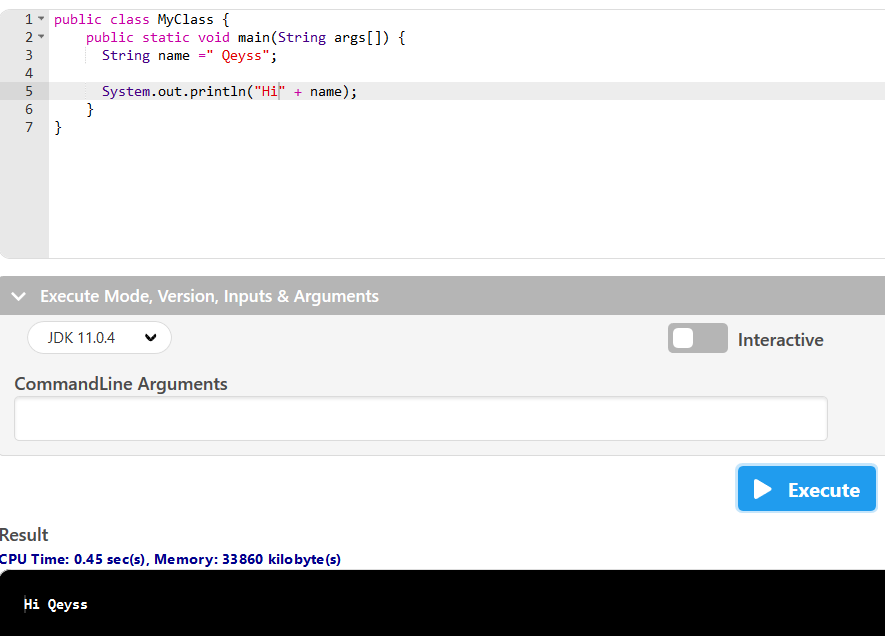
**Installez votre environnement de développement Java avec Eclipse**

# Écrivez du code Java en ligne ou dans un éditeur de texte

## Utilisez un éditeur en ligne

Avec un éditeur en ligne, rien à installer, tout se passe dans votre navigateur Internet : vous pouvez **coder**, **compiler** et **tester** votre code facilement et rapidement.

C’est une bonne solution pour débuter et s’exercer en Java. Plusieurs outils sont disponibles, comme **JDoodle**, compilejava.net, codiva.io, etc.



# Utilisez l’éditeur Visual Studio Code

[Visual Studio Code](https://code.visualstudio.com/) (ou VS Code) est un éditeur de code **léger** et **rapide** qui permet d’ajouter des **extensions**. Ce qui signifie que vous pouvez intégrer de nombreux langages et fonctionnalités. Ce n’est pas un IDE, même si la confusion est possible puisque de nombreuses **extensions**, ou *plugins*, sont disponibles pour l’améliorer.

Ces extensions donnent aux développeurs la possibilité d’installer :

* **un éditeur**, dans lequel on écrit le code ;
* **un débogueur**, pour aider à localiser les bugs et les erreurs dans votre code ;
* **un compilateur**, pour transformer votre code en quelque chose que votre ordinateur peut lire.

# Découvrez la puissance d’un IDE : Eclipse

Un IDE est un logiciel créé spécialement pour les développeurs, pour faciliter le développement des logiciels et des applications. C’est un outil tout-en-un pour le développement, puisqu’il inclut :

* un éditeur de texte avec **coloration syntaxique** et **autocomplétion** ;
* un **débogueur**, permettant de suivre pas à pas l’exécution de votre code ;
* un outillage de construction des sources en fichiers exécutables, le **build**, qui permet de :
  + **compiler** : transformer un fichier source en fichier binaire compréhensible par l'ordinateur,
  + **packager** : transformer un ensemble de fichiers binaires et d'autres fichiers de ressources en un seul fichier de bibliothèque ou d'application. Nous verrons la notion de bibliothèque dans la prochaine partie ;
* un outillage d’exécution, **le run**, pour tester le projet Java ;
* un système de **versionning** standard (par exemple Git), qui permet de sauvegarder le code en conservant l’historique des modifications (c’est primordial pour le travail en équipe, nous verrons cela en troisième partie) ;
* des outils complémentaires pour optimiser et fiabiliser le code, on parle souvent de ***refactoring*** *de code*.

Eclipse comprend aussi des systèmes de **construction de code** (*build*) comme **Maven** ou Gradle, pour créer et maintenir des projets plus complexes. Il peut aussi vous aider à mettre en place une base de projet, grâce aux assistants, ou ***wizards***, afin de ne pas partir d’un fichier vide à chaque fois

Eclipse simplifie le développement Java, en particulier pour les gros projets dans un contexte professionnel.

## D’autres IDE

**Intellij IDEA : c’**est un IDE qui a connu une progression fulgurante durant ces dernières années.

**NetBeans : c’**est un IDE gratuit et open source depuis 2000, qui supporte une grande variété de technologies associées au langage Java, mais aussi à d’autres langages. Il est relativement proche d’Eclipse

# Installez un JDK et Eclipse

## Qu’est-ce qu’un JDK ?

Le kit de développement Java, ou *JDK*, permet de **compiler** et d’**exécuter** le code Java sur votre ordinateur.

JDK contient :

* **un compilateur** permettant de traduire le code source Java en un code binaire optimisé pour n’importe quel système Windows, Linux, Mac ;
* **une JVM**, ou *machine virtuelle Java*. Il s’agit d’un logiciel sachant lire ce code binaire et l’exécuter sur le système de votre ordinateur.

## Téléchargez un JDK

Depuis janvier 2019, la politique tarifaire a changé, et pour une utilisation commerciale, il n’est plus possible d’utiliser gratuitement le JDK d'Oracle.

Heureusement, la communauté développe des versions libres du JDK. Les deux projets libres principaux sont :

* **OpenJDK** : c’est un peu le projet *source* de Java. Il contient toutes les dernières évolutions et corrections du compilateur et de la JVM. Mais vous ne pouvez pas l’utiliser directement sur votre ordinateur, en particulier s’il est sous Windows, car ce sont les codes sources du logiciel de Java et non les logiciels exécutables sur votre ordinateur ;
* **AdoptOpenJDK** : il s'agit d’un projet communautaire visant à fournir les logiciels exécutables de Java pour n’importe quel système Windows, Linux ou Mac, en incorporant au mieux les dernières mises à jour.

C’est donc sur AdoptOpenJDK que nous allons télécharger la JDK et choisir OpenJDK 8 et HotSpot.

Une fois le fichier téléchargé, cliquez dessus pour le lancer et l’installer, en gardant toutes les options par défaut.

## Téléchargez et installez Eclipse

<https://www.eclipse.org/downloads/packages/release/2020-03/r/eclipse-ide-java-developers>

Voici les étapes de base :

* cliquez sur ce lien pour accéder au [**site Eclipse**](https://www.eclipse.org/ide/);
* cliquez sur **Desktop IDEs**. Les liens qui apparaissent incluent les **plugins préinstallés** dont vous aurez besoin, selon le langage que vous allez utiliser ;
* sélectionnez **Eclipse IDE for Enterprise Java Developers**, qui comprend les plugins préinstallés non seulement pour un IDE Java, mais aussi pour le développement de projets web ;
* https://www.eclipse.org/downloads/packages/
* téléchargez la version **adaptée** à votre système et installez-la. Quand vous cliquez sur le **fichier exécutable** que vous avez téléchargé, conservez les paramètres par défaut. Eclipse vous fournit un espace de travail, c’est-à-dire un emplacement où vous pouvez placer vos fichiers. Vous pouvez choisir l’emplacement par défaut ou créer un nouveau dossier ;
* une fois l’installation terminée, lancez Eclipse depuis le programme d’installation en cliquant dessus.

# Créez votre premier projet Java sur Eclipse

**Créez un nouveau projet**

Vous pouvez commencer un nouveau projet en cliquant sur *File -> New -> Java Project*.

**Utilisez des paquetages**

Dans le panneau de gauche, vous pourrez voir le **Package Explorer**, qui contient tous les fichiers de votre projet. Il vous permet de classer et d’accéder à vos fichiers sources. Les **paquetages** permettent de classer votre code, un peu comme on classe des documents dans des répertoires.

Vous pouvez créer un paquetage en faisant un clic droit sur le nom du projet. Clic droit sur *HelloWorld -> New -> Package*.

À quoi servent les paquetages ?

Avec Java, le partage de code s’est développé très rapidement dès la création du langage. Un développeur, qui code une fonctionnalité utile pour d’autres codeurs, diffuse sa production sous forme de **bibliothèque**. Elle est alors utilisable par les autres développeurs.

D’innombrables **bibliothèques** ont été produites depuis la création de Java. Lorsque l’on utilise des bibliothèques, il faut alors absolument éviter les collisions, c’est-à-dire utiliser des fichiers de code source ayant le même nom, même s’ils proviennent de bibliothèques différentes. On les classe donc dans des **paquetages distincts** et **hiérarchisés**, un peu comme une arborescence de répertoires.

## Créez votre première classe Java sur Eclipse

En Java, on développe ce qu’on appelle des **classes**.

Généralement, on développe une classe par fichier, et le nom de la classe correspond au nom du fichier source sans l’extension *.java* à la fin.

**Qu'est-ce qu'une classe ?**

C'est comme une maquette permettant de créer des objets ayant une forme spécifique et des caractéristiques propres. Créer une instance d'une classe = créer un objet  
Un objet = une instance d'une classe.

**Qu'est-ce qu'un objet ?**

Un objet a des caractéristiques c'est à dire une identité, des attributs (propriétés ou variables d'instances) et des comportements (méthodes)

Par exemple l'objet Personne aura : Un nom  
Un âge  
Une adresse

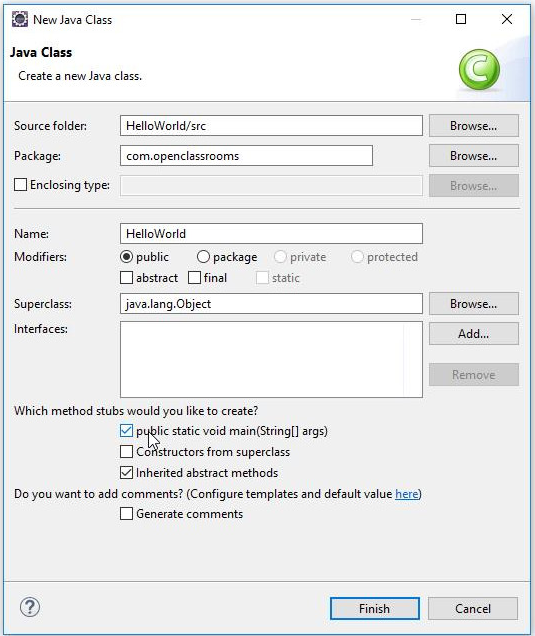
Il pourra courir, sauter, manger, etc..  
Par exemple un compte bancaire possède :

Un numéro Un détenteur Un solde

On peut faire des virements, retirer de l'argent, le clôturer, etc..

La classe doit contenir une méthode appelée main ce sera le point de démarrage de notre programme.

Cette classe principale avec main ne va pas créer d’objets mais simplement nous permettre de démarrer notre programme.



# Les variables et les opérateurs

Une variable est un élément qui stocke des informations de toute sorte en mémoire : des chiffres, des résultats de calcul, des tableaux, des renseignements fournis par l'utilisateur

## Les différents types de variables

Une déclaration de variable se fait comme ceci :

<type de variable> <Nom de la variable>

### Les variables de type numérique :

byte temperature ;

temperature = 64 ;

float pi ;

pi = 3.14159263f

int code;

code = 77200;

Ou bien int code = 77200 ;

Etc…

### Des variables stockant un caractère : char

Char caractere ;

caractere = "A";

### Le type booléen :

Le type boolean lui ne peut contenir que deux types de valeur : true ou false et ce sans guillemets.

boolean question ;

question = true ;

### Le type String :

Il permet de gérer les chaînes de caractères, c’est un type plus complexe que l’on appelle objet, c’est un objet de la classe String.

**//Première méthode de déclaration**

String phrase; (Déclaration)

phrase = "Titi et Grosminet "; (Initialisation)

**//Deuxième méthode de déclaration**

String phrase = new String();

phrase = "Titi et Grosminet";

//Troisième méthode de déclaration

String phrase = "Titi et Grosminet";

//Quatrième méthode de déclaration

String phrase = new String("Titi et Grosminet");

Convention pour nommer des classes et variables :

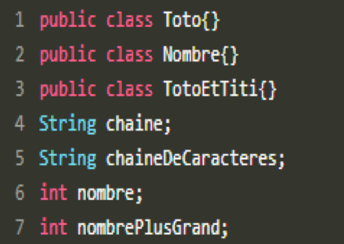
• tous vos noms de classes doivent commencer par une majuscule ;

• tous vos noms de variables doivent commencer par une minuscule ;

• si le nom d'une variable est composé de plusieurs mots, le premier commence par une minuscule, le ou les autres par une majuscule, et ce, sans séparation ;

• tout ceci sans accentuation

Exemples :



On peut aussi écrire ceci : int nbr1 = 2, nbr2 = 3, nbr3 = 5;

# Les commentaires:

Il est important de commenter son code.

Pour commenter sur une ligne on utilise //

Pour commenter sur plusieurs lignes on fait /\* … \*/

# Les opérateurs arithmétiques

« + » : permet d'additionner deux variables numériques (mais aussi de concaténer des chaînes de caractères).

• « - » : permet de soustraire deux variables numériques.

• « \* » : permet de multiplier deux variables numériques.

• « / » : permet de diviser deux variables numériques (mais je crois que vous aviez deviné).

• « % » : permet de renvoyer le reste de la division entière de deux variables de type numérique ; cet opérateur s'appelle le modulo.

int nbre1, nbre2, nbre3 ;

nbr1 = 1 + 1 ; //nbre1 vaut 2

nbre2 = 2 \* 7 ; //nbre2 vaut 14

nbre3 = nbre2/nbre1 ; //nbre3 vaut 7

nbre1 = 5 % 2 ; //nbre1 vaut 1

nbre2 = 27 % 4 ; //nbre2 vaut 3

nbre3 = 50 % 5 ; // nbre3 vaut 0

**L’incrémentation**

nbre1 = nbre1 + 1 ;

nbre1 += 1 ;

nbre1++

**Soustraction :**

nbre1 = nbre1 - 1 ;

nbre1 -= 1 ;

nbre1 --

On peut aussi multiplier, etc ..

nbre1 = nbre1 \* 2 ;

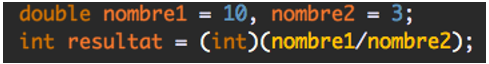
nbre1 \*=2 ;

On ne fait des calculs que sur des variables de même types, donc on additionne qu’un int avec un autre int et non un int avec un float.

## Les conversions, ou « cast »

Un byte contient peu d’informations, un int en contient davantage et un double encore plus.

Parfois on doit convertir une variable en une autre, par exemple un double en int :



Le résultat sera 3 et non 3.3333333 car le int ne retient que les entiers.

Int en float:

int nombre1 = 10, nombre2 = 3;

float resultat = (float)(nombre1/nombre2);

Va afficher 3.0

Priorités des calculs dans Java:

Java exécute d’abord le calcul, puis le cast et ensuite il affecte le résultat dans la variable.

int nombre1 = 3, nombre2 = 2;

double resultat = (double)(nombre1/nombre2);

Ici le résultat sera 1.0 et non 1.5 car Java a d’abord fait le calcul de deux int donc résultat 1, puis il a fait le cast donc toujours 1.0 et enfin il a affecté la valeur à la variable résultat donc 1.0

Pour obtenir 1.5 je dois faire ceci :

int nombre1 = 3, nombre2 = 2;

double resultat = (double)(nombre1)/(double)(nombre2);

Transformer la valeur d’un int en String:

int a = 7;

String b = new String();

b = b.valueOf(a);

Maintenant b est un String ayant pour valeur 12.

Transformation de la valeur d’un String en int:

int a = 7;

String b = new String();

b = b.valueOf(a);

int c = Integer.valueOf(b).intValue();

System.out.println(c + 5); // résultat = 12

# Lire les entrées clavier

## La classe Scanner

Pour que Java puisse lire ce que vous tapez au clavier, il faut utiliser un objet de type Scanner

Donc on tape par exemple:

Scanner keyboard = new Scanner(system.in);

Là on aura une erreur car il faut indiquer à Java où se trouve la classe Scanner. Pour faire ceci, nous devons importer la classe Scanner grâce à l'instruction import. La classe que nous voulons se trouve dans le package java.util

Donc on tape en haut : import java.util.Scanner;

Un **package** est un ensemble de classes.

Les classes qui se trouvent dans les packages autres que java.lang (package automatiquement importé par Java, on y trouve entre autres la classe System) sont à importer à la main dans vos classes Java pour pouvoir vous en servir.

Si je veux importer toutes les classes du package java.util j’écris : import java.util.\*;

## Pour récupérer un mot tapé :

On va utiliser la méthode nextLine()

import java.util.Scanner;

public class ScanString {

public static void main(String[] args) {

Scanner keyboard = new Scanner(System.in); // Création d'un objet Scanner

System.out.println("Please enter your name");

String name = keyboard.nextLine(); /\*On récupère ce qui a été tapé et on indique que c'est lui l'objet Scanner \*/

System.out.println(" So your name is " + name);

## Pour récupérer un nombre tapé :

On va utiliser la méthode nextInt()

import java.util.\*; // ainsi j'importe toutes les classes du package

public class ScanInt {

public static void main(String[] args) {

Scanner b = new Scanner(System.in);

System.out.println("Veuillez saisir votre âge");

int age = b.nextInt();

System.out.println(" Vous avez donc " + age + " ans");

En tapant le .nextInt tu appelles la méthode nextInt, si tu travailles sur un double ça serait nextDouble, etc..

Le type char n’est pas pris en compte par la classe Scanner, il faut donc récupérer un String puis utiliser une méthode de String permettant de récupérer le premier caractère à savoir charAt(0) :

import java.util.\*;

public class ScanChart {

public static void main(String[] args) {

System.out.println("Veuillez taper une lettre");

Scanner clavier = new Scanner(System.in);

String lettre = clavier.nextLine();

char carac = lettre.charAt(0);

System.out.println("Vous avez tapé la lettre " + carac);

Autre variante :

System.out.println("Veuillez taper une lettre");

Scanner clavier = new Scanner(System.in);

char lettre = clavier.nextLine().charAt(0);

System.out.println("Vous avez tapé la lettre " + lettre);

1

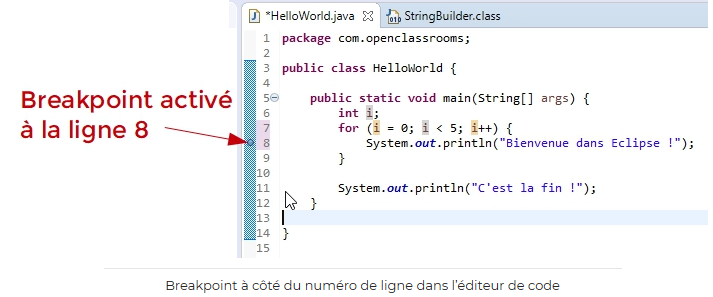
# Utilisez les fonctionnalités principales d’Eclipse

## Déboguez dans Eclipse

**Choix des points d'arrêts**

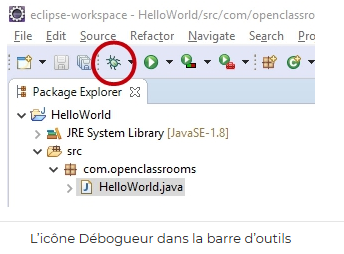
Pour arrêter et démarrer un programme, nous avons besoin de **points d'arrêts**, ou ***breakpoints***. C’est ce qui vous permet d’arrêter le programme là où vous l’avez décidé. Mettons un breakpoint au niveau de la ligne où le code imprime notre message. Pour ce faire, rendez-vous au numéro de ligne 8, clic droit et cliquez sur *Toggle breakpoint*. Un petit point bleu apparaîtra à côté du numéro de ligne. Vous pouvez aussi l’enlever en cliquant sur *Disable breakpoint*.

Vous pouvez aussi cliquer deux fois dans la marge à gauche de l'éditeur de code, à la hauteur de la ligne concernée. Le point bleu apparaît, puis disparaît si vous cliquez deux fois sur le point bleu.



**Exécution en mode débogage**

Maintenant, trouvez le bouton « Debug as... » dans votre barre d’outils, et cliquez dessus. Il ressemble à ça :



Vous pouvez avancer dans le programme en cliquant sur l'icône ***Step Over*** dans la barre d’outils de l’onglet de débogage, ou sur la barre d’outils principale, comme indiqué ci-dessous.



Vous pouvez tester les autres actions pas à pas comme ***Step Into*** et ***Step Return***. Suivez le screencast ci-dessus pour en savoir plus. Cliquez ensuite sur ***Step Over*** pour continuer d’itérer le programme. Pour pouvez aussi cliquer sur ***Resume***(flèche verte de la barre débogage), pour vous emmener directement au deuxième breakpoint, à la ligne 11. Si vous continuez de cliquer sur ***Step Over***, vous verrez le message final s'afficher, puis c'est la fin du programme.

Vous avez fini de déboguer ! Pour sortir de la perspective, cliquez sur *Window* et *Reset perspective*.

## Testez vos méthodes avec JUnit

**JUnit** est la bibliothèque Java la plus répandue pour coder des tests.

Qu'est-ce que cela signifie, *coder des tests* ?

Quand vous codez de plus en en plus de fonctionnalités, tester manuellement l'exécution de l'application et toutes ses fonctionnalités est une activité répétitive et peu créative. De plus, coder une fonctionnalité peut avoir un impact à un autre endroit de l'application. Donc, si l'on est rigoureux, il faudrait tester toute l'application à chaque développement ! Afin de pallier ces tâches fastidieuses, le développeur code des tests qui seront exécutés à la demande et feront le travail **automatiquement** ! On code ces tests une fois, mais on va pouvoir les exécuter mille fois !

Les tests les plus répandus sont les **tests unitaires** ou ***TU***. Ils testent une "chose" à la fois : une fonction, une méthode, un comportement, de manière isolée du reste. JUnit permet donc de coder des tests en Java, et Eclipse apporte une interface graphique pour faciliter la conception et l'exécution de tests JUnit.

**Création du test JUnit**

Nous allons créer une méthode dans notre classe HelloWorld pour ajouter deux nombres. Insérez le code ci-dessous dans la classe Salam.

}

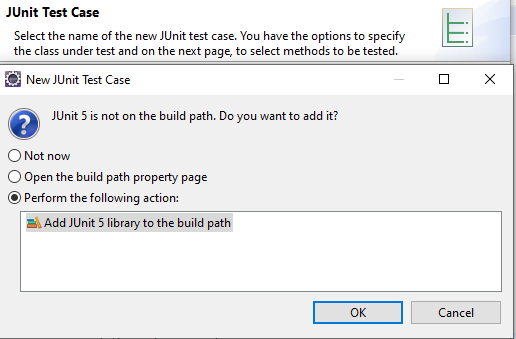
**public** **int** ajouter(**int** num1, **int** num2) {

**return** num1 + num2;

}

Maintenant, nous allons coder une méthode pour tester si la fonction "ajouter" fonctionne correctement. Allez dans *File → New →  Test JUnit*

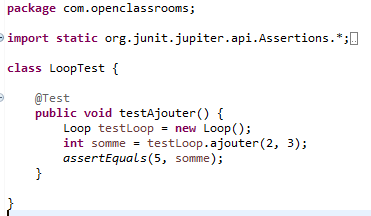
Vous pouvez nommer le test en ajoutant « **Test** » après le nom de la méthode. C'est une **convention de nommage** usuelle en Java. Dans le cas présent, je l’ai appelé HelloWorldTest avant de cliquer sur Finish. Eclipse vous propose alors d'ajouter la bibliothèque JUnit à votre projet. Validez cette proposition.



Vers le haut du fichier, une ligne commençant par le mot clé *import* indique que le code de test va utiliser  org.junit.jupiter.api.Assertions. Cette bibliothèque Java vous permettra d’utiliser une fonction appelée assertEquals() qui vous aidera à tester vos résultats facilement. Voici les étapes pour modifier la fonction de test vide par défaut :

1. Renommez votre fonction de test selon la nomenclature Java. Ici, appelez-la  public void testAjouter().
2. Créez un nouvel objet à utiliser par la classe HelloWorld dans le test. Le nouvel objet se nomme *testHelloWorld*.  HelloWorld testHelloWorld = new HelloWorld();
3. Appelez la fonction à tester (c'est ajouter()), et stockez le résultat dans une variable *somme* de type entier :int somme = testHelloWorld.ajouter(2, 3)**;**
4. Testez si le résultat de la méthode Ajouter correspond au résultat attendu, c'est-à-dire 5. Pour cela, on utilise une fonction de JUnit qui se nomme *assertEquals.* Cette fonction prend en premier paramètre la valeur attendue, et en deuxième paramètre la valeur à tester. La ligne de code est donc assertEquals(5, somme);

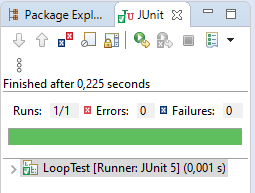
En résumée la méthode de test modifiée est :



**Exécution du test JUnit**

Maintenant, testons la méthode avec JUnit ! Clic droit sur le fichier de test. En l’occurrence, le fichier HelloWordTest.java. Allez dans *Run As -> JUnit Test*.

Vos résultats apparaîtront dans l’onglet JUnit à la place du package explorer.



L’exemple ci-dessus montre un **test réussi**. Un échec affichera une barre rouge.

## Le refactoring de code

C’est le fait de retravailler le code pour qu'il soit plus clair.

Une des grandes forces d'Eclipse est de vous **faciliter le développemen**t. En particulier lorsque vous avez besoin de remettre votre code au propre. Vous travaillerez en équipe, et lorsque du code passe d'un développeur à un autre, il doit être le plus lisible et le plus clair possible.

Voici un aperçu rapide et non exhaustif de ce que propose Eclipse :

**Formatage du code**

C’est le fait d'écrire un code bien espacé, avec les retours de ligne corrects.

Il existe plusieurs **conventions** pour **formater** correctement du code. Eclipse en propose une par défaut. Et pour l'appliquer, c'est simple :

1. Ouvrez le code source de HelloWorld.java.
2. Choisissez une ligne de code et enlevez les **espaces** avant ce code. Vous pouvez aussi mettre deux lignes de code sur une seule ligne. Java autorise de faire cela, grâce au symbole ";" qui sépare bien les différentes instructions de code. Mais c'est moins lisible !
3. Allez dans le menu *Source -> Format*, ou bien saisissez le raccourci clavier CtrlMajF
4. Votre code est formaté !

**Renommage de méthode**

Il s'agit aussi de nommer les variables de manière explicite, de regrouper les fonctions de manière logique, etc.