

# **TRAVAUX DIRIGES DE SIMULATION**

## **ENSTA BRETAGNE - PARTIE 0**

### **ANNEE 2019-2020**

Enseignant : Olivier VERRON (simuenstabretagne@gmail.com)

## **1. PREAMBULE**

*Cette partie du TD va vous conduire à apprendre à vous servir de 3 bibliothèques fournies et à utiliser dans le cadre des parties 1 et 2.*

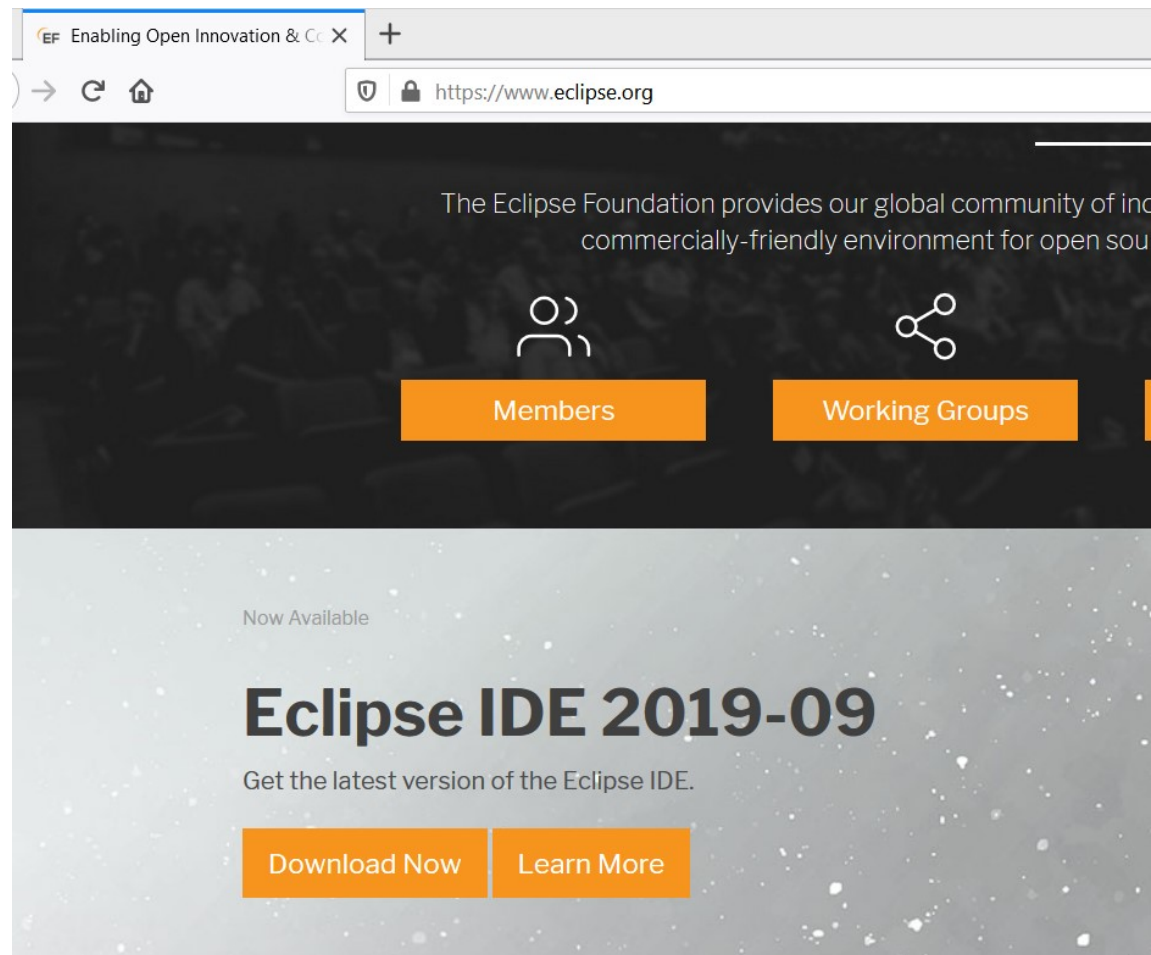
## **2. OBJET DE L'EXERCICE**

- Retrouver quelques réflexes en Java
- Apprendre à utiliser des classes qui vous aideront dans la suite des TDs
- Se refamiliariser avec Eclipse
- Faire un peu d'Excel

## **3. RAPPELS SUR ECLIPSE**

Le TD a été bâti à l'aide de l'IDE JDT de la plateforme Eclipse. Vous pouvez bien sûr utiliser d'autres IDE !  
Le présent paragraphe va rappeler les grands concepts de JDT et fournira quelques trucs et astuces.

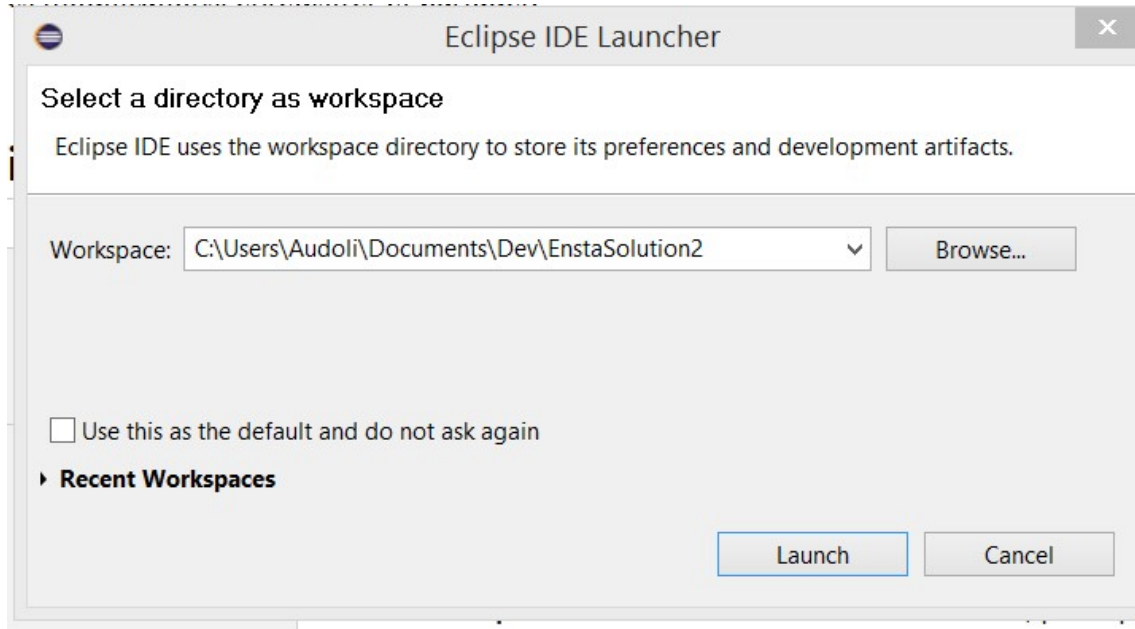
### 3.1 Téléchargement



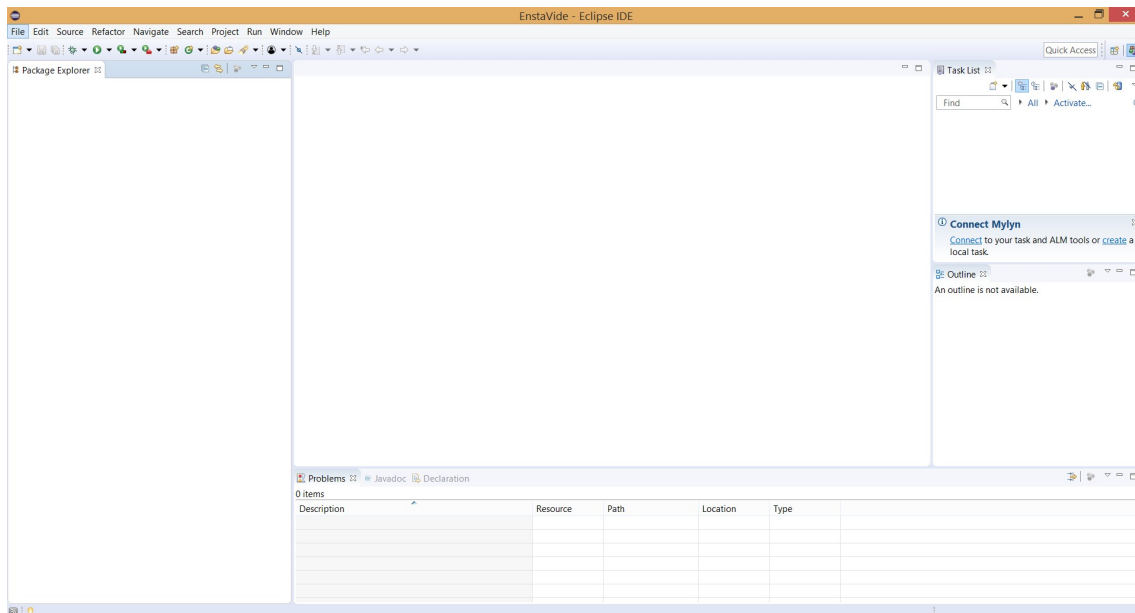
Vous obtenez en cliquant sur Download Now un fichier zip à unzipper. Pas besoin des droits d'administration pour l'installer.  
Lancer eclipse.exe.

### 3.2 Notions fondamentales

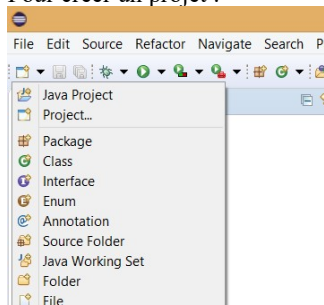
Tout d'abord il y a la notion de workspace. Il se choisit au démarrage d'Eclipse



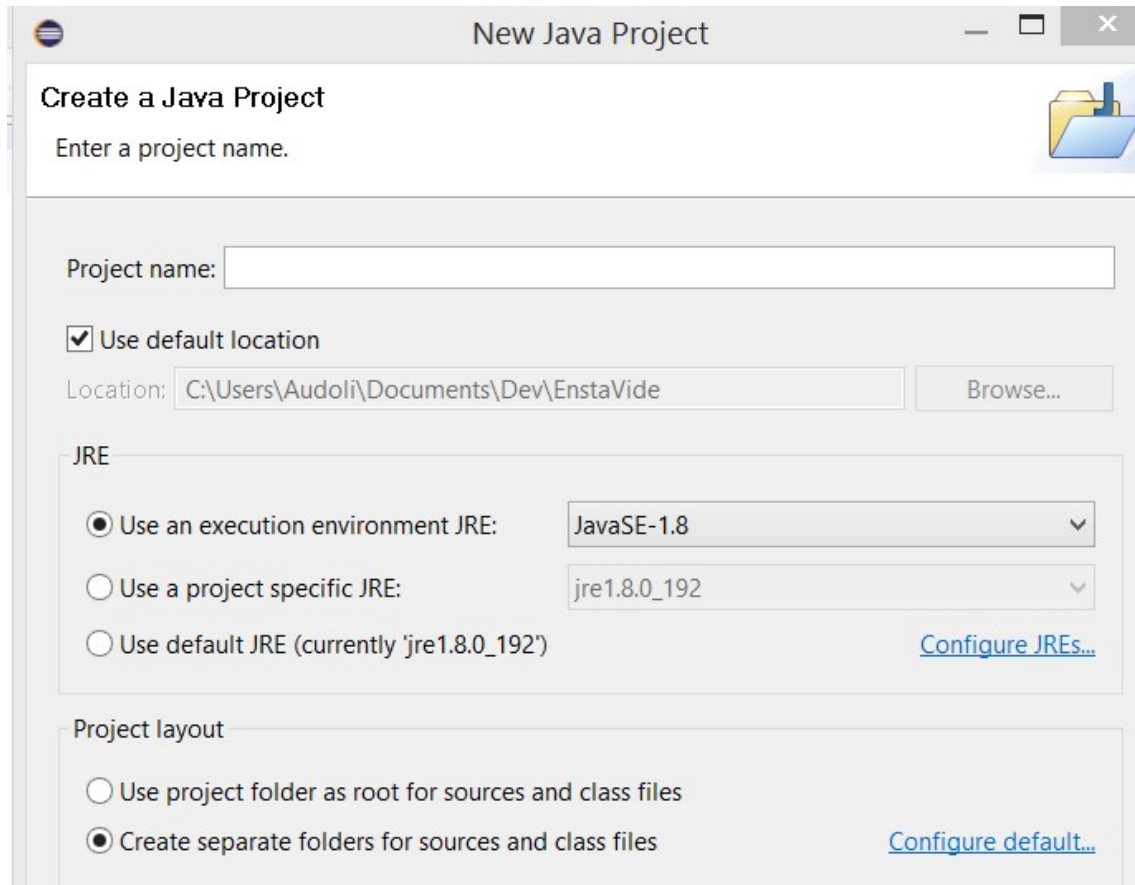
L'IDE se lance. Le package explorer est ouvert mais vide.



Pour créer un projet :



Choisir Java Project



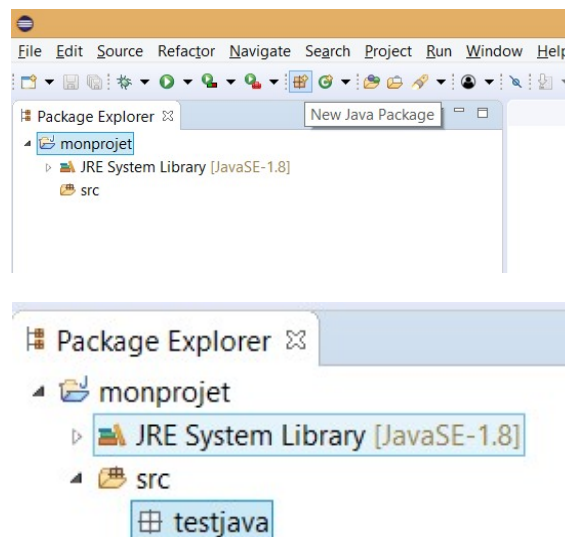
The screenshot shows the 'New Java Project' dialog box. It has a title bar with the Eclipse logo and the text 'New Java Project'. The main area is titled 'Create a Java Project' and contains the following fields and options:

- 'Enter a project name.' with a text input field.
- 'Project name:' with a text input field.
- A checked checkbox 'Use default location'.
- 'Location:' with a text input field containing 'C:\Users\Audoli\Documents\Dev\EnstaVide' and a 'Browse...' button.
- A section titled 'JRE' with three radio buttons:
  - 'Use an execution environment JRE:' (selected) with a dropdown menu showing 'JavaSE-1.8'.
  - 'Use a project specific JRE:' with a dropdown menu showing 'jre1.8.0\_192'.
  - 'Use default JRE (currently 'jre1.8.0\_192')' with a link 'Configure JREs...'.
- A section titled 'Project layout' with two radio buttons:
  - 'Use project folder as root for sources and class files'.
  - 'Create separate folders for sources and class files' (selected) with a link 'Configure default...'.

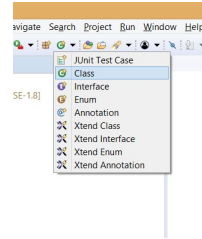
Saisir un nom comme `monprojet`.

Un projet peut être vu comme une bibliothèque.

On va créer un package en cliquant ici et choisir un nom comme par exemple `testjava`:



Puis on peut créer une classe Java, `Montest`, puis une interface `MonInterface`.



Ajouter une méthode à l'interface par exemple `String hello();` ;

```

1 package testjava;
2
3 public interface Moninterface {
4     String hello();
5 }
6

```

Ajouter deux attributs `String nom` et `Int age` à Montest.

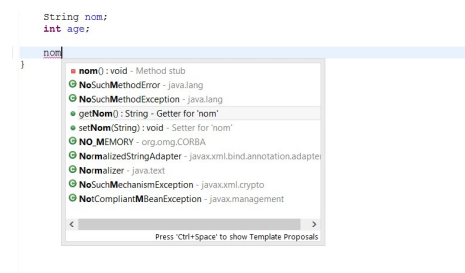
```

1 package testjava;
2
3 public class Montest {
4     String nom;
5     int age;
6
7 }
8

```

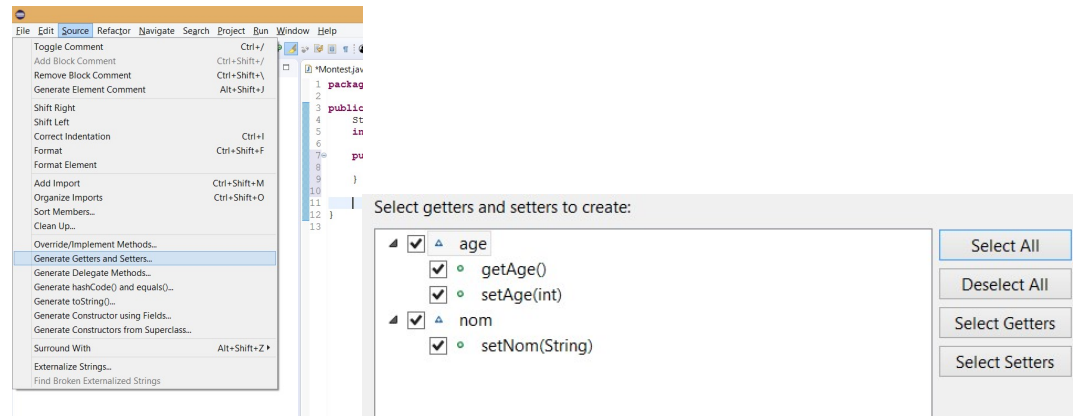
On va tout d'abord créer un `getter`.

Simplement taper dans le corps de la classe `nom` puis `CTRL+espace`.



Sélectionner le `getter` et taper `entrer`.

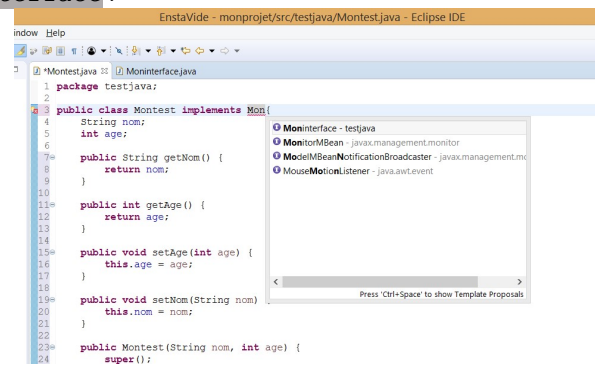
Maintenant on va générer tous les `getter` et `setter` manquant :



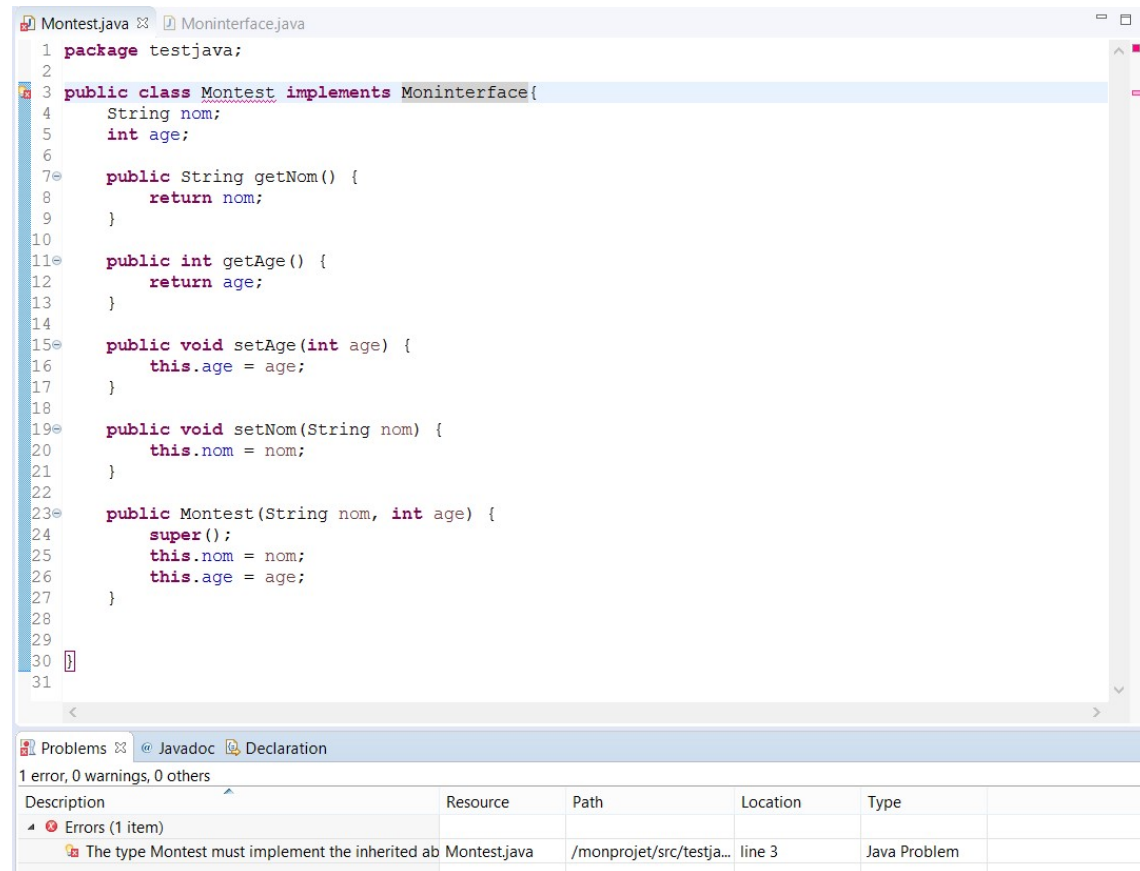
Puis cliquer sur **Generate**.

Générer de la même manière le constructeur, en prenant cette fois la fonction **Generate Constructor using Fields**.

Maintenant on va faire implémenter l'interface à la classe. Taper **Mon** puis **CTRL+espace**, un menu déroulant s'ouvre et choisir **Moninterface** :



La classe passe en erreur :



```

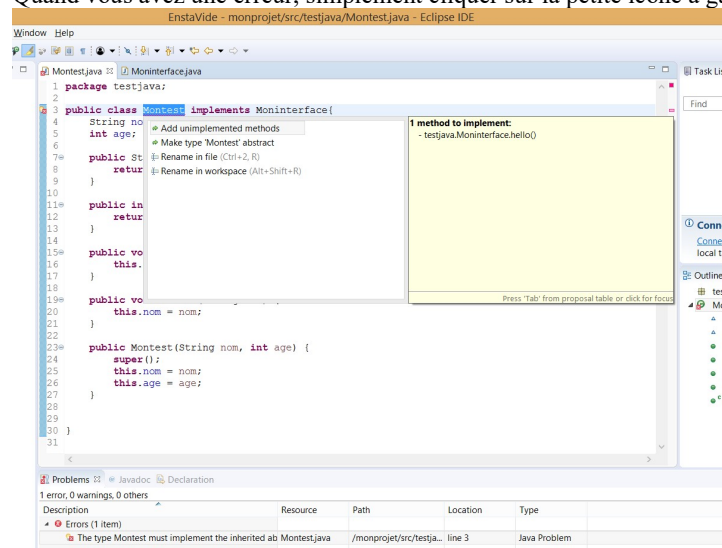
1 package testjava;
2
3 public class Montest implements Moninterface{
4     String nom;
5     int age;
6
7     public String getNom() {
8         return nom;
9     }
10
11    public int getAge() {
12        return age;
13    }
14
15    public void setAge(int age) {
16        this.age = age;
17    }
18
19    public void setNom(String nom) {
20        this.nom = nom;
21    }
22
23    public Montest(String nom, int age) {
24        super();
25        this.nom = nom;
26        this.age = age;
27    }
28
29
30
31

```

Problems 1 error, 0 warnings, 0 others

| Description                                      | Resource     | Path                     | Location | Type         |
|--|--------------|--------------------------|----------|--------------|
| Errors (1 item)                                  |              |                          |          |              |
| The type Montest must implement the inherited ab | Montest.java | /monprojet/src/testja... | line 3   | Java Problem |

Quand vous avez une erreur, simplement cliquer sur la petite icône à gauche de l'erreur en rouge :



EnstaVide - monprojet/src/testjava/Montest.java - Eclipse IDE

1 method to implement:  
- testjava.Moninterface.hello()

Press 'Tab' from proposal table or click for focus

Problems 1 error, 0 warnings, 0 others

| Description                                      | Resource     | Path                     | Location | Type         |
|--|--------------|--------------------------|----------|--------------|
| Errors (1 item)                                  |              |                          |          |              |
| The type Montest must implement the inherited ab | Montest.java | /monprojet/src/testja... | line 3   | Java Problem |

Il vous propose généralement une bonne aide ! Cliquer sur **add unimplemented methods**.

Remplacer le **return null** généré dans le corps de la méthode hello() par « Bonjour ».

```
@Override
public String hello() {
    return "Bonjour";
}
```

Retournez maintenant dans l'interface `Moninterface`.

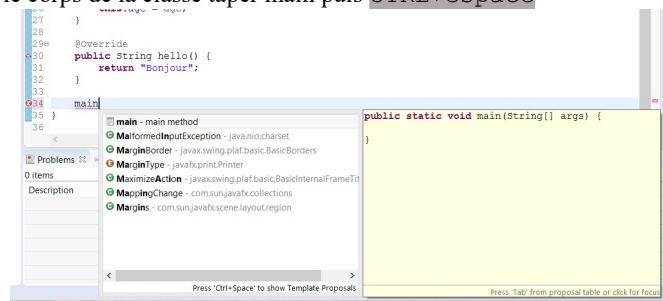
Placer la souris sur la méthode `hello` puis appuyer sur `CTRL`.

Trois propositions vous sont faites :

- `Open Declaration` (ce serait là où on est)
- `Open Implementation` (c'est là où on a implémenter la méthode !, si plusieurs possibilités, Eclipse vous aurait proposé les différentes implémentations possibles)
- `Open Return Type` (ouvre la définition du type retourné par la méthode)

Cliquer sur `Open Implementation`. On retourne dans `Montest`.

Dans `Montest`, dans le corps de la classe taper `main` puis `CTRL+espace`



Sélectionnez `main method` puis entrer.

Le corps du `main` est créé.

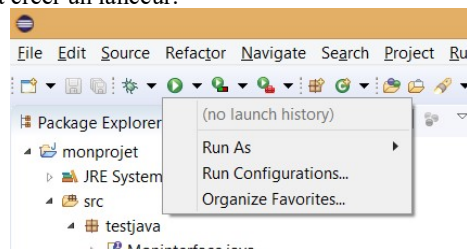
Maintenant dans le corps du `main` taper « `sysout` » puis `CTRL+espace`.

Compléter le code comme suit :

```
public static void main(String[] args) {
    Montest t= new Montest("Yoda",500);
    System.out.println(t.nom+" dit "+t.hello());
}
```

### 3.3 Lancer un programme

Pour lancer un programme il faut créer un lanceur.



En Java, il faut lancer le `main` d'une classe. Par défaut, le comportement d'Eclipse quand au lieu de cliquer sur le menu déroulant de l'icône `run` verte est de créer une configuration d'exécution sur la classe active si elle contient un `main`.

Donc cliquer sur le bouton vert en ayant ouvert et avec le focus le fichier `Montest.java`.

Lancer comme une application java si Eclipse vous demande (cela dépend des plugin fournis avec la distribution).



```
28
29 @Override
30 public String hello() {
31     return "Bonjour";
32 }
33
34 public static void main(String[] args) {
35     Montest t= new Montest("Yoda",500);
36     System.out.println(t.nom+" dit "+t.hello());
37 }
38 }
```

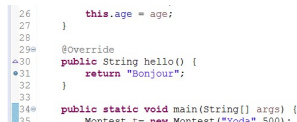
Problems @ Javadoc Declaration Console

<terminated> Montest [Java Application] C:\Program Files\Java\jre1.8.0\_192\bin\javaw.exe (2 déc. 2019 à 22:15:18)  
Yoda dit Bonjour

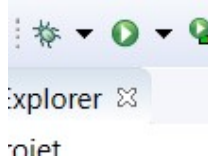
L'onglet **Console** s'active et vous voyez la sortie prévue.

### 3.4 Debugger

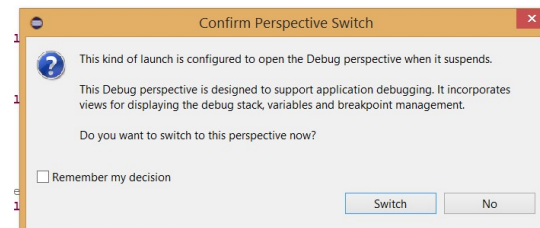
On va maintenant essayer de voir ce qui se passe au moment où hello est invoqué sur l'instance t de **Montest**. Pour cela double cliquer sur la partie gauche où se trouve le chiffre 31 de ligne. Une petite bulle va apparaître :



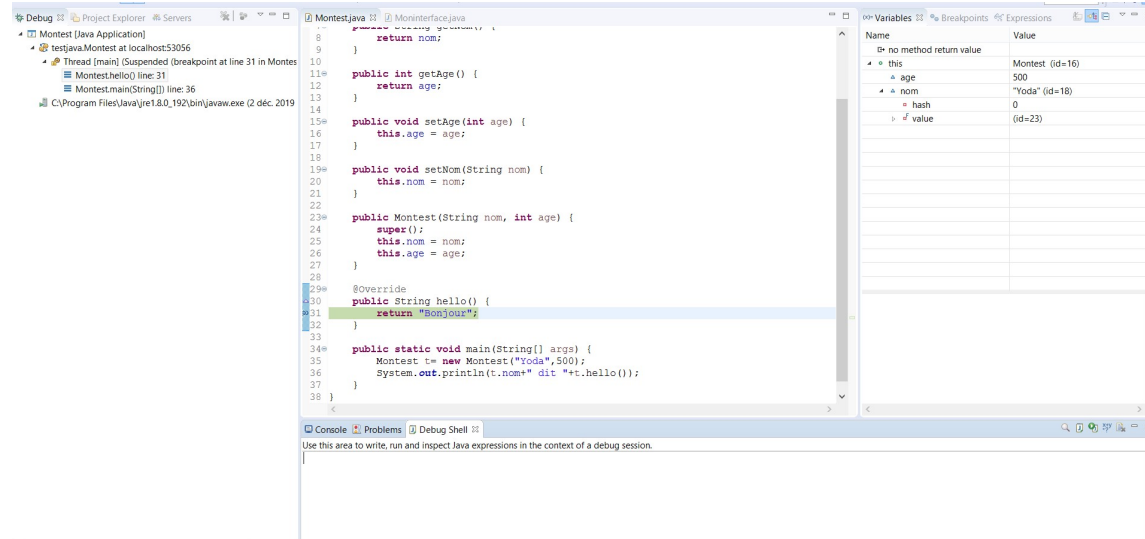
Maintenant lancer votre programme en mode débog en cliquant sur le cafard :



Eclipse va alors changer de mise en page et se placer dans ce qu'on appelle dans Eclipse une perspective c'est-à-dire une organisation bien pensée de fenêtres aux fonctions de la perspective donc ici à déboguer !



Vous pouvez cocher **remember**... puis **switch**



A gauche se trouve la pile d'appel. Cela vous permet de savoir qui appelle la fonction en question. On voit que c'est Main() qui appelle hello().

A droite vous avez différentes vues superposées.

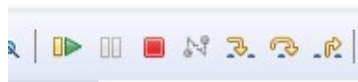
Variables vous montre le contexte mémoire où vous êtes.

On peut avoir besoin de faire des traitements ou des comparaisons d'objets en mémoire. Pour cela vous avez en bas le Debug shell.

Par exemple taper `age+15`, sélectionnez le texte et cliquez sur la loupe.



Avec les outils ci-dessous vous pouvez faire du pas à pas, remonter à la fonction appelante, rentrer dans une fonction. Attention si vous rentrez dans une fonction native de java, vous ne verrez pas son code si le code source du java n'a pas été téléchargé !

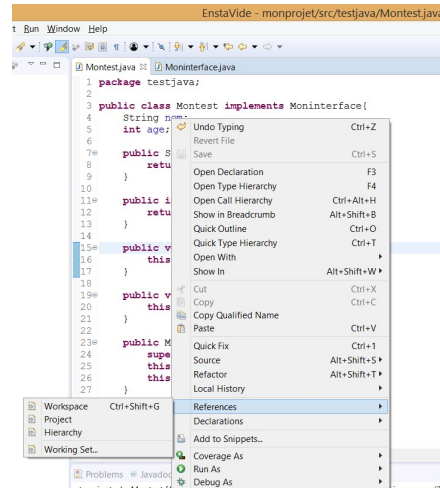


Pour revenir à la perspective d'édition cliquer en haut à droite sur l'icône :



### 3.5 Naviguer

Aller dans le code sur l'attribut « nom » avec la souris cliquer droit :




Cliquer sur Workspace (notez que **CTRL+SHIFT+G** fera la même chose).



Une panoplie d'outils de recherche s'ouvre en bas et vous montre où se trouve chaque référence à nom.

Enfin dernier outil pratique, la synchronisation de l'écran avec l'arborescence :



Cliquer sur .

Cliquez sur les onglets de code (par exemple sur l'onglet où moninterface est ouvert. vous verrez que le focus dans l'arbre se met sur le fichier Moninterface.java !

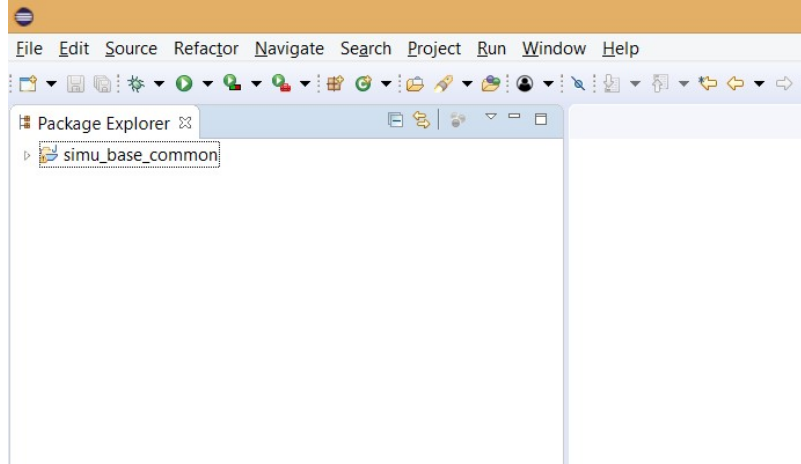
Très pratique quand on a beaucoup de fichiers....

#### 4. PREPARATION DE L'ESPACE DE TRAVAIL

- Récupérer sur Moodle les données : `simu_base_common.zip`
- Lancer Eclipse

- File > Import... > General > Existing Projects into Workspace
- Select archive file > Browse
- Vérifier que tout est coché et faire finish

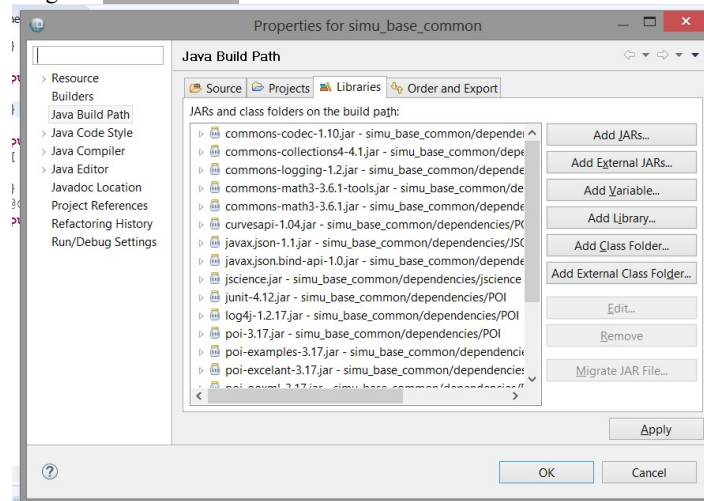
Cette opération va extraire les projets présents dans le zip et les placer dans l'espace de travail :



#### 4.1 Objectif et Structuration de Simu\_base\_common

Dans Eclipse, dans la vue Package Explorer,

- Cliquer droit sur `Simu_base_common`.
- Cliquer sur : `Build Path > Configure Build Path`
- Cliquer sur l'onglet : `Libraries`



Vous pouvez voir ici, les dépendances tierce de cette bibliothèque. Dans l'onglet Order and Export, vous pourrez voir que ces bibliothèques (jar) sont réexportées. Ceci signifie que si on crée un projet dépendant de `Simu_base_common` alors il pourra exploiter ces bibliothèques.

**La première fonction de Simu\_base\_common est donc de fournir aux projets complémentaires qui viendront dans les TD suivants et à vos futurs développements ces bibliothèques tierces.**

Les librairies tierces sur lesquelles certaines fonctions d'aide qui vous sont proposées sont bâties sont les suivantes :

- **POI de la fondation Apache** : il s'agit d'une bibliothèque vous permettant de lire et produire des fichiers au format MS Office

- **Jscience** : bibliothèque contenant de nombreuses fonctions scientifiques. Elle contient notamment un référentiel de grandeurs physiques, d'unités etc...
- **CommonMath** de la fondation Apache : il s'agit d'une bibliothèque contenant de nombreuses fonctions mathématiques, notamment des fonctions matricielles. Elles ne seront pas vraiment utilisées dans le cours. Mais peut vous servir un jour...
- **JSON-B** : il s'agit d'une bibliothèque qui permet un mapping POJO / fichier au format JSON. Très pratique pour créer des fichiers de configuration.

La seconde fonction est de vous fournir quelques fonctions indispensables à la réalisation de simulations :

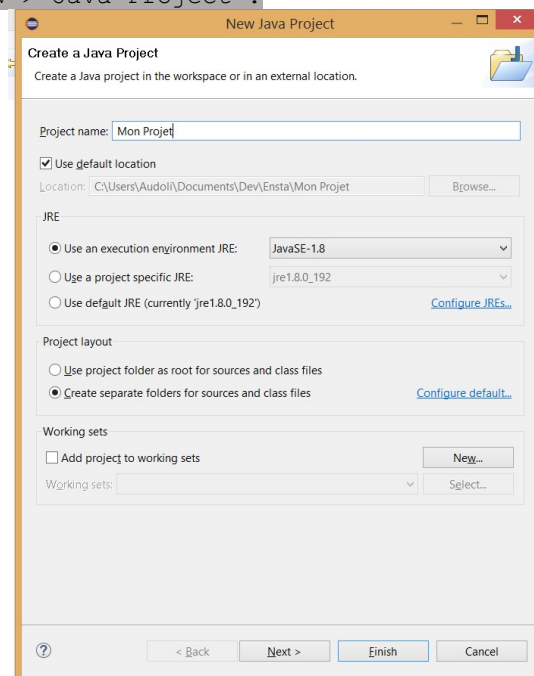
- Utilisation du temps logique supporté par deux classes : `LogicalDateTime` et `LogicalDuration`
- Utilisation du `logger` qui permet de journaliser des événements et aussi d'afficher ou d'enregistrer des données
- Utilisation des outils de génération de nombres aléatoires
- Utilisation d'une gestion de queue qui sera la colonne vertébrale du moteur de simulation.

Ces dernières fonctions vont être pratiquées par ce TD.

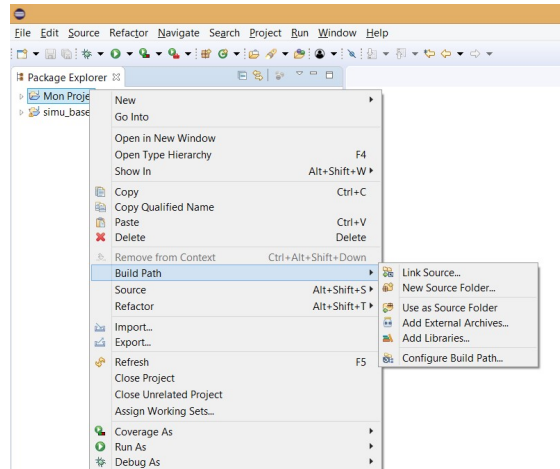
## 4.2 Mise en place d'un projet de travail pour le TD

L'objectif est de créer un espace de travail dépendant de `simu_base_common` et ainsi de bénéficier de ses classes sans avoir à polluer le projet fourni.

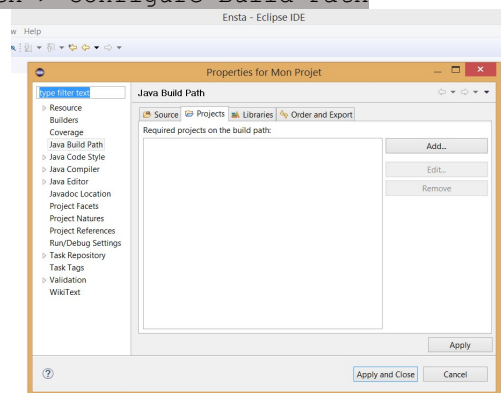
Dans Eclipse, `File > New > Java Project :`



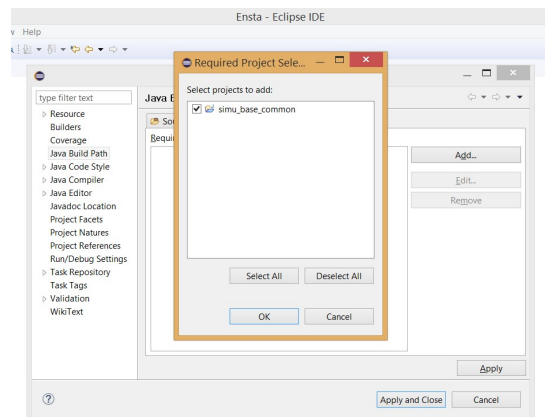
- Cliquer sur Finish.
- cliquer droit sur `Mon Projet`



- cliquer sur **Build Path > Configure Build Path**



- Cliquer sur **Add...**



- Cliquer sur le projet dont vous voulez dépendre, cliquer sur OK puis Apply and close
- Voilà, **Mon Projet** dépend de **simu\_base\_common**. Vous pouvez utiliser les classes et jar disponibles dans le projet.

## 5. EXERCICE 1 : METTRE EN ŒUVRE LE TEMPS LOGIQUE

### 5.1 Pré requis

Vous êtes invités à regarder le code de **LogicalDateTime** et **LogicalDuration**.

**Focus sur LocalDateTime :**

Pour la créer il on peut utiliser la fonction new.

La classe possède deux constructeurs.

Celui que vous utiliserez le plus est : `public LocalDateTime(String dateTimeFrenchFormat)`.

Le format de `dateTimeFrenchFormat` doit se conformer à la norme ISO : `JJ/MM/AAAA HH:MM:SS.SSSSSS`

Elle possède les opérations simples d'addition et soustraction.

```
ILogicalDateTime add(LogicalDuration offset);  
LogicalDuration subtract(LogicalDateTime d);
```

Ces méthodes permettent de manipuler facilement les dates lorsqu'on récupérer des Dates « rondes ».

```
ILogicalDateTime truncateToYears();  
ILocalDateTime truncateToDays();  
ILocalDateTime truncateToHours();  
ILocalDateTime truncateToMinutes();
```

Cette méthode permet de récupérer une valeur correspondant à un énuméré du JDK (Java Development Kit).

```
DayOfWeek getDayOfWeek();
```

Cette méthode permet de renvoyer une copie de la date.

```
ILocalDateTime getCopy();
```

La classe `LocalDateTime` est `Comparable`, ceci signifie qu'elle implémente la méthode `compareTo`. Cette méthode renvoie -1 si la date `o` est plus grand que l'objet sur lequel on invoque la méthode, 0 si c'est égale, +1 si `o` est plus petit.

```
int compareTo(ILocalDateTime o);
```

Cette méthode permet de renvoyer sous une forme de chaîne de caractères la date.

```
String toString();
```

Elle possède quelques constantes utiles, leur nom indique ce qu'elles sont:

```
LocalDateTime Zero;  
LocalDateTime MaxValue;  
LocalDateTime MinValue;  
LocalDateTime UNDEFINED;
```

**Focus sur LogicalDuration :**

`LogicalDuration` a une logique un peu différente. En effet, on manipule généralement des doubles pour calculer des deltas de temps par rapport au temps logique en secondes mais des entiers pour manipuler les minutes. Pour pouvoir interagir avec le temps logique, il a été décidé d'utiliser une durée elle aussi logique.

`LogicalDuration` n'a pas de constructeur public. Ceci signifie que vous ne pouvez pas faire `new LogicalDuration()` dans votre code.

Les durées seront fournies par des méthodes statiques. Ainsi pour obtenir les objets `Duration` vous saisissez un code du type `LogicalDuration.OfMinutes(5)` ce qui vous donnera une durée de 5 minutes. Vous n'avez pas à vous préoccuper de la représentation informatique de cette grandeur...

Les méthodes statiques disponibles qui vous seront utiles sont :

- `LogicalDuration Max(LogicalDuration, LogicalDuration)`
- `LogicalDuration ofDay(long)`
- `LogicalDuration ofHours(long)`
- `LogicalDuration ofMillis(long)`
- `LogicalDuration ofMinutes(long)`
- `LogicalDuration ofNanos(long)`

- `LogicalDuration ofSeconds(double)`

S'il vous manque des méthodes ne pas hésiter à en ajouter mais l'indiquer dans **votre rapport**.

Une fois un objet `LogicalDuration` obtenu, vous pouvez utiliser les méthodes suivantes, leur nom suffit à comprendre leur finalité :

```
int getMinutes();  
LogicalDuration add(LogicalDuration value);  
String toString();  
int compareTo(LogicalDuration o);
```

La fonction suivante permet d'avoir la représentation en double de la durée **en secondes**.

```
double DoubleValue();
```

## 5.2 Exercice

- Dans un `main()`, créer une date logique : le 10 décembre 2016, à 10h34min47.6789 secondes
  - Pour rappel, le formalisme de la date ISO est `JJ/MM/AAAA HH:MM:SS.SSSSSSS`
  - Pour cela mettre en œuvre la classe `enstabretagne.base.time.LogicalDateTime`
  - Regarder la structure du code en éditant le code de cette classe (taper par exemple F3) dans Eclipse
- Lui ajouter une durée logique de 15minutes et 12.67 secondes
- Ne pas hésiter à regarder le code source de ces classes. Vous verrez qu'elles encapsulent les classes de temps abstrait existantes dans le JDK 8
- Corrigé : `TestLogicalTimeSimulation`

## 6. EXERCICE 2 : METTRE EN ŒUVRE LE LOGGER SIMPLEMENT

### 6.1 Pré requis

Vous êtes invités à lire le document qui se trouve dans `.\doc\Logger.docx` dans le projet `Simu_base_common`.

### 6.2 Exercice

Vous être invité à utiliser les différentes méthodes de journalisation existantes et les différents loggers.

Deux d'entre elles (par classes anonymes et logger simple) sont proposées dans le corrigé : `TestLoggerSimple.java`

## 7. EXERCICE 3 : METTRE EN ŒUVRE LE LOGGER ET LA GENERATION DE NOMBRES ALEATOIRES

### 7.1 Pré requis

#### 7.1.1 Logger et CategoriesGenerator

Vous êtes invités à lire le document qui se trouve dans `.\doc\Logger.docx` dans le projet `Simu_base_common`.



### 7.1.2 MoreRandom

La classe `MoreRandom` est une classe étendant la classe `Random` de Java. En plus des nombreuses fonctions `nextXXXX()` de `Random`, `MoreRandom` y ajoute quelques fonctions :

- `getSeed() / setSeed(long)` : vous permet de spécifier le germe et de le récupérer.
- `nextUniform()` : vous permet de générer un double aléatoire suivant une distribution uniforme entre 0 et 1
- `nextUniform(double, double)` : vous permet de générer un double aléatoire suivant une distribution uniforme entre deux valeurs
- `nextExp(double)` : vous permet de générer un double aléatoire suivant une distribution exponentielle avec son paramètre Lambda
- `nextTriangle(double, double, double)` : vous permet de générer un double aléatoire suivant une distribution triangulaire

### 7.1.3 Tableaux et Graphiques Croisés sous Excel

Vous pouvez consulter ce lien pour apprendre à faire des tableaux croisés : [ici](#).

Vous pourrez synthétiser en quelques clicks les données. Seul inconvénient, les résultats présents dans ce tableau ne s'utilisent pas comme des cases Excel standard. Ainsi, si vous avez besoin de réaliser des traitements supplémentaires sur les données croisées, **il est recommandé de les copier-coller sans formule dans un autre tableau pour les exploiter.**

## 7.2 Exercice

Il vous est simplement demandé ici de montrer que `MoreRandom` génère bien une distribution Gaussienne avec la méthode `nextGaussian()`.

Vous montrerez qu'il s'agit d'une gaussienne :

- 1) De manière empirique graphiquement
- 2) De manière mathématique en utilisant la loi du Khi2.

Il vous faudra donc :

- Instancier `MoreRandom`
- Tirer un grand nombre de nombres dans `x` par exemple et les classer
- Enregistrer les données
- Utiliser le `CategoriesGenerator` pour créer des familles de données à des fins statistiques et retrouver les distributions.

Vous pourrez utiliser `Logger` pour :

- tracer ce que vous faites dans le programme (début, fin).
- Enregistrer la donnée `x` dans un fichier Excel
- Simplement instancier la classe `MoreRandom` avec une graine codée en dur
- Demander à `MoreRandom` de générer un grand nombre de fois un nombre suivant une des lois (ex : Gauss)
- Logger l'ensemble des nombres générés dans Excel/OpenOffice
- Vérifier que les nombres générés vérifient bien la loi demandée :
  - Soit en utilisant les fonctions statistiques d'Excel
  - Soit en ajoutant quelques petites fonctions en Java

Le corrigé est dans la classe : `LoggerAndProba.java`

## 8. EXERCICE 4 : RETROUVER LE NOMBRE PI EN UTILISANT LES NOMBRES ALEATOIRES

Cet exercice a juste pour but de se remémorer le type de capacités de l'exécution répétée d'un algorithme avec des nombres aléatoires.

Ceci illustre qu'on peut faire émerger des règles, lois « inattendues ».

- Calcul du nombre PI
  - Montrer que calculer la probabilité qu'une aiguille jetée sur un carré de 2 m de côté tombe dans un cercle de 1 m de rayon inscrit dans ce carré revient à calculer le rapport de deux surfaces
  - En déduire une méthode probabiliste pour calculer Pi en s'intéressant à un quart de cercle.

Le corrigé est dans la classe : `PICalculation.java`

## 9. EXERCICE 5 : METTRE EN ŒUVRE LA LISTE ORDONNEE

### 9.1 Pré requis

Le moteur de simulation événementiel repose sur un échéancier.

Il est fondamental d'avoir une classe dont la spécification est la suivante :

- A chaque fois qu'on lui ajoute un objet ayant une relation d'ordre, les objets sont ajoutés de manière à ce que pris successivement ces objets respectent la relation d'ordre.

Il vous est proposé une classe répondant à ce critère.

```
public class SortedList<T extends Comparable<T>> implements Iterable<T> {  
    List<T> l;  
    public SortedList() {  
        l = new ArrayList<T>();  
    }  
    [...]  
}
```

Elle encapsule une liste paramétrée. En Java il s'agit de Generics. La syntaxe indique que le type qui doit paramétrer cette classe est un type doté d'une relation d'ordre.

En Java, l'interface qui exprime qu'un type dispose d'une relation d'ordre est l'interface elle-même générique `Comparable<T>`.

Instancier la liste revient à écrire :

```
SortedList<Toto> liste = new SortedList<Toto>() ;
```

Si `Toto` n'implémente pas `Comparable<T>` alors le compilateur vous empêchera de compiler.

La classe `SortedList` est itérable.

Ceci signifie notamment que vous pourrez réaliser le type de parcours suivant :

```
For(Toto t : liste)  
    System.out.println(t.toString()) ;
```

### 9.2 Exercice

- 1) Il vous est demandé de créer votre propre type d'objet `Comparable`.

De l'instancier et de remplir une instance de `SortedList<T>` avec des objets dans le désordre selon sa relation d'ordre.

Vérifier que vous avez bien une liste bien ordonnée en parcourant dans l'ordre son contenu.

- 2) Créez une nouvelle `SortedList<T>` mais contenant des `LogicalDate`.

Créer des `LogicalDate` à des dates variées dans le désordre.

Pareil vérifier que le contenu est bien ordonné dans le temps croissant.

Un corrigé est proposé dans : `TestSortedList.java`