la classe par la première colonne du tableur .CSV sélectionné
```

```
public double[] getSreel()
```

Une fois appelée, cette fonction renvoie l'attribut Sreel de la classe Data. La fonction est public et permet de retourner Sreel qui est private

```
public NombreComplexe[] getScomplexe()
```
Une fois appelée, cette fonction renvoie l'attribut Scomplexe de la classe Data. La fonction est public et permet de retourner Scomplexe qui est private

```
private String[][] CSVparser(String path, int taille)
```
Appelée par la fonction initCSV, elle lit le fichier afin de transformer le contenu du fichier en un tableau 2D de String

```
public boolean CSVWrite(NombreComplexe donnees[])
```
Appelée dans le main, cette fonction écrit le tableau de nombre complexes fourni en paramètre dans le fichier CSV d'adresse stockée comme attribut.

```
private String menuOuverture(String type)
```
Appelée dans initCSV, cette fonction affiche un menu qui permet à l'utilisateur de sélectionner le fichier CSV à importer

Classe NombreComplexe :

```
public NombreComplexe(double partieReelle, double partieImaginaire, double margeErreur)
```
Ce constructeur modifie les attributs de la classe (la partie réelle et imaginaire et la marge d'erreur) en fonction des valeurs passées en paramètre.

```
public NombreComplexe(double partieReelle, double partieImaginaire)
```
Ce constructeur modifie les attributs de la classe (la partie réelle et imaginaire) en fonction des valeurs passées en paramètre et initialise la marge d'erreur à 10^-10.

```
public NombreComplexe()
```
Ce constructeur modifie les attributs partie réelle et imaginaire de la classe, les mets à 0 et initialise la marge d'erreur à 10^-10.

```
public double getPartieReelle()
```
Une fois appelée, cette fonction renvoie l'attribut partieReelle de la classe NombreComplexe. La fonction est public et permet de retourner partieReelle qui est private

```
public void setPartieReelle(double partieReelle)
```
Une fois appelée, cette fonction modifie l'attribut partieReelle de la classe NombreComplexe. La fonction est public et permet de modifier l'attribut partieReelle qui est private

```
public double getPartieImaginaire()
```
Une fois appelée, cette fonction renvoie l'attribut partieImaginaire de la classe NombreComplexe. La fonction est public et permet de retourner partieImaginaire qui est private

```
public void setPartieImaginaire(double partieImaginaire)
```
Une fois appelée, cette fonction modifie l'attribut partieImaginaire de la classe NombreComplexe. La fonction est public et permet de modifier l'attribut partieImaginaire qui est private

```
public void setNombreComplexe(NombreComplexe z)
```

Une fois appelée, cette fonction modifie l'attribut partieImaginaire et partieReelle de la classe NombreComplexe, en fonction du nombre complexe passé en paramètre.

`public NombreComplexe conjugue()`
Appelée, cette fonction renvoie un nouveau nombre complexe qui est le conjugué du nombre complexe qui appelle cette fonction.

`public double module()`
Appelée, cette fonction renvoie le module du nombre complexe qui appelle cette fonction.

`public double argument()`
Appelée, cette fonction renvoie l'argument du nombre complexe qui appelle cette fonction.

`public String toString()`
Override de la fonction toString de java.lang.object Cette fonction permet de définir comment va être affiché notre objet, si on "l'appelle" dans une chaîne de caractères.

`public String formeExponentielle()`
Cette fonction permet de définir comment va être affiché notre objet, si on veut sa forme exponentielle. Cette forme exponentielle est représentée dans un String

Classe OperationComplexe :

`private OperationComplexe()`
Ce constructeur ne fait absolument rien et ne peut pas être appelé. La classe ne peut pas être instanciée.

`public static NombreComplexe inverse(NombreComplexe z)`
Cette fonction permet de calculer l'inverse d'un nombre complexe passé en paramètre.

`public static NombreComplexe additionner(NombreComplexe z1, NombreComplexe z2)`
Cette fonction permet de calculer l'addition entre deux nombre complexes passés en paramètre.

```
public static NombreComplexe additionner(NombreComplexe z1, double x)
Cette fonction permet de calculer l'addition entre un nombre complexe et un double passés en paramètre.

public static NombreComplexe soustraire(NombreComplexe z1, NombreComplexe z2)
Cette fonction permet de calculer la soustraction entre deux nombres complexes passés en paramètre.

public static NombreComplexe soustraire(NombreComplexe z, double x)
Cette fonction permet de calculer la soustraction entre un nombre complexe et un double passés en paramètre.

public static NombreComplexe soustraire(double x, NombreComplexe z)
Cette fonction permet de calculer la soustraction entre un double et un nombre complexe passés en paramètre.

public static NombreComplexe multiplier(NombreComplexe z1, NombreComplexe z2)
Cette fonction permet de calculer la multiplication entre deux nombres complexes passés en paramètre.

public static NombreComplexe multiplier(NombreComplexe z, double x)
Cette fonction permet de calculer la multiplication entre un nombre complexe et un double passés en paramètre.

public static NombreComplexe diviser(NombreComplexe z1, NombreComplexe z2)
Cette fonction permet de calculer la divsion entre deux nombres complexes passés en paramètre.

public static NombreComplexe diviser(NombreComplexe z, double x)
Cette fonction permet de calculer la division entre un nombre complexe et un double passés en paramètre.

public static NombreComplexe expoVersAlgebrique(double module, double argument)
Cette fonction permet de transformer un nombre complexe en forme exponentielle, en forme algebrique. La forme exponentielle est passée en paramètre grâce au module et à l'argument du nombre complexe.
```

```
public static NombreComplexe puissance(NombreComplexe z, double x)
Cette fonction permet de calculer un nombre complexe à la puissance d'un réel.
```

Classe Main :

```
public static void main(String args[])
Cette fonction lance et fait tourner le programme
```

```
public static void genereChartFourier(NombreComplexe retour[], boolean inverse)
Cette fonction permet de créer une frame qui contient le graph d'un signal, dont les données sont passées dans un tableau, lui même passé en paramètre. Les titres des axes varient si on choisit d'afficher notre signal FFT ou iFFT
```

```
public static void genereChartSignal(Fourier monFourier)
Cette fonction permet de créer une frame qui contient le graph de notre signal d'entré.
```