Génie logiciel

Implémentation

Louis-Edouard LAFONTANT

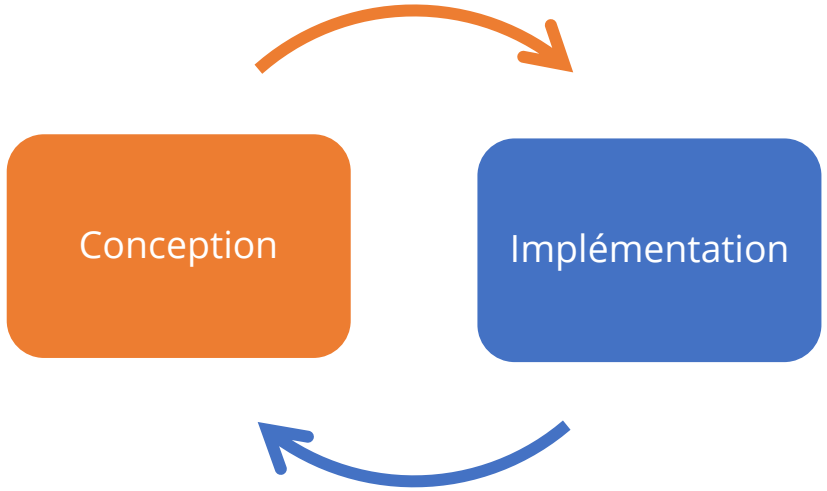
geodes

Université de Montréal

De la conception au code

Après avoir élaboré une stratégie (architecture) et un plan (diagrammes), nous pouvons les mettre à l'épreuve dans l'implémentation (code exécutable).

- **L'implémentation et la conception doivent rester cohérentes**



Une bonne conception…

… contribue à la **moitié de l'effort** d'implémentation !

✓**Rigueur**
- S'assure que toutes les exigences sont satisfaites

✓**Séparation des préoccupations**
- **Modularité**
  - Permet le travail isolé et parallèle: composants sont indépendants des autres
- **Abstraction**
  - Permet le travail isolé et l'intégration: interfaces garantissent que les composants vont fonctionner ensemble

✓**Anticipation du changement**
- Permet d'absorber les changements sans effort

✓**Généralisation**
- Permet de réutiliser les composants à travers le système et d'autres systèmes

Une mauvaise conception…

… ne sera **jamais implémentée** !

- × Manque de rigueur ⇒ oubli de fonctionnalités
- × Manque de modularité ⇒ conflits entre développeurs ou travail en double
- × Manque d'anticipation au changement ⇒ re-conception et ré-implémentation
- × Manque de généralité ⇒ gonflement du code: allongement, ralentissement, duplication, gaspillage de ressources

Flux d’implémentation

But: Implémenter le produit logiciel cible

- Un grand système est partitionné en **sous-systèmes,** implémentés en **parallèle** par l’équipe de développement**.**
- Sous-systèmes consistent en des **composants** ou des **artéfacts de code**
- L’implémentation d’un artéfact est toujours suivi de **tests**
  - Module est envoyé à l’équipe d’**assurance qualité** pour le tester plus en profondeur
  - Tester fait partie de l’implémentation: **l’implémentation n’est pas complète tant qu’elle n’est pas testée**

Taches d’implémentation

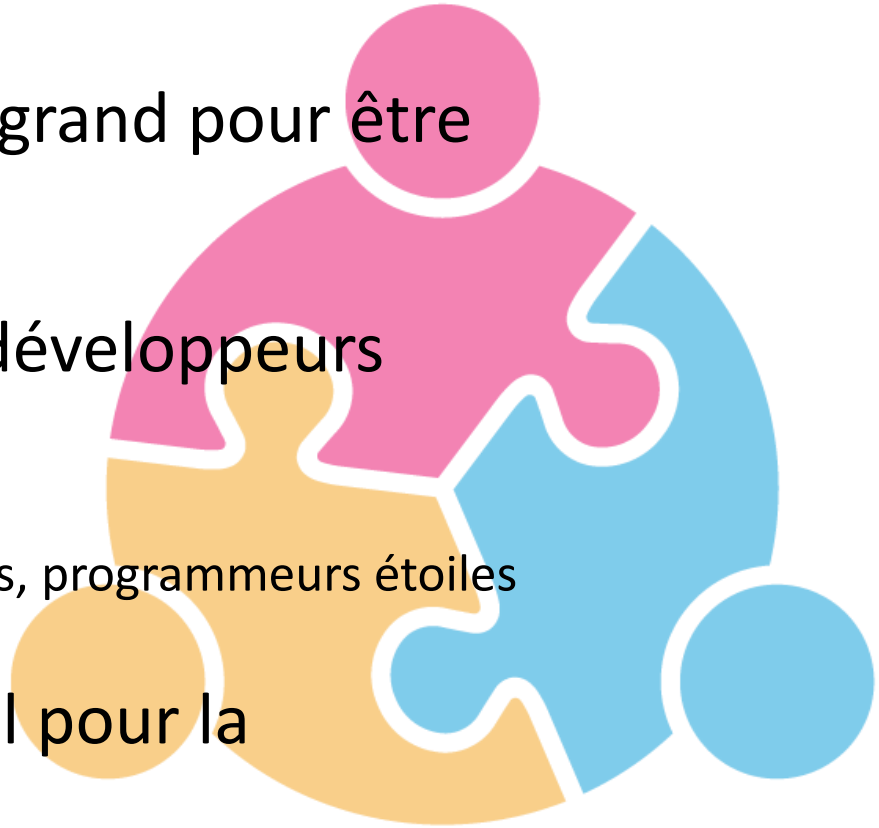
- Choisir le langage de programmation le plus approprié
- Établir les normes de programmation
- Répartir l’effort de travail
- Implémenter = coder + tester
- Intégrer

Répartition de l'effort

- Les produits logiciel sont en général trop grand pour être implémentés par un seul programmeur

- Assigner différents modules à différents développeurs
  - Assignation peut être **incrémentale**
  - Gérer les **changements** d'assignation
    - Maladie, démission, recrue, ajustement de temps, programmeurs étoiles

- Le développement **d'interfaces** est crucial pour la programmation distribuée
  - Ils forment des **contrats** entre les modules



Choix du langage de programmation

Critique!

Choix des langages

➢Pas toujours libre de choisir ce que l'on veut

Contraintes

- Le langage est généralement spécifié dans le contrat
  - Mais le contrat peut stipuler que le logiciel doit être implémenté dans le langage *le plus approprié*
- L'environnement de déploiement peut restreindre les choix
  - Exemple: Web, iOS
- L'équipe de développement a des préférences
  - Expérience avec un langage en particulier
  - Librairies disponibles à l'interne

Générations des langages de programmation

- **Langages de 1ère Génération**
  - Langage machine
- **Langages de 2e Génération**
  - Assembleur spécifique à une machine
- **Langages de 3e Génération**
  - Langage de haut niveau compilé vers code machine
- **Langages de 4e Génération**
  - Base de données, Visual Basic, Formulaires
- **Langages spécifiques au domaine**
  - Décrits en utilisant les concept du domaine, pas le code
  - Code est automatiquement généré à partir des modèles

```
11011000111010001001000100101001010010010010
```

```
LOAD [26], R1
LOAD 3, R2
ADD R1, R2, R3
STORE R3, [1700000029]
```

```
ATM atm = new ATM(10);
atm.deposit(100);
atm.withdraw(20);
System.out.print(atm.getBalance());
```

```
for every surveyor
  if rating is excellent
    add 6500 to salary
```

```
document <no diaryNumber>
  handler: <no handler>
  description: <no description>

  requirements

    << ... >>
```

Chronologie des langages de programmation

Premiers langages de programmation modernes apparurent dans les 1950

- **1951 - Regional Assembly Language**
- 1952 - Autocode
- **1954 - FORTRAN**
- 1954 - IPL
- 1955 - FLOW-MATIC
- 1957 - COMTRAN
- 1958 - LISP
- 1958 - ALGOL
- 1959 - FACT
- **1959 - COBOL**
- **1962 - Simula**
- 1962 - SNOBOL
- 1963 - CPL
- **1964 - BASIC**
- 1964 - PL/I
- 1967 – BCPL

Paradigmes fondamentaux

- 1968 - Logo
- **1970 - Pascal**
- 1970 - Forth
- **1972 - C**
- **1972 - Smalltalk**
- **1972 - Prolog**
- **1973 - ML**
- 1975 - Scheme
- **1978 – SQL**

Grande échelle et performance

- **1980 - C++**
- 1983 - Objective-C
- **1983 – Ada**
- 1984 - Common Lisp
- **1985 - Eiffel**
- 1986 - Erlang
- **1987 - Perl**
- 1988 - Tcl
- 1989 – FL

L'ère de l'internet

- 1990 - Haskell
- **1991 - Python**
- 1991 - Visual Basic
- **1993 – Ruby**
- 1993 - R
- 1993 - Lua
- **1995 - Java**
- 1995 - Delphi
- **1995 - JavaScript**
- **1995 - PHP**
- 1997 - Rebol
- 1999 – D

Plus récemment

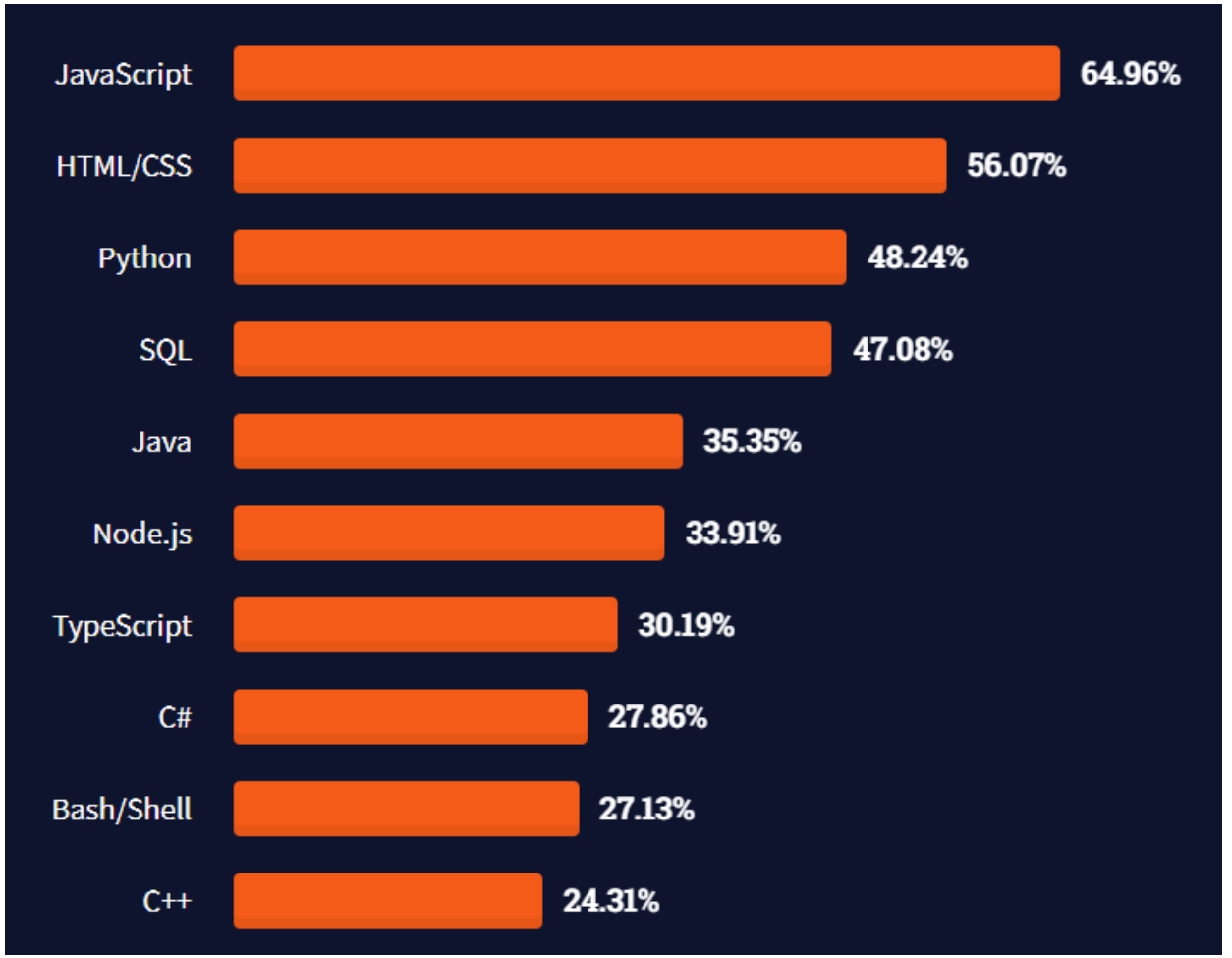
- **2001 - C#**
- 2001 - Visual Basic .NET
- 2002 - F#
- 2003 - Scala
- 2003 - Factor
- 2007 - Clojure
- 2007 - Groovy
- **2009 – Go**
- 2011 – Dart
- 2011 – Kotlin
- **2014 – Swift**

Ce n'est qu'une liste partielle…
https://en.wikipedia.org/wiki/History_of_programming_languages

Popularité des langages de programmation

Rank	Language	Type	Score
1	Python		100.0
2	Java		95.4
3	C		94.7
4	C++		92.4
5	JavaScript		88.1
6	C#		82.4
7	R		81.7
8	Go		77.7
9	HTML		75.4
10	Swift		70.4

IEEE SPECTRUM



[Top Programming Languages 2021 - IEEE Spectrum]

[Stack Overflow Developer Survey 2021]

FORTRAN

- « FORmula TRANslator », inventé par John Backus et IBM
- Impact révolutionnaire en informatique
  - Premier langage de haut niveau qui est exécutable
- Utilisé principalement pour le **calcul scientifique**
- D'excellents **compilateurs** existent encore aujourd'hui
  - Dernière version FORTRAN 2015

```
      PROGRAM DEGRAD
! Déclaration des variables
      INTEGER DEG
      REAL RAD, COEFF
!
! En-tête de programme
      WRITE (*, 10)
   10 FORMAT (' ',20('*') /
     &        ' * Degres * Radians *' /
     &        ' ', 20('*') )
!
! Corps de programme
      COEFF = (2.0 * 3.1416) / 360.0
      DO DEG = 0, 90
         RAD = DEG * COEFF
         WRITE (*, 20) DEG, RAD
   20 FORMAT (' *  ',I4,'  * ',F7.5,' *')
      END DO
!
! Fin du tableau
      WRITE ( *, 30)
   30 FORMAT (' ',20('*') )
!
! Fin de programme
      STOP
      END PROGRAM DEGRAD
```

COBOL

- **COmmon Business-Oriented Language** : Un langage commun pour la gestion de l'administration
- Très puissant pour les **calculs numéraires**: le must en matière de gestion et de manipulation précise
- *Le plus répandu des langages (développé par et pour le DoD)*

```
 IDENTIFICATION DIVISION.
   PROGRAM-ID. HELLO-WORLD.
*
 ENVIRONMENT DIVISION.
*
 DATA DIVISION.
*
 PROCEDURE DIVISION.
 PARA-1.
   DISPLAY "Hello, world.".
*
      EXIT PROGRAM.
   END PROGRAM HELLO-WORLD.
```

Lisp

- Langage **fonctionnel** développé par McCarthy comme implémentation du calcul lambda de Church
  - Fonctions sont des éléments de premier ordre (comme `Object`)
- Très puissant pour les calculs symboliques, beaucoup d'applications en **intelligence artificielle**
- A influencé plusieurs langages fonctionnels (ML et Haskell)

```
(defun factorial (n)
  (if (<= n 1)
      1
    (* n (factorial (- n 1)))))
```

C

- Langage de **programmation système**
  - Systèmes d'exploitation, superordinateurs, systèmes embarqués
- Le compilateur de plusieurs langages sont implémentés en C (ex: Python, PHP)
- Un des langages les plus efficaces (optimisés) et des plus utilisés encore aujourd'hui

```c
#include <stdio.h>
#include <time.h>

void getSeconds(unsigned long *par);

int main () {
   unsigned long sec;
   getSeconds( &sec );
   /* print the actual value */
   printf("Number of seconds: %ld\n", sec );
   return 0;
}

void getSeconds(unsigned long *par) {
   /* get the current number of seconds */
   *par = time( NULL );
   return;
}
```

C++

- Langage moderne orienté-objet qui permet aussi la manipulation de mémoire à bas niveau: basé sur un sous-ensemble de C
- Polymorphisme, héritage multiple, Standard Template Library (collections et itérateurs génériques), pointeurs, opérations pour le garbage collection

```cpp
#include<iostream>

int main()
{
    using std::cout;
    cout << "Hello, new world!" // s
         << std::endl;          // 
}

void foo()
{
    std::cout << "Hello, new world!"
              << std::endl;
}
```

```cpp
// messageinternet.hpp
#include<string>

class MessageInternet
{
    private:
        const std::string m_sujet, m_expediteur, m_d
attributs

    public:
        MessageInternet(
            const std::string& sujet,
            const std::string& expediteur,
            const std::string& destinataire); // con
        ~MessageInternet(); // destructeur
        const std::string& get_sujet() const; // mét
        const std::string& get_expediteur() const; /
        const std::string& get_destinataire() const;
};
```

C#

- Similaire à Java et C++ mais pour la plateforme Microsoft .NET et .NET Core
- Le *common language infrastructure* gère les objets qui peuvent être partagés entre **différents langages et paradigmes** de programmation (Visual Basic, F#)
- Aussi utilisé dans les projets
  - **Xamarin**: développement d'application mobile multiplateforme
  - **Unity**: moteur de jeu multiplateforme

```
using System;
using System.Linq;

class Program {
  static void Main()
  {
    string[] words = {"hello", "wonderful", "LINQ", "beautiful", "world"};
        //Get only short words
        var shortWords = from word in words
                         where word.Length <= 5
                         select word;

        //Print each word out
        foreach (var word in shortWords) {
      Console.WriteLine(word);
        }
        Console.ReadLine();
  }
}
```

Java

- Développé par Sun Microsystems, maintenant propriété d'Oracle
- Simplification de C++ pour ne supporter que la programmation orientée-objet
- Portable et indépendant de la machine grâce à une machine virtuelle (JVM)

```java
package fibsandlies;
import java.util.HashMap;

/**
 * This is an example of a Javadoc comment; Javadoc can compile documentation
 * from this text. Javadoc comments must immediately precede the class, method, or f
 */
public class FibCalculator extends Fibonacci implements Calculator {
    private static Map<Integer, Integer> memoized = new HashMap<Integer, Integer>();
    /*
     * The main method written as follows is used by the JVM as a starting point for
     */
    public static void main(String[] args) {
        memoized.put(1, 1);
        memoized.put(2, 1);
        System.out.println(fibonacci(12));
    }

    /**
     * Given a non-negative number FIBINDEX, returns
     * the Nth Fibonacci number, where N equals FIBINDEX.
     * @param fibIndex The index of the Fibonacci number
     * @return The Fibonacci number
     */
    public static int fibonacci(int fibIndex) {
        if (memoized.containsKey(fibIndex)) {
            return memoized.get(fibIndex);
        } else {
            int answer = fibonacci(fibIndex - 1) + fibonacci(fibIndex - 2);
            memoized.put(fibIndex, answer);
            return answer;
        `
```

Bonnes pratiques

À pratiquer ☺

Différents styles de programmation

Étudiant: ça fonctionne

```
public int fibonacci(int x) {
    if (x == 1) {
        return 1;
    } else if (x == 2) {
        return 1;
    } else {
        return fibonacci(x - 1) + fibonacci(x - 2);
    }
}
```

Optimisation algorithmique au détriment de la compréhension

```
public int getFibonacciNumber(int n) {
    return (int) divide(subtract(exponentiate(phi(), n), exponentiate(psi(), n)),
        subtract(phi(), psi()));
}

public double exponentiate(double a, double b) {
    if (equal(b, zero())) {
        return one();
    } else {
        return multiply(a, exponentiate(a, subtract(b, one())));
    }
}

public double phi() {
    return divide(add(one(), sqrt(add(one(), one(), one(), one(), one()))),
        add(one(), one()));
}

public double psi() {
    return subtract(one(), phi());
}
```

Démonstration: juste pour la démo

```
public int getFibonacciNumber(int n) {
    switch(n) {
        case 1: return 1;
        case 2: return 1;
        case 3: return 2;
        case 4: return 3;
        case 5: return 5;
        case 6: return 8;
        case 7: return 13;
        default:
            // good enough for the demo, lol
            return -1;
    }
}
```

Bonne pratiques de programmation

Utilisez des noms **significatifs** et de façon **cohérente** pour faciliter la compréhension du code

Variables : substantifs représentant la signification ou l'unité

```
double money;   vs.   double salary;
```

Méthodes : verbes à l'infinitif représentant l'action ou le retour

```
getName(), update(), isValid(), hasChildren()
```

Classes : titre substantif représentant le but de l'abstraction

```
Graph, Delivery, RentalHistory, Employee
```

Paquets : URI représentant la hiérarchie et la fonctionnalité

```
org.applicationname.gui,
com.companyname.productname.tier
```

Nom de variables significatifs

Supposons qu'un artéfact de code contient les noms de variables suivants: `freqAverage frequencyMaximum minFr frqncyTotl`

❓ Un nouveau développeur doit savoir si `freq frequency fr frqncy` se réfèrent toutes à la même chose

➢ **Si oui, utilisez le même mot**, préférablement `frequency`, peut-être `freq` ou aussi `frqncy`, mais pas `fr`

➢ **Sinon, utilisez un mot différent** (ex: `rate`) pour une quantité différente

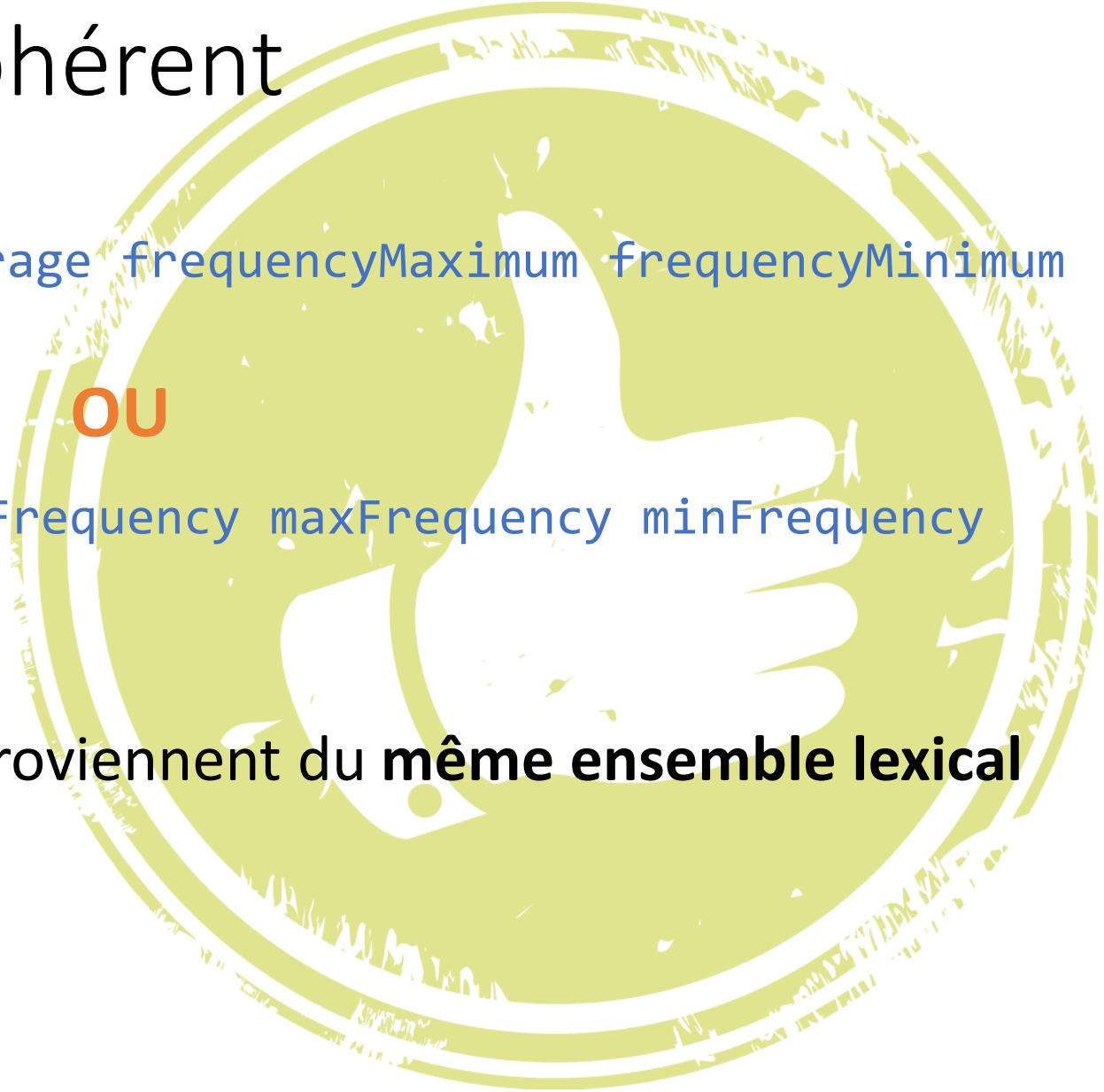
Nom de variables cohérent

- On peut utiliser `frequencyAverage` `frequencyMaximum` `frequencyMinimum` `frequencyTotal`

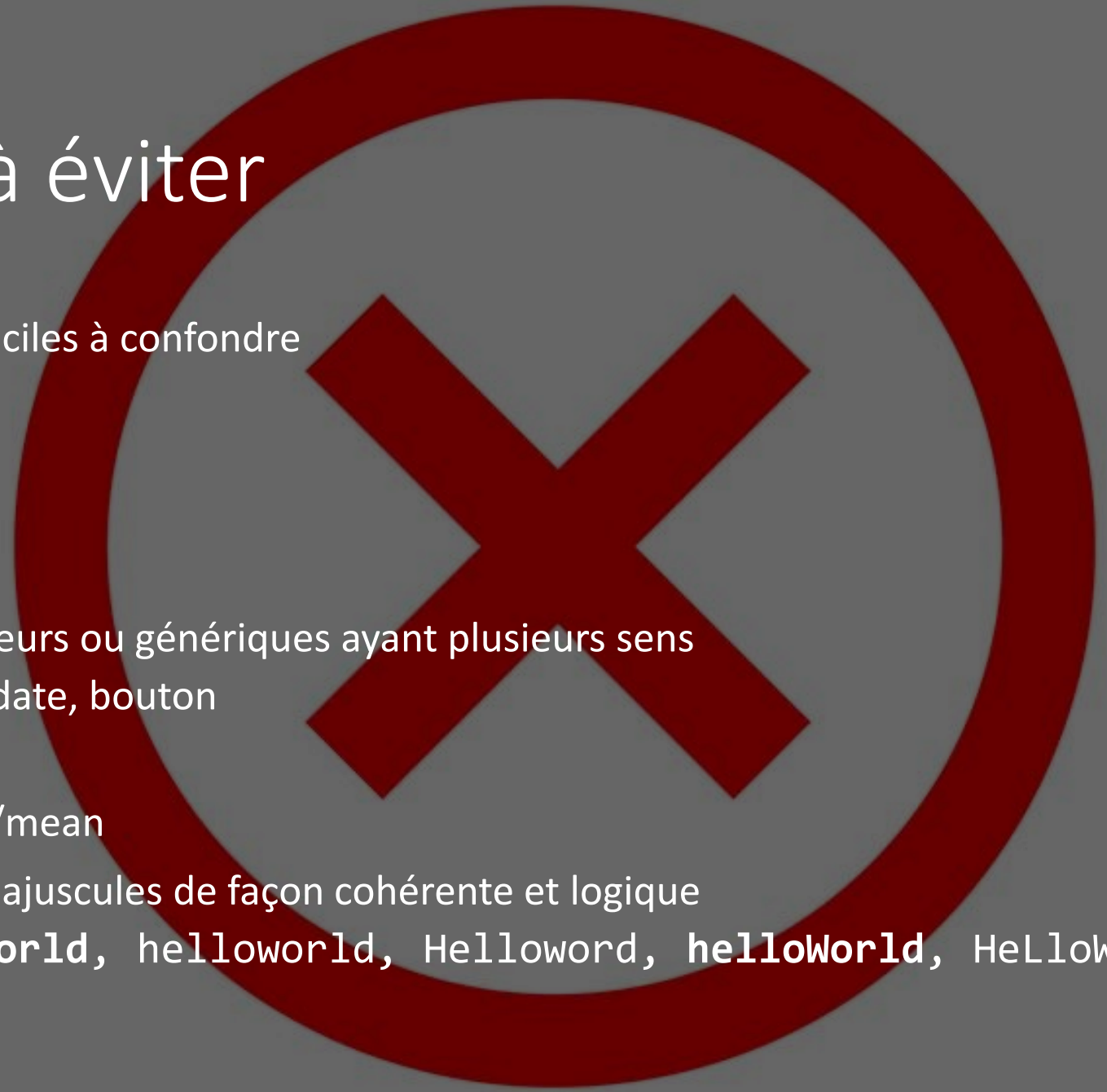
OU

- On peut aussi utiliser `averageFrequency` `maxFrequency` `minFrequency` `totalFrequency`

➢ À condition que les 4 noms proviennent du **même ensemble lexical**



Noms à éviter



- Caractères faciles à confondre
  - 1 l L
  - o O 0
  - S 5
  - G 6
- Noms trompeurs ou génériques ayant plusieurs sens
  - money, date, bouton
- Synonymes
  - average/mean
- Utiliser les majuscules de façon cohérente et logique
  - **`HelloWorld`**`, helloworld, Helloword, `**`helloWorld`**`, HeLloWoRlD, HELLOWORLD`

Nomenclature Java & camelCase

Identifier type	Rules for naming	Examples
Classes	Class names should be nouns in UpperCamelCase, with the first letter of every word capitalised. Use whole words — avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).	•`class Raster;` •`class ImageSprite;`
Methods	Methods should be verbs in lowerCamelCase or a multi-word name that begins with a verb in lowercase; that is, with the first letter lowercase and the first letters of subsequent words in uppercase.	•`run();` •`runFast();` •`getBackground();`
Variables	Local variables, instance variables, and class variables are also written in lowerCamelCase. Variable names should not start with underscore (_) or dollar sign ($) characters, even though both are allowed. This is in contrast to other coding conventions that state that underscores should be used to prefix all instance variables.Variable names should be short yet meaningful. The choice of a variable name should be mnemonic — that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.	•`int i;` •`char c;` •`float myWidth;`
Constants	Constants should be written in uppercase characters separated by underscores. Constant names may also contain digits if appropriate, but not as the first character.	•`static final int MAX_PARTICIPANTS = 10;`

Code Conventions for the Java Programming Language: 9. Naming Conventions (oracle.com)

Disposition du code

- Espaces, lignes blanches
  - Votre code c’est comme un texte technique: séparé en paragraphes, par idées
- Regroupement
  - Méthodes (ou variables) qui collaborent sont rapprochées
  - Grouper par visibilité
- Alignement, indentation, aération
  - Logique complexe: conditions, boucles imbriquées, contenant/contenu
- Parenthèses, accolades
  - Cohérence, mettre en valeur

Obfuscation de code

- Rend le code illisible et incompréhensible
- Utile pour cacher le code remis ou pour l'alléger
  - Ex: JavaScript
- Peut facilement être retranscrit automatiquement

```
m(f,a,s)char*s;
{char c;return f&1?a!=*s++?m(f,a,s):s[11]:f&2?a!=*s++?1+m(f,a,s):1:f&4?a--?
 putchar(*s),m(f,a,s):a:f&8?*s?m(8,32,(c=m(1,*s++,"Arjan Kenter. \no$../.\""),
 m(4,m(2,*s++,"POCnWAUvBVxRsoqatKJurgXYyDQbzhLwkNjdMTGeIScHFmpliZEf"),&c),s)):
 65:(m(8,34,"rgeQjPruaOnDaPeWrAaPnPrCnOrPaPnPjPrCaPrPnPrPaOrvaPndeOrAnOrPnOrP\
nOaPnPjPaOrPnPrPnPrPtPnPrAaPnBrnnsrnnBaPeOrCnPrOnCaPnOaPnPjPtPnAaPnPrPnPrCaPn\
BrAnxrAnVePrCnBjPrOnvrCnxrAnxrAnsrOnvjPrOnUrOnornnsrnnorOtCnCjPrCtPnCrnnirWtP\
nCjPrCaPnOtPrCnErAnOjPrOnvtPnnrCnNrnnRePjPrPtnrUnnrntPnbtPrAaPnCrnnOrPjPrRtPn\
CaPrWtCnKtPnOtPrBnCjPronCaPrVtPnOtOnAtnrxaPnCjPrqnnaPrtaOrsaPnCtPjPratPnnaPrA\
aPnAaPtPnnaPrvaPnnjPrKtPnWaOrWtOnnaPnWaPrCaPnntOjPrrtOnWanrOtPnCaPnBtCjPrYtOn\
UaOrPnVjPrwtnnxjPrMnBjPrTnUjP"),0);}

main(){return m(0,75,"mIWltouQJGsBniKYvTxODAfbUcFzSpMwNCHEgrdLaPkyVRjXeqZh");}
```

Commentaire prologue

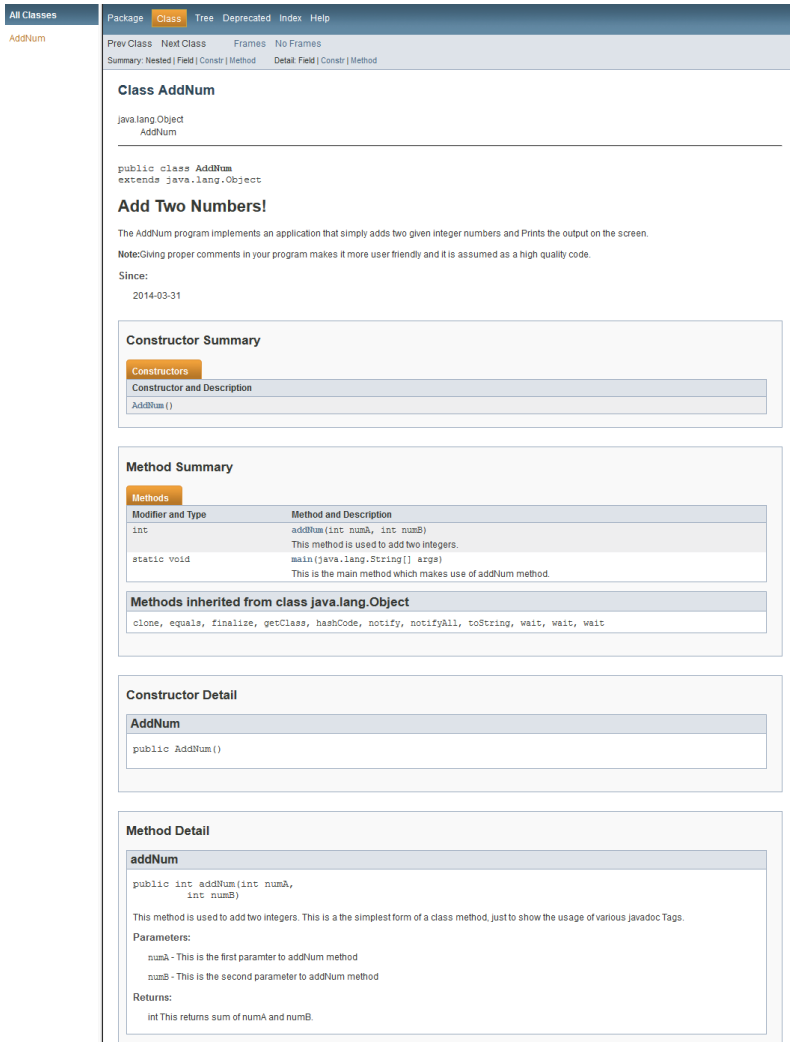
The name of the code artifact
A brief description of what the code artifact does
The programmer's name
The date the code artifact was coded
The date the code artifact was approved
The name of the person who approved the code artifact
The arguments of the code artifact
A list of the name of each variable of the code artifact, preferably in alphabetical order, and a brief description of its use
The names of any files accessed by this code artifact
The names of any files changed by this code artifact
Input–output, if any
Error-handling capabilities
The name of the file containing test data (to be used later for regression testing)
A list of each modification made to the code artifact, the date the modification was made, and who approved the modification
Any known faults

- Au tout **début de chaque artefact** de code (classe ou méthode) on explique chaque variable dans le prologue
- Les autres programmeurs et *futur moi* pourront rapidement **comprendre ce que chaque variable représente**
  - Auteur(s)
  - Dates de création et modification(s)
  - Explications
  - Dépendances
  - License

Autres commentaires

- Insérer des commentaire en ligne (//) pour aider les programmeurs à **comprendre ce que le code fait**
- Les commentaires sont essentiels quand le code est **écrit d'une manière non-évidente**, utilise des **aspects non-communs du langages**, ou dépend d'une **logique complexe**
- Si le code est trop embrouillé, mélangeant, perturbant, il faut le recoder plus clairement
  - **Ne jamais promouvoir ou excuser une programmation médiocre**

Documentation du code



All Classes

AddNum

Package Class Tree Deprecated Index Help

Prev Class Next Class Frames No Frames

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

Class AddNum

java.lang.Object
AddNum

public class **AddNum**
extends java.lang.Object

Add Two Numbers!

The AddNum program implements an application that simply adds two given integer numbers and Prints the output on the screen.

Note: Giving proper comments in your program makes it more user friendly and it is assumed as a high quality code.

Since:

2014-03-31

Constructor Summary

Constructors

Constructor and Description
AddNum ()

Method Summary

Methods

Modifier and Type	Method and Description
int	addNum(int numA, int numB) This method is used to add two integers.
static void	main(java.lang.String[] args) This is the main method which makes use of addNum method.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

AddNum

public AddNum()

Method Detail

addNum

public int addNum(int numA,
int numB)

This method is used to add two integers. This is a the simplest form of a class method, just to show the usage of various javadoc Tags.

Parameters:

numA - This is the first paramter to addNum method

numB - This is the second parameter to addNum method

Returns:

int This returns sum of numA and numB.

```java
import java.io.*;
/**
* <h1>Add Two Numbers!</h1>
* The AddNum program implements an application that
* simply adds two given integer numbers and Prints
* the output on the screen.
* <p>
* <b>Note:</b> Giving proper comments in your program makes
* user friendly and it is assumed as a high quality code.
* @author  Zara Ali
* @version 1.0
* @since   2014-03-31
*/
public class AddNum {
  /**
  * This method is used to add two integers. This is
  * a the simplest form of a class method, just to
  * show the usage of various javadoc Tags.
  * @param numA This is the first paramter to addNum method
  * @param numB  This is the second parameter to addNum meth
  * @return int This returns sum of numA and numB.
  */
  public int addNum(int numA, int numB) {
    return numA + numB;
  }
  /**
  * This is the main method which makes use of addNum method
  * @param args Unused.
  * @return Nothing.
  * @exception IOException On input error.
  * @see IOException
  */
  public static void main(String args[]) throws IOException
    AddNum obj = new AddNum();
    int sum = obj.addNum(10, 20);
    System.out.println("Sum of 10 and 20 is :" + sum);
  }
}
```

- Permettent de générer automatiquement la documentation de l'API du programme
- **javadoc** produit un fichier HTML (voir démo)

Normes de programmation

- Les normes peuvent être à la fois une bénédiction et une malédiction
- Modules peux cohésifs sont souvent causer par des règles telles que: *“Chaque module doit contenir entre 35 et 50 instructions exécutables”*
- Meilleure règle: *“Le programmeur doit consulter sont supérieur avant de construire un module de moins de 35 ou de plus de 50 instructions exécutables”*
- **Aucune norme ne sera jamais universellement applicable**
- Les normes telles que la 1ère seront ignorées
- Idéalement, une norme devrait être **vérifiable sans intervention humaine**
- Le but de standardiser est de rendre la **maintenance plus facile**

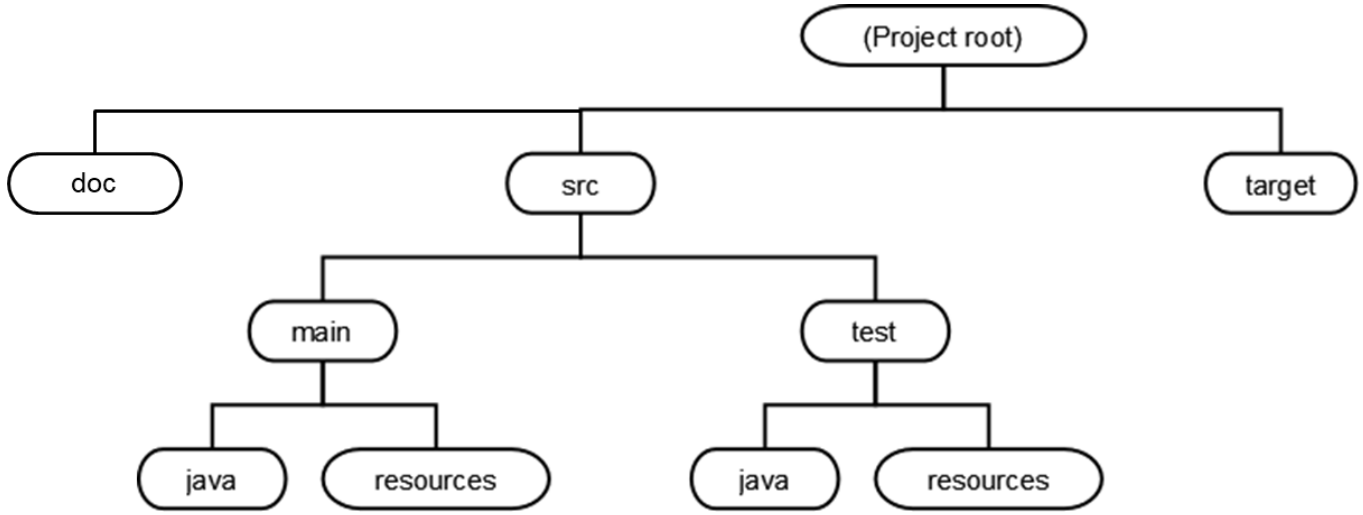
Exemple de bonnes normes

- *L’imbrication d’expressions SI ne doit pas dépasser 3 niveaux, sauf avec approbation*

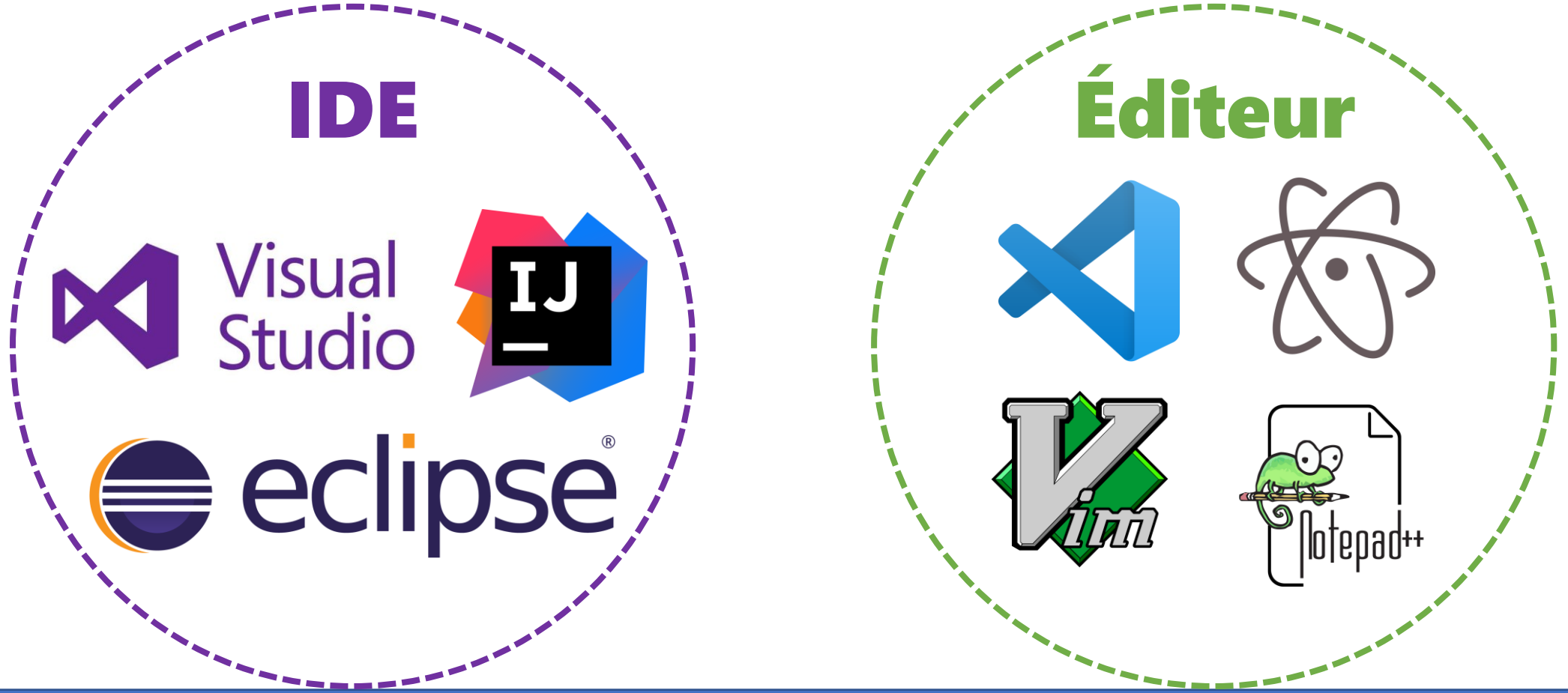
- *Éviter l’utilisation des GOTO. Néanmoins, avec l’approbation du chef d’équipe , un GOTO vers l’avant peut être utilise pour le traitement d’erreur*

Organiser un gros projet Java

- Organisation structurée des répertoires et fichiers
  - `src/main/java/` : code source
  - `src/test/` : tous le code et autres fichiers pour les tests. Doit correspondre à la structure de `main/`
  - `src/main/resources/` : fichers textes, images, etc.
  - `target/` : fichiers classes compilés
  - `doc/` : la documentation générée



http://cs.lmu.edu/~ray/notes/largejavaapps/

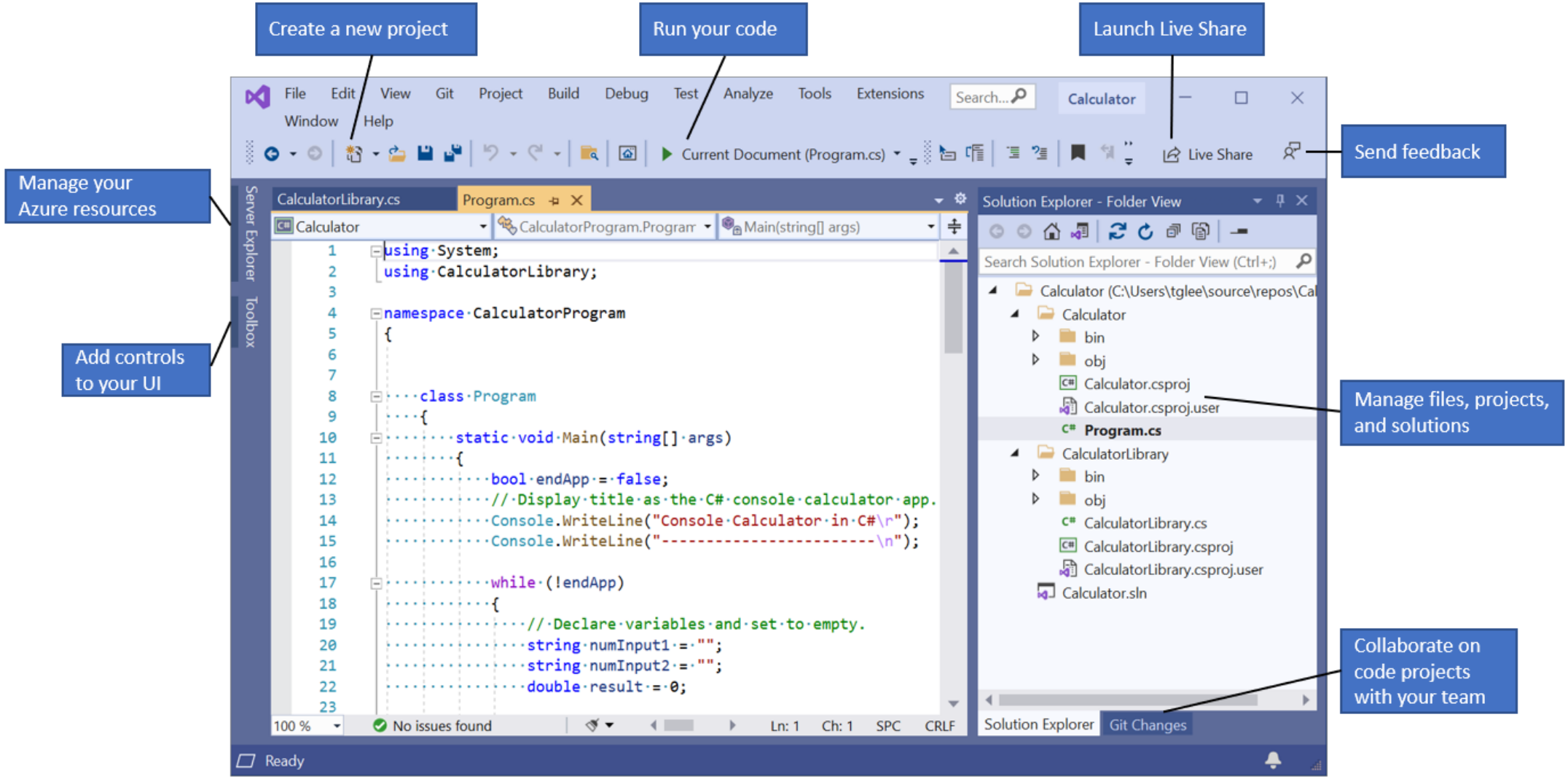


Outils de programmation

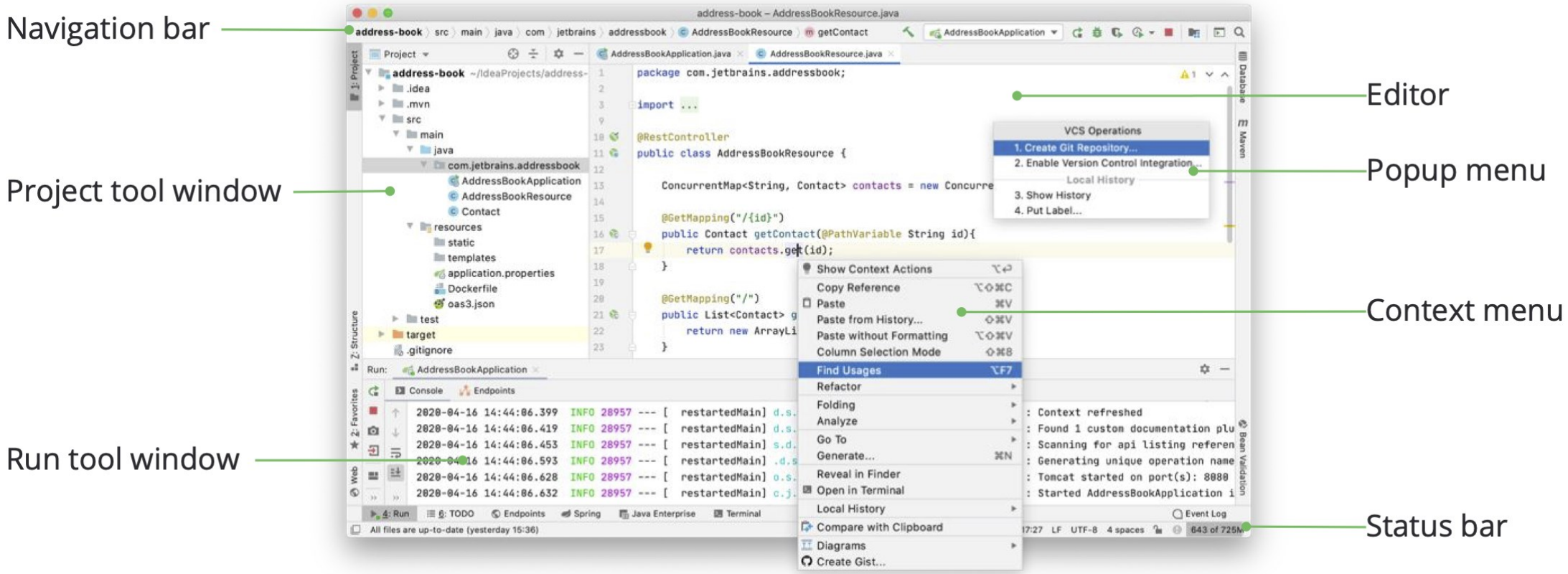
Environnement de développement intégré (IDE)

- Les développeurs utilisent différents outils durant la vie du logiciel
  - Vérificateur d'interfaces, compilateur, éditeur texte, etc.
- Les outils peuvent être combiné dans un **environnement** qui apporte un support par ordinateur à plusieurs activités
  - Programmation, test, contrôle de configuration, gestion de révisions, moteur de production, interpréteur, débogueur
- Une **IDE** intègre ces environnements et outils dans une interface utilisateur commune et uniforme
  - Même présentation
  - Outils communiquent avec des données compatibles
- Idéalement, une IDE devrait **intégrer tout le processus** de développement

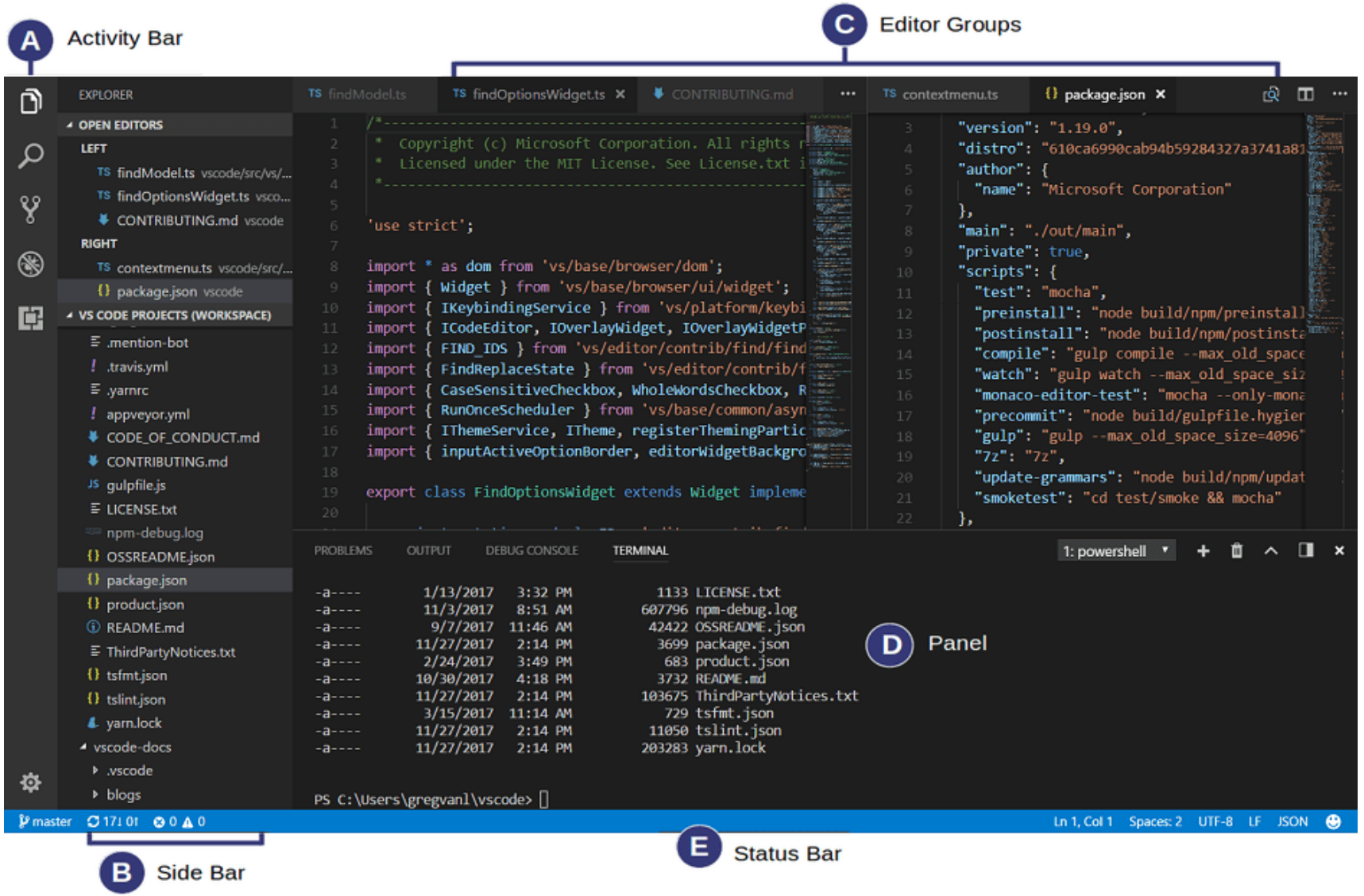
Utilisation d'un IDE: Visual Studio



Utilisation d’un IDE: IntelliJ IDEA



Utilisation d’un éditeur: VSCode




Onsite PM
TEST Team
T Lead
P Manager
SOLID
BORN BRAVE
Developer
BUG

Débogage

Débogage

- Processus méthodique de trouver et réduire le nombre de **défauts** dans un programme **lors de l'exécution** afin qu'il se comporte tel qu'attendu
- Un débogueur **simule l'exécution** du code à examiner en pouvant l'exécuter et le **suspendre** lorsque des conditions spécifiques sont rencontrées
- Caractéristiques:
  - **Exécution en étapes** *(step)*: animation du programme exécutant une expression à la fois
  - **Suspension** *(break)*: faire une pause afin d'examiner l'état courant

Sévérité des défauts

- **Bloqueur:** empêche de poursuivre les tests jusqu'à ce qu'il soit corrigé ou une alternative est identifiée
- **Critique:** impossible d'éviter la perturbation d'opérations essentielles; sécurité compromise
- **Majeur:** opération essentielle est affectée, mais on peut continuer
- **Mineur:** opération non-essentielle est perturbée
- **Inconséquent:** pas d'impact significatif sur les opérations

IEEE Standard 1044-2009:
IEEE Standard Classification for Software Anomalies

Méthodes de débogage

Assertions	Exceptions	Traçage
Log manuel	Débogeur intéractif	Débogage en direct

Assertion

- Vérifier une condition lors de l'exécution et mettre fin au programme en cas d'échec
- Utilisé dans les tests unitaires

```
int total = countNumberOfUsers();
if (total % 2 == 0) {
    // total is even
} else {
    // total is odd and non-negative
    assert total % 2 == 1;
}
```

Exception

- Détecter une erreur logique ou un cas extrême
- **Lancer** une exception quand une erreur se produit
- **Traiter** une exception pour corriger l'erreur
- Continuer l'exécution, arrêter l'exécution, ou propager (récursivement) l'exception au module supérieur

```
try {
    line = console.readLine();

    if (line.length() == 0) {
        throw new EmptyLineException("The line read from console was empty!");
    }

    console.printLine("Hello %s!" % line);
    console.printLine("The program ran successfully");
}
catch (EmptyLineException e) {
    console.printLine("Hello!");
}
catch (Exception e) {
    console.printLine("Error: " + e.message());
}
finally {
    console.printLine("The program terminates now");
}
```

Traçage

- La **trace d'appels** *(stack trace)* retrace l'historique d'exécution du programme
- Garde une trace de tous les appels de méthodes faits

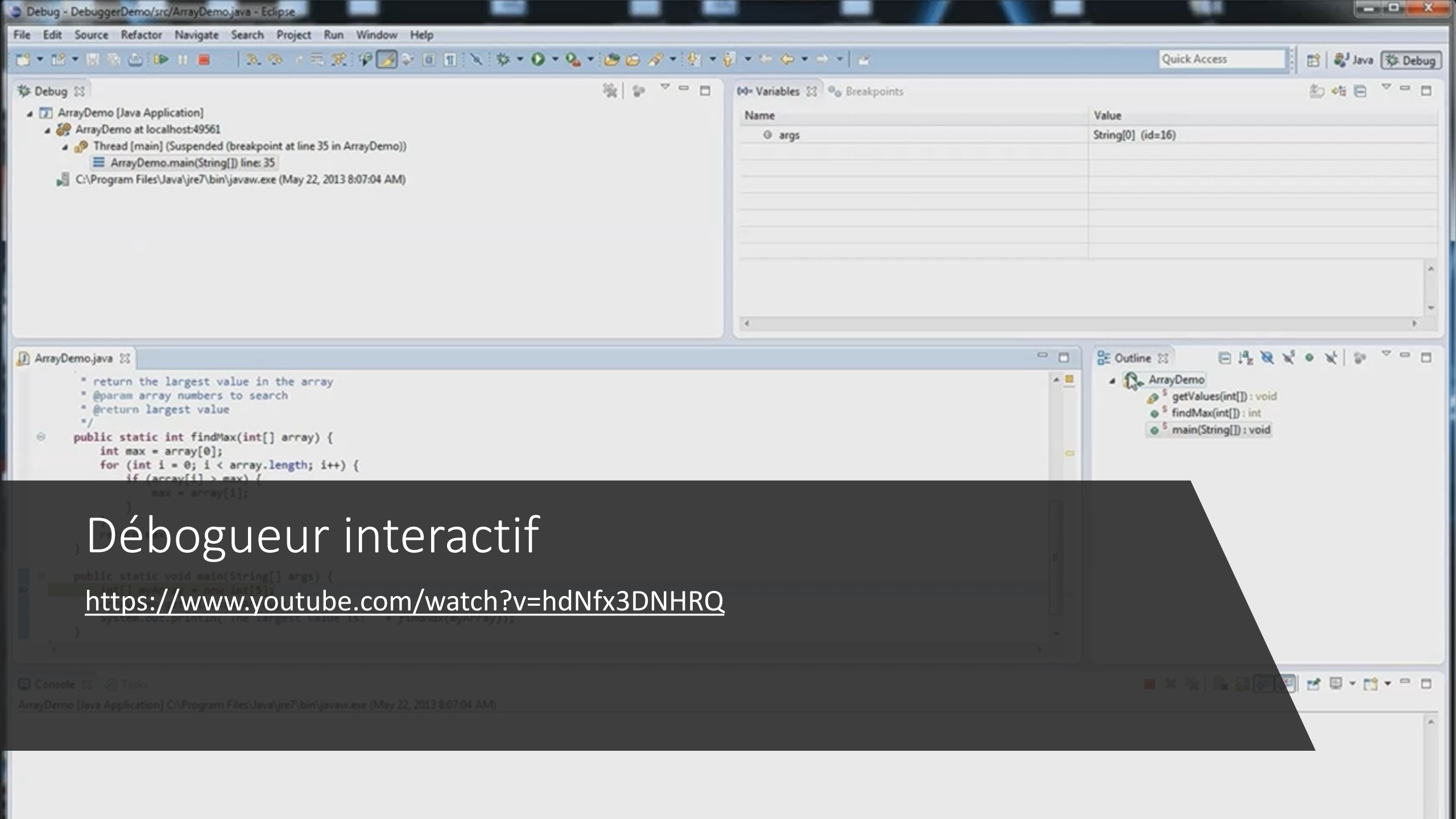
```
def a():
    b()

def b():
    c()

def c():
    erreur()

a()
```

```
Traceback (most recent call last):
  File "tb.py", line 10, in <module>
    a()
  File "tb.py", line 2, in a
    b()
  File "tb.py", line 5, in b
    c()
  File "tb.py", line 8, in c
    error()
NameError: global name 'erreur' is not defined
```

Log manuel

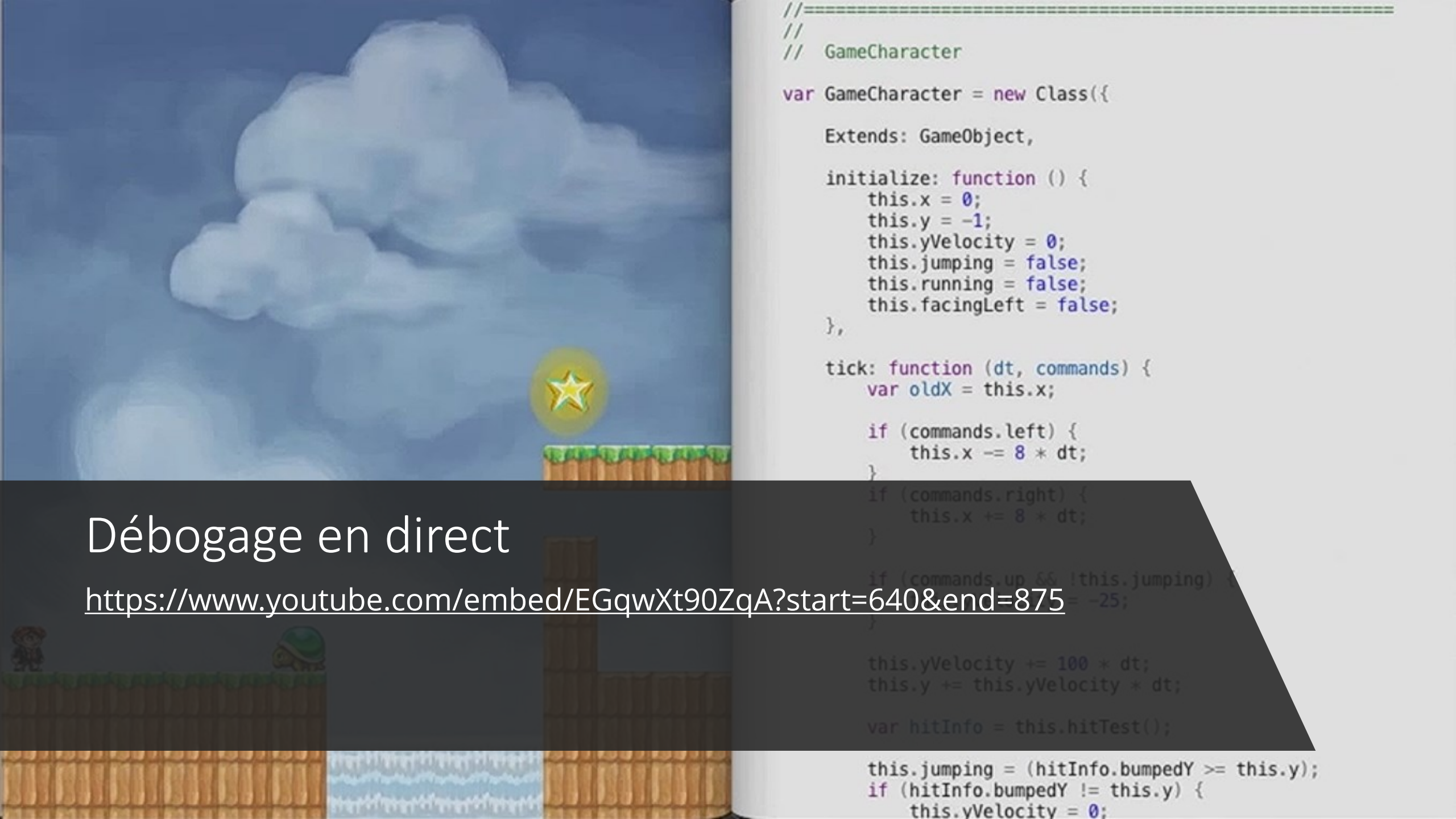
- Afficher l'information sur **l'état** du programme ou son **flux de contrôle** dans un log
- Instructions manuellement insérées dans le code
- S'affiche dans la console, un fichier, etc.

```
import java.util.logging.*;
class myClass {
  private Logger logger = Logger.getLogger(this.getClass().getPackage().getName());
  void m() {
    logger.severe("Message de haute sévérité");
    logger.warning("Message d'alerte");
    logger.info("Message d'information");
  }
}
```



Débogueur interactif

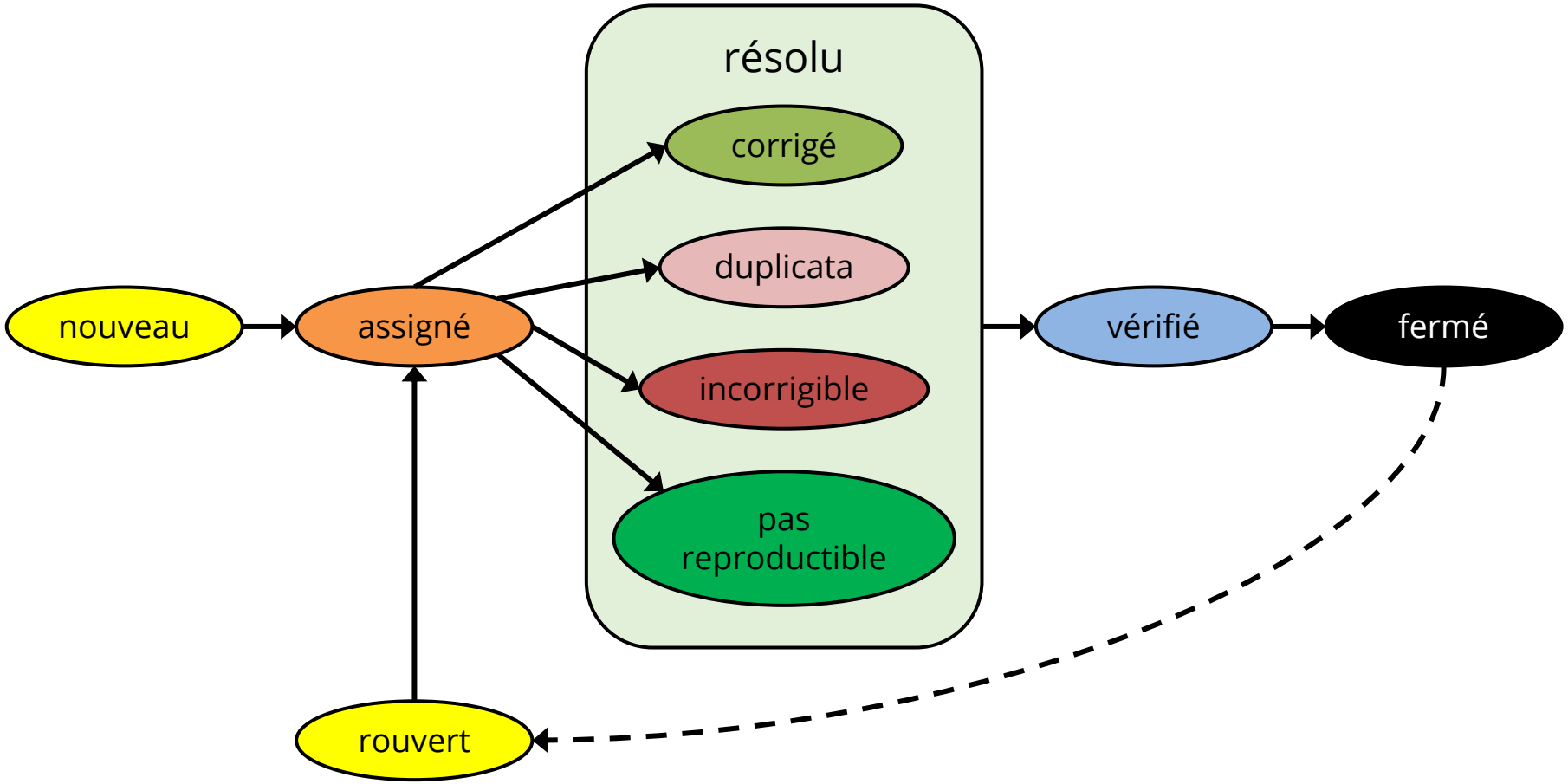
https://www.youtube.com/watch?v=hdNfx3DNHRQ



Débogage en direct

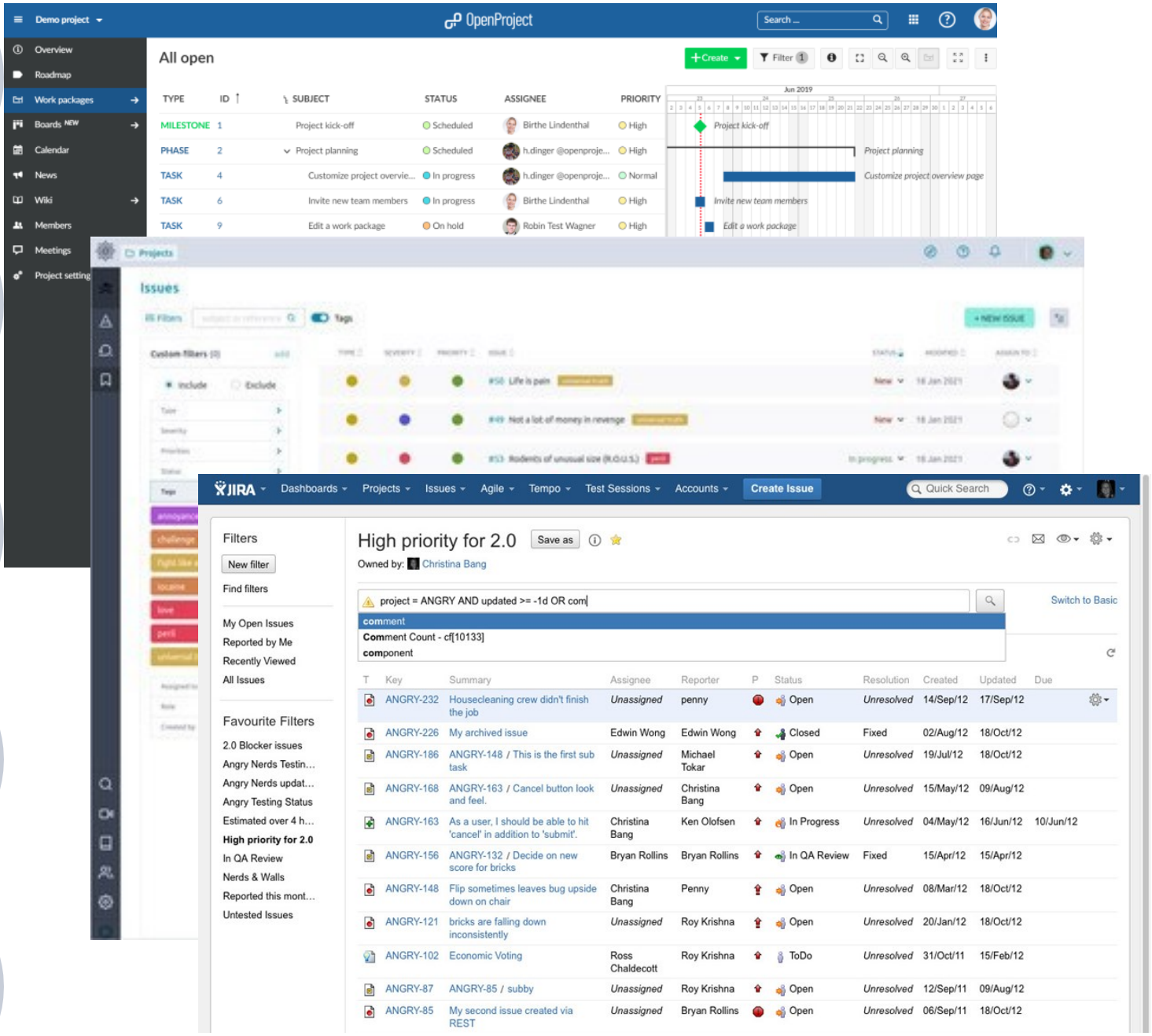
https://www.youtube.com/embed/EGqwXt90ZqA?start=640&end=875

Cycle de vie d’une bogue

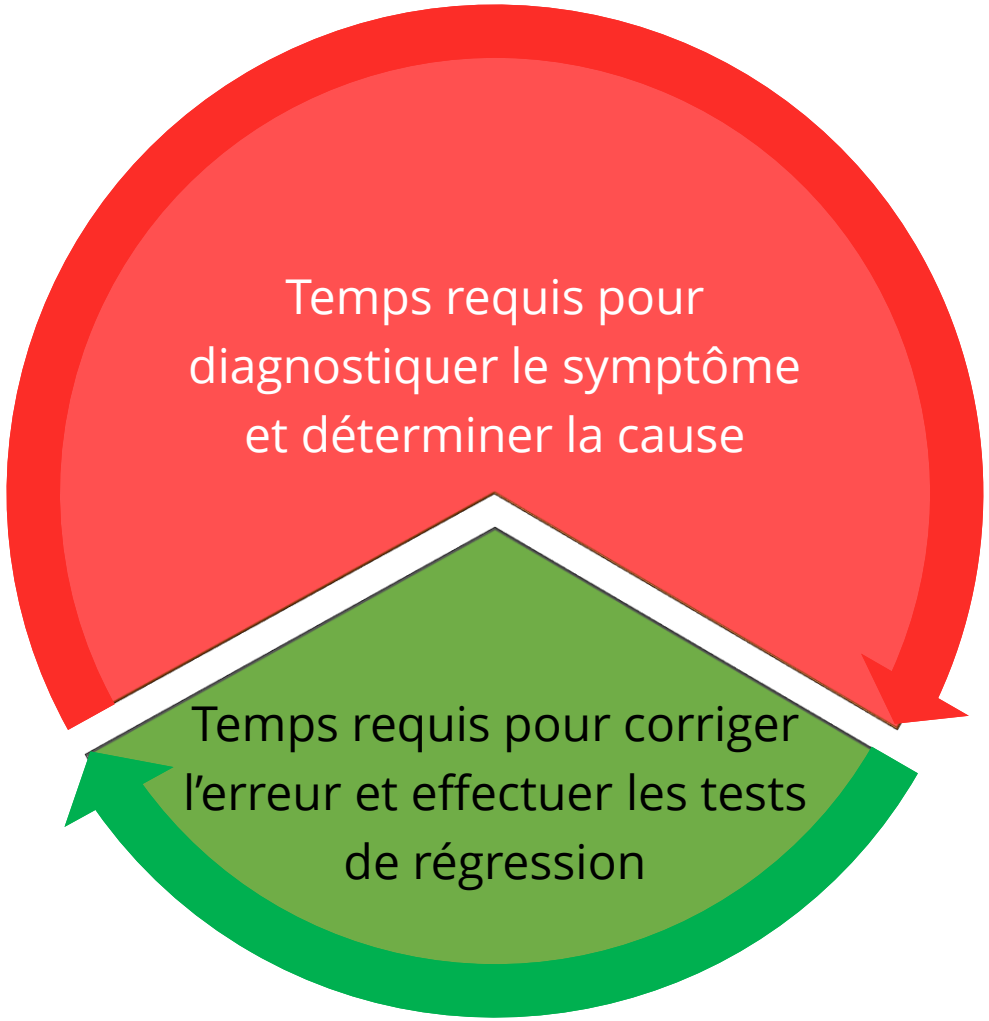


Logiciel de suivi de bogues

- JIRA
- Taiga
- OpenProject



Effort de déboggage



Symptômes et causes

- Symptômes et causes peuvent être **séparés géographiquement**
- Symptôme peut disparaître quand un **autre problème** est corrigé
- Symptôme peut être **intermittent**
- Cause peut être due à une **combinaison de manipulations correctes**
- Cause peut être due à une **erreur système** ou du **compilateur**
- Cause peut être due à une **hypothèse** que l'on croit vraie