Chapitre 01

Prés-requis en programmation objet et en Java

L'objectif de ce chapitre de cours est de faire un **RAPPEL** des principes de la programmation objet et des bases du langage Java qui correspond normalement aux prérequis du cours NFP121.

1. Introduction

Dans le cursus de CNAM le cours NFP121 fait suite aux cours NFA 031 et NFA 032 dans lesquels sont présentés les concepts de la programmation objet et l'apprentissage du langage JAVA.

Ces cours constituent donc le prérequis au cours NFP121.

Dans le paragraphe §2 suivant, je vais faire un rappel sur les concepts de la programmation objet, forme d'introduction au cours NFP121.

Dans le paragraphe §3 suivant, je vais balayer rapidement les cours de NFA 031 qui constituent donc une très grande partie de ce prérequis, afin que vous soyez capable d'identifier, les notions qui nécessiteraient d'être approfondies.

Cet approfondissement pourra être réalisée sous différentes formes :

- Un fort travail personnel pour rejoindre le niveau requis par une lecture précise de ces cours et des exercices que vous ferez chez vous.
- Un soutien par son partenaire de binôme lors des TD et TP qui aurait le niveau requis.
- Un soutien de ma part en dernier recours.

Au sujet du langage de modélisation UML, il est indispensable de savoir écrire des diagrammes de classes. Normalement, cela fait parti du prérequis au cours NFP 121 mais ce n'est généralement majoritairement pas le cas. Un cours sera consacré à l'approche méthodologique de la conception d'un programme Java dans lequel sera présenté les diagrammes de classe.

2. Rappels des concepts de la POO

Il est indispensable de connaître parfaitement les concepts de base de la Programmation Orientée Objet suivants :

- La classe
- L'objet : les attributs
- L'objet : les méthodes
- Les liens d'associations : Composition et Agrégation
- L'héritage des classes
- La classe abstraite et l'Interface

2.1. La classe

La classe est le premier pilier des concepts de la programmation (le deuxième étant l'héritage).

Une classe est un cadre de définition. Il correspond à une unité de programmation dont le rôle est de définir la structure des objets qui pourront être créés en instanciant la classe (new). Les objets sont appelées instances de classe.

Classe = type de données

La classe a un nom unique. Tous les objets créés à partir d'une classe sont du type de la classe. En résumé, le nom de la classe est le type des objets.

La classe contient :

- Les attributs de la classe (la mémoire de l'objet).
- Les méthodes de la classe (le code). =,traitements

2.2. L'objet

Chaque objet créé représente une partie de la mémoire du programme informatique.

Si un attribut d'un objet de type A est un autre objet de type B alors on crée une association (ou liaison) entre A et B. Ce lien de dépendance entre A et B peut être de différentes natures, voir §2.4.

Un attribut peut être :

- public
- private
- static public
- static private

2.3. Les méthodes

Les méthodes sont le code de votre programme informatique.
Les méthodes d'un objet s'appliquent sur les attributs de l'objet.

Une méthode peut être :

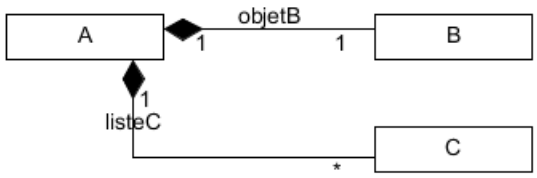
- public
- private
- static public
- static private

Une méthode statique ne peut utiliser que les attributs statiques de la classe.
Les autres méthodes s'appellent des méthodes objets.

Une méthode objet peut utiliser les attributs statiques de la classe.

2.4. Associations

Le lien de Composition est un lien très fort (le plus fort de tous). Il lie deux instances d'objet entre eux. On dit que A est composé de B et C (plusieurs instances de C) :

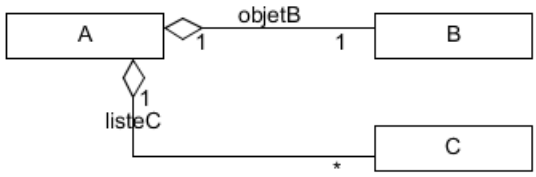


Composition = Sous-type

B est C sont des sous-objets de A. Ils font parti intégrante de sa définition.
Quand A est créé, B et C sont créés.
Quand A est détruit, B et C sont détruits.

Exemple : A = Camion B = Moteur C = Roue

Le lien d'Agrégation est aussi un lien de « composition » mais moins fort que le précédent car la destruction de A n'entraine pas la destruction de B et C.



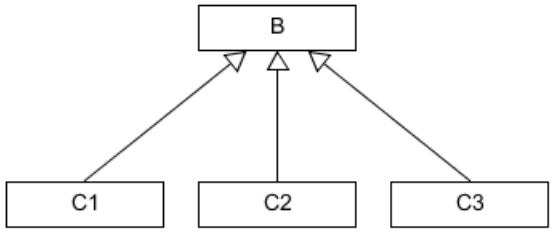
A est bien composé de B et C (plusieurs instances de C).

On ne sait pas dans cette description comment B et C sont créés. B est par exemple passé en paramètre du constructeur de A ou en utilisant le setteur de B. Une instance de C est passé en paramètre d'une méthode de A qui permet d'ajouter une instance de C à A.

Exemple : A = Entreprise B = Patron C = Camion

2.5. Héritage

L'héritage de classe est le lien qui justifie toute l'approche des langages à objet.
On dit que les classes C1, C2 et C3 héritent de B :

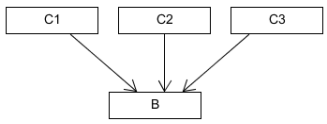


C1 "est un" B.

B = VehiculeRoulant → double vitesseMax ; roule()
C1 = Voiture
C2 = Velo → guidon pedaler() roule() {…… ; super.roule() ; }

On utilise l'héritage de classe simple (à opposer à la notion de classe abstraite que nous verrons plus loin) pour 3 raisons :

- **Factoriser** du code. B contient les méthodes communes à C1, C2 et C3. Cela correspond à la notion de couches de librairies dans la programmation classique (non objet) :

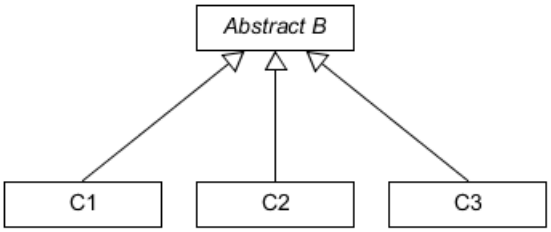


- **Etendre** (extends) les propriétés de la classe B. Cela correspond à la notion d'héritage la plus courante. On ajoute dans les classes C1, C2 et C3 de nouvelles méthodes et de nouveaux attributs qui s'ajoutent à la classe B sans impacter cette dernière.
- **Spécialiser** la classe B en redéfinissant une partie de ses méthodes dans les classes C1, C2 et C3.

Il est à noter que ces 3 raisons ne sont pas exclusives. Elles peuvent se cumuler.

2.6. Classe abstraite et interface

La classe Abstraite est une classe comme une autre (tout ce qui a été dit précédemment sur l'héritage s'applique ici) mais il n'est pas possible de créer une instance d'une classe abstraite (new).



Il faut créer des instances de C1, C2 et C3 qui héritent de B.

Toutes les méthodes abstraites (sans code) de B doivent impérativement être définies dans les classes C1, C2 et C3 (sinon erreur de compilation).

Ainsi, une méthode dans le programme qui utilise en paramètre une classe B utilisera les méthodes abstraites sans savoir à priori quelles seront les méthodes réelles de C1, C2 ou C3 qui seront utilisées dans l'algorithme de cette méthode.

On parle de traitement générique.

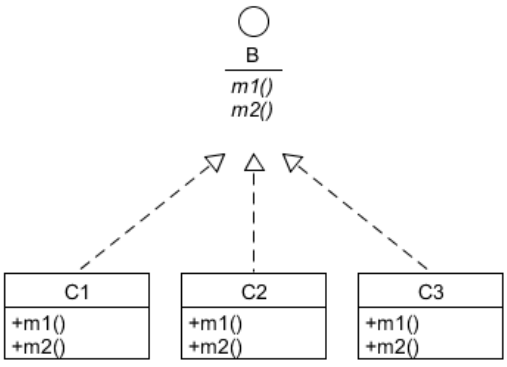
Cette notion est primordiale dans la conception d'un programme objet. Elle permet d'abstraire sa conception et donc d'être générique dans sa conception.

Par définition, un DP est un concept générique.

Une classe abstraite peut contenir à la fois des méthodes abstraites et des méthodes réelles étant donné qu'elle contient ses propres attributs et donc ses propres traitements.

Dans le cas où la classe abstraite ne contient aucun attribut d'objet et ne contient que des méthodes abstraites, on parle d' **Interface**.

La notion d'interface est symbolisée en UML ainsi :



On dit que les classes C1, C2 et C3 **implémentent** l'interface B.

Par définition, les classes C1, C2 et C3 doivent obligatoirement implémenter toutes les méthodes de l'interface.

Une classe peut implémenter plusieurs interfaces.

2.7. Association vs héritage

Souvent, il n'est pas facile de faire son choix entre un lien d'association et un lien d'héritage.

Il faut pouvoir répondre aux questions :

- "X est composé de Y" ➔ Composition
- "X est un Y" ➔ Héritage
- "X s'appuie sur Y" ou "X utilise Y" ➔ Agrégation

Nous verrons comment on peut implémenter un lien d'héritage par un lien de composition.

Beaucoup de langages ne font pas d'héritage multiple. Il est donc nécessaire de remplacer l'héritage par des liens de compositions ou par des interfaces car une classe peut implémenter plusieurs interfaces en même temps.

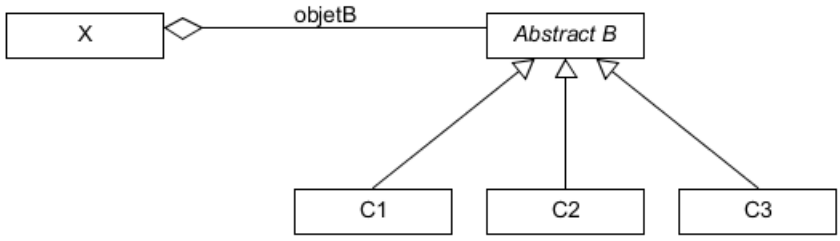
2.8. Les techniques d'association

Il existe de nombreuses façons de réaliser une association basée sur une classe abstraite ou une interface (dans les exemples qui suivent on utilise une classe abstraite mais cela est tout à fait valable sur une Interface).

Une association explicite dans le programme :

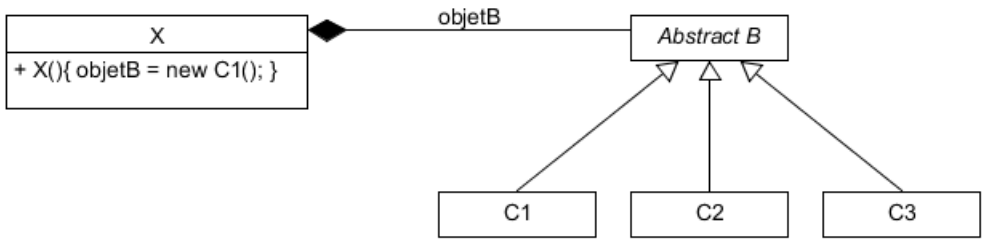
```
Constructeur de X :   X(B :b)

X x = new X(new C1())
```



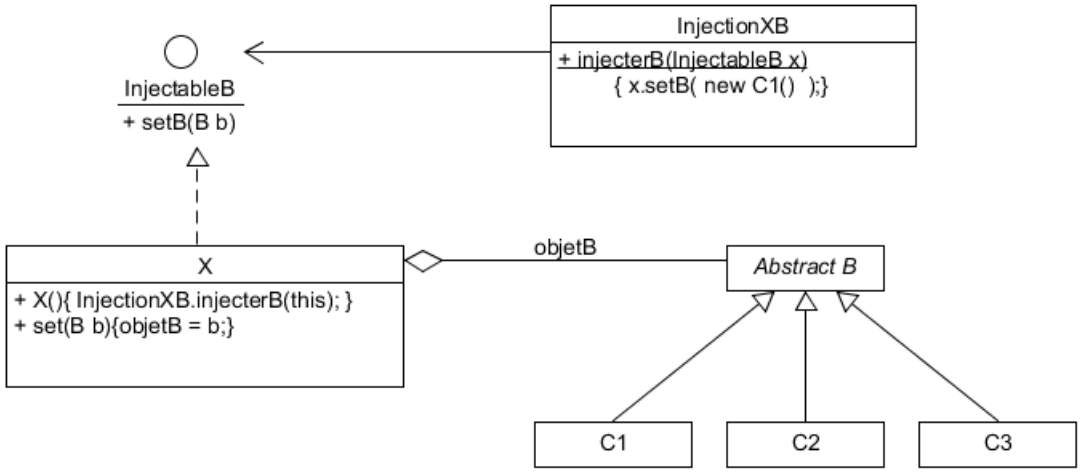
Une association explicite dans le constructeur :

```
X x = new X()
```



Une association implicite dans le constructeur. On parle d'injection de dépendance (nous verrons cela plus loin en détail) :

```
X x = new X()
```



Ici, la classe X délègue à une autre classe (l'injecteur de dépendance) de faire l'association entre lui-même (this) et une classe qui hérite de B (ici C1) à travers une méthode statique afin d'accentuer l'indépendance de X par rapport aux autres classes. De plus, grâce à l'utilisation de l'interface InjectableB, il n'existe pas de dépendance entre InjectionXB et X.

2.9. Les principes d'instanciation

Dans un programme informatique (qu'il soit réalisé avec un LOO ou non), il existe deux raisons (toujours revenir au besoin 😊) pour créer un objet.

Soit on crée un objet afin de l'utiliser (lui physiquement et pas un autre) dans tout ou une "partie" du programme. Soit on crée un objet parce qu'il est naturellement la décomposition d'un autre objet.

La deuxième raison ne pose pas de difficulté de compréhension et est directement la suite naturelle de la programmation structurée classique issue de la réflexion et du besoin de composer ces données.

La première raison, par contre, nécessite que l'on s'y arrête un moment.

A quel moment doit-on créer cet objet ? Bien avant son utilisation ou au moment de son utilisation ?

Comment accède-t-on à cet objet (comment on obtient sa référence) ? Par passage en paramètre dans tous les constructeurs et/ou méthodes traversés ? Globalement ?

A quel moment doit-on le détruire ? et le recréer ?

Comment accéder à cet objet depuis l'extérieur du programme (réseau) ?

S'il existe plusieurs occurrences de cet objet, comment peu-on gérer l'ensemble de ces occurrences ?

De plus, comparons deux techniques permettant de créer l'instance d'une classe :

```
TypeObjet obj = new TypeObjet(paramètres)
```

```
TypeObjet objet = TypeObjet.getInstance(paramètres);
// (cela n'existe pas mais cela revient à cela : TypeObjet.new(paramètres)
```

avec

```
static public TypeObjet getInstance(paramètres) {
…….
   return new TypeObjet(paramètres); }
```

Les deux techniques sont strictement équivalentes.

```
OU

TypeObjet objet = f.getInstance(paramètres);
// TypeObjet.new(paramètres)
```

La première technique ne répond pas aux questions posées alors que la deuxième technique permet de répondre à toutes ces questions puisque l'on crée un point d'entrée commun d'instanciation sur laquelle on a la main.

Il faudrait donc toujours créer des objets de référence avec la deuxième technique.

Cela signifie que conceptuellement il faut toujours avoir en tête cette deuxième façon de faire.

Cette approche se traduit par la mise en œuvre de différents Designs Patterns et composition de Designs Patterns permettant sa création et son utilisation (Singleton, Factory, Builder, Injection, Stratégie, Proxy, Interface, ….)

Cela démontre que les DP ne sont pas avant tout une démarche d'architecture, de réutilisabilité, ou de factorisation mais aussi une démarche de développement au plus bas niveau.

3. Rappels sur le langage Java

3.1. Concepts de la programmation

http://coursjava.fr/SITE_NFA031/Cours/Site/NFA031-Chapitre-02_ConceptsDeLaProgrammation.pdf

L'objectif de ce chapitre de cours est d'avoir un aperçu des concepts de base de la programmation en général et une introduction à la Programmation Orientée Objet ou POO (on parle aussi de Programmation Objet)

Remarque : Les paragraphes §9 à §11 correspond à ce que l'on vient de voir dans le paragraphe précédent mais en plus précis.

3.2. Le programme en JAVA

http://coursjava.fr/SITE_NFA031/Cours/Site/NFA031-Chapitre-03_ProgrammeJava.pdf

L'objectif de ce chapitre de cours est d'écrire ses premiers programmes JAVA. Mais aussi d'introduire et d'anticiper sur la structuration d'un programme JAVA.

Nous aborderons donc des notions qui seront détaillées par la suite.

Votre objectif sera d'installer sur votre PC un environnement de programmation Java et de vous exercer car la programmation est une discipline pratique.

3.3. Structure et syntaxe de Java

http://coursjava.fr/SITE_NFA031/Cours/Site/NFA031-Chapitre-05_SyntaxeJava.pdf

L'objectif de ce chapitre de cours est de préciser la syntaxe du langage Java (classes, attributs, méthodes, structures de contrôle, variables, paramètres).

3.4. La création des objets en Java

http://coursjava.fr/SITE_NFA031/Cours/Site/NFA031-Chapitre-06_CreationObjetJava.pdf

L'objectif de ce chapitre de cours est de comprendre comment un objet est géré par le langage Java, et de mieux maitriser sa représentation en mémoire. Cette vision est essentielle car elle permet de mieux comprendre la dépendance physique entre les objets.

Remarque : ce cours n'aborde pas l'utilisation des méthodes **finalize** et **clone**.

3.5. Les chaînes de caractère en Java

http://coursjava.fr/SITE_NFA031/Cours/Site/NFA031-Chapitre-07-StringJava.pdf

L'objectif de ce chapitre de cours est de comprendre l'utilisation des classes **String**, **StringBuffer** et **StringTokenizer**.

3.6. Les concepts de base des collections

http://coursjava.fr/SITE_NFA031/Cours/Site/NFA031-Chapitre-08_CollectionsBase.pdf

L'objectif de ce chapitre de cours est de comprendre le concept de collections d'éléments en informatique et de découvrir l'utilisation de la classe prédéfinie Java **ArrayList**.

La suite de cours sur les collections est celui abordé dans le cours NFA 032 :
http://coursjava.fr/SITE_NFA032/Cours/Site/NFA032-Chapitre-03_CollectionsAvances.pdf
Nous reverrons et utiliserons ces concepts de collection avancés dans le cadre de l'étude des Factory (ou Fabrique)et Itérateur.

3.7. Les packages

http://coursjava.fr/SITE_NFA031/Cours/Site/NFA031-Chapitre-10_LesPackages.pdf

L'objectif de ce chapitre de cours est de décrire la notion de package en Java.

Avec l'utilisation des IDE, comme Eclipse, maitriser les contraintes de l'utilisation des packages en Java devient de moins en moins une priorité. On pourra venir sur ce concept en travaux pratiques si le besoin est nécessaire. Que cela ne vous empêche pas de lire quand même ce cours.

4. Configuration de vos postes

L'objectif est que vous soyez individuellement autonome sur vos ordinateurs personnels car il vous sera demandé de faire des exercices chez vous.

Vous aurez le choix de réaliser les TD sur les postes de travail de la salle de cours ou sur votre ordinateur portable personnel.

Vous travaillerez en binôme.

L'IDE utilisé pour programmer en Java est **Eclipse**.

Si vous avez déjà une version de Eclipse installée sur votre poste et que la version du JDK Java est supérieure à la version 8, vous pouvez garder votre version.

4.1. Installation du JDK Java Version 11

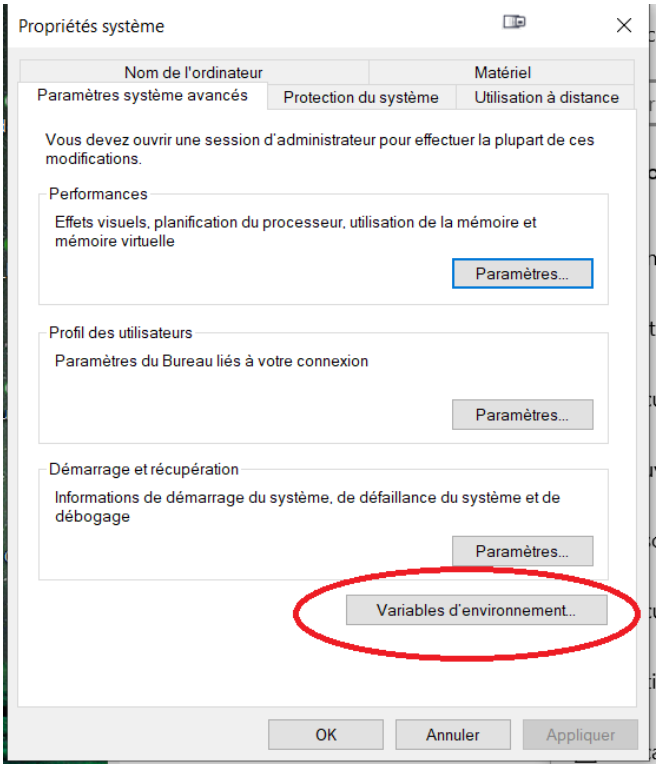
https://www.oracle.com/java/technologies/javase/jdk11-archive-downloads.html

152 Mo

Linux x64 Compressed Archive	173.46 MB	jdk-11.0.11_linux-x64_bin.tar.gz
macOS Installer	167.15 MB	jdk-11.0.11_osx-x64_bin.dmg
macOS Compressed Archive	167.67 MB	jdk-11.0.11_osx-x64_bin.tar.gz
Solaris SPARC Compressed Archive	184.51 MB	jdk-11.0.11_solaris-sparcv9_bin.tar.gz
Windows x64 Installer	152.05 MB	jdk-11.0.11_windows-x64_bin.exe
Windows x64 Compressed Archive	171.53 MB	jdk-11.0.11_windows-x64_bin.zip

Télécharger et exécuter.
Vérifier que vous avez le jdk est bien installé : C:\Program Files\Java\jdk-11.0.11

Dans les propriétés du système, modifier les variables d'environnement :

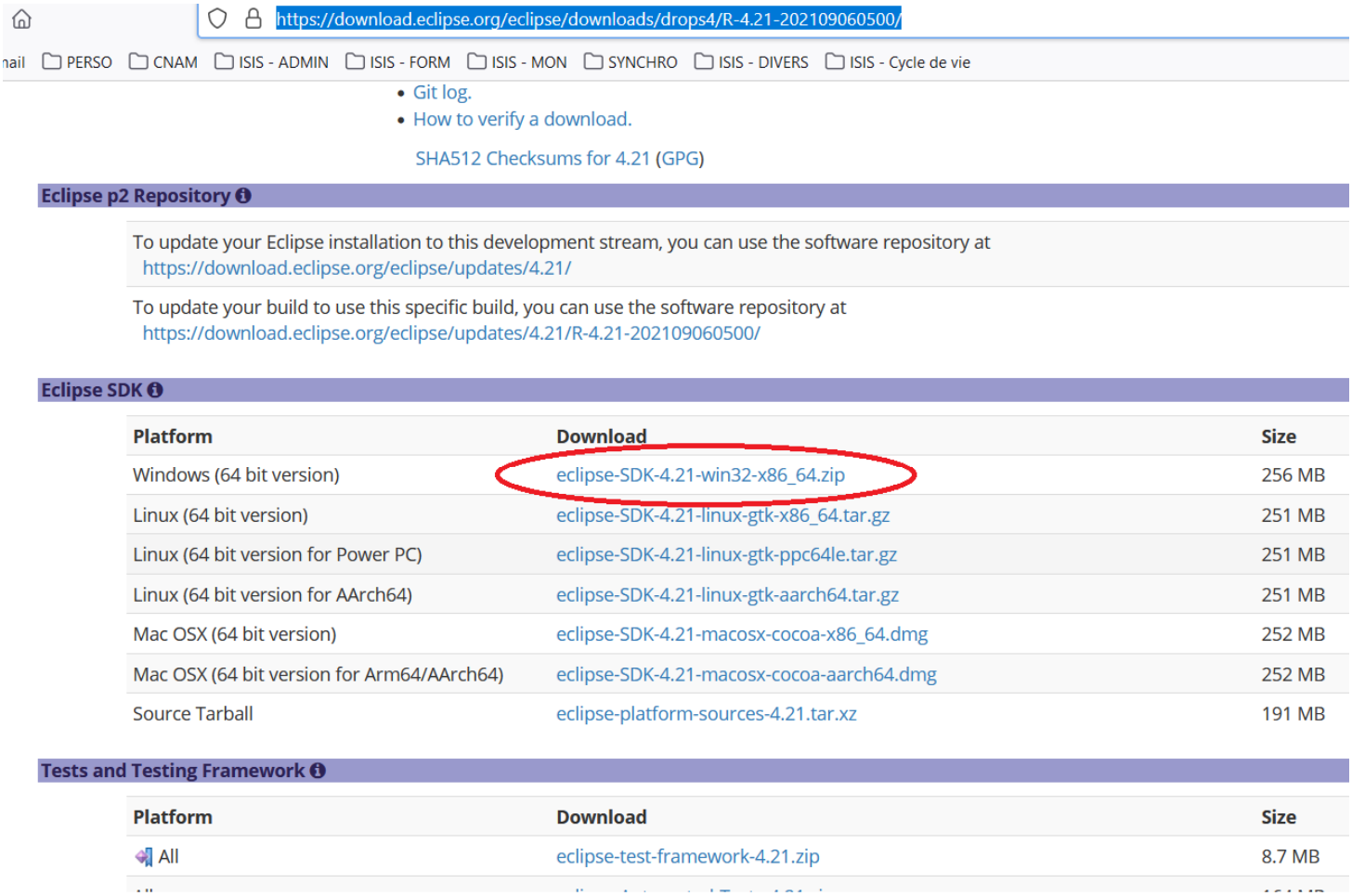


Ajouter dans la variable d'environnement **path** :
`C:\Program Files\Java\jdk-11.0.11\bin`

Créer la variable d'environnement **JAVA_HOME** avec
`C:\Program Files\Java\jdk-11.0.11`

4.2. Installation de Eclipse

Lien de téléchargement : (256Mo)
https://download.eclipse.org/eclipse/downloads/drops4/R-4.21-202109060500/



Télécharger et déziper dans un répertoire quelconque.
Pour Eclipse il n'existe pas d'exécutable d'installation.

Lancer l'exécutable : **eclipse.exe**

Si vous avez une erreur de lancement qui vous dit que la version 11 est requise alors modifier le fichier eclipse/**eclipse.ini** en ajoutant les 2 lignes surlignées en jaune :

```
-startup
plugins/org.eclipse.equinox.launcher_1.6.300.v20210813-1054.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.win32.win32.x86_64_1.2.300
.v20210828-0802
--launcher.defaultAction
openFile
--launcher.appendVmargs
-vm
C:\Program Files\Java\jdk-11.0.11\bin\javaw.exe
-vmargs
-Dosgi.requiredJavaVersion=11
-Dosgi.dataAreaRequiresExplicitInit=true
-Dorg.eclipse.swt.graphics.Resource.reportNonDisposed=true
-Xms256m
-Xmx2048m
--add-modules=ALL-SYSTEM
```

5. La documentation

La doc de l'API Java 11 :
https://docs.oracle.com/en/java/javase/11/docs/api/index.html

6. Travaux Dirigé : TD 1

Faire l'exercice 01 du site :
http://coursjava.fr/NFP121_Exercices.php?repertoire=Exercice01_Phrase