

# Théorie des langages : syntaxe et sémantique

PILS

“JAMAIS LA PROGRAMMATION N'A ÉTÉ AUSSI POPULAIRE”



James Ortiz

james.ortizvega@unamur.be

Année académique  
2020- 2021


INFOB314/IHDCB332

# **Mission Briefing**



# Objectifs

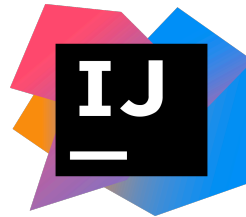
---

- Travail par groupe 
- Mise en pratique des concepts vus au cours
- Lecture et compréhension d'une spécification semi-formelle
- Construction d'un compilateur correct
- Utilisation d'outils de développement



**git**

***maven***



**JUnit**

**ANTLR**

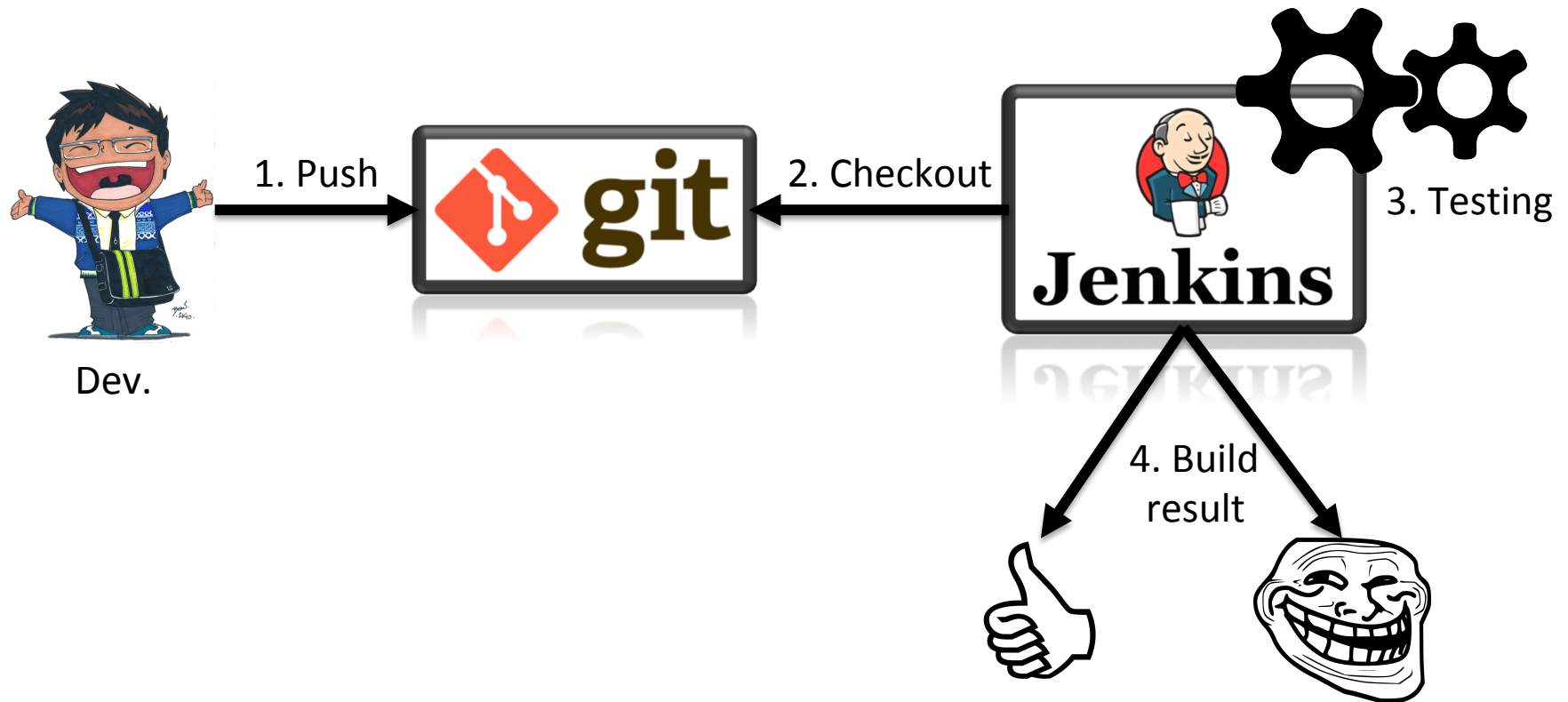
# Méthodologie

---

- Travail à partir d'une spécification semi-formelle
- Evaluation
  - Fonctionnalités aux échéances
    - 7 mars, 11 avril et 11 mai
  - Choix techniques et architecturaux
  - Qualité du code et des tests
  - Rapport + code source documenté
    - 13 mai 2021
  - Interrogation individuelle en juin
  - Etc.

# Evaluation des Fonctionnalités aux échéances via un serveur CI

---



# Environnement de développement (IDE)

---

- Un **environnement de développement** est un ensemble d'outils pour augmenter la productivité des programmeurs qui développent des logiciels,
- Un IDE comporte typiquement une interface graphique pour lancer les différents outils, un éditeur pour le code source, un compilateur, un débogueur, ainsi que, souvent, un outil permettant de construire les interfaces graphiques des logiciels,
- Des environnements de développement populaires:
  - Eclipse,
  - [IntelliJ IDEA](#),
  - NetBeans,

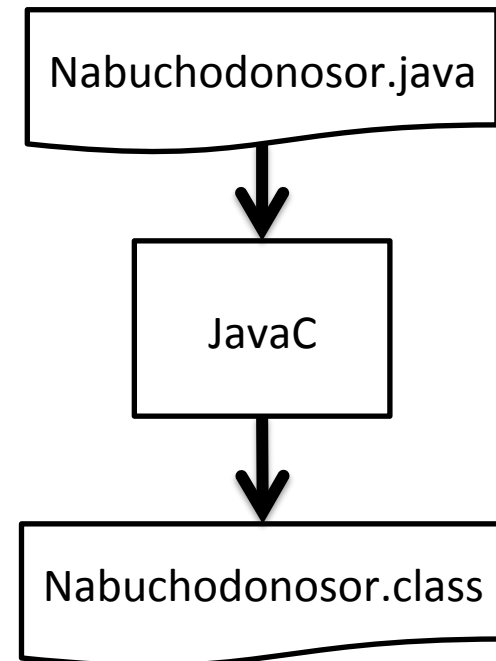
**Compiler, WTF?!?**



# Compilateur

---

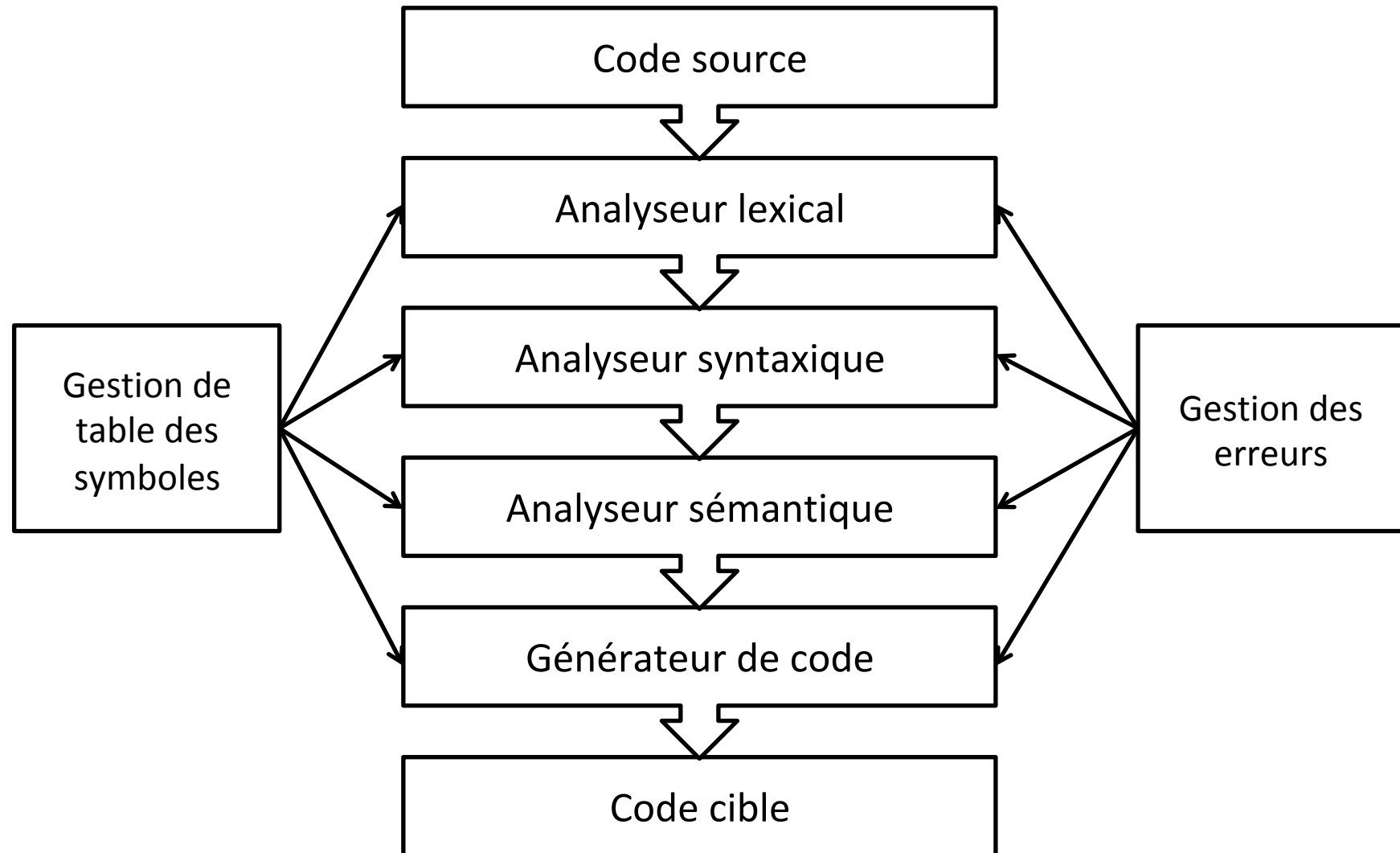
- Traduit
  - un code source
  - écrit dans un langage source
    - langage de programmation, DSL, etc.
- en
  - Un programme cible
  - écrit dans un langage cible
    - langage machine, langage de programmation, etc.
- Attention : un compilateur ne fait que traduire du code, il ne l'exécute pas (<> interpréteur) !
- Exemples : javac, gcc, gpc, etc. ... et ANTLR





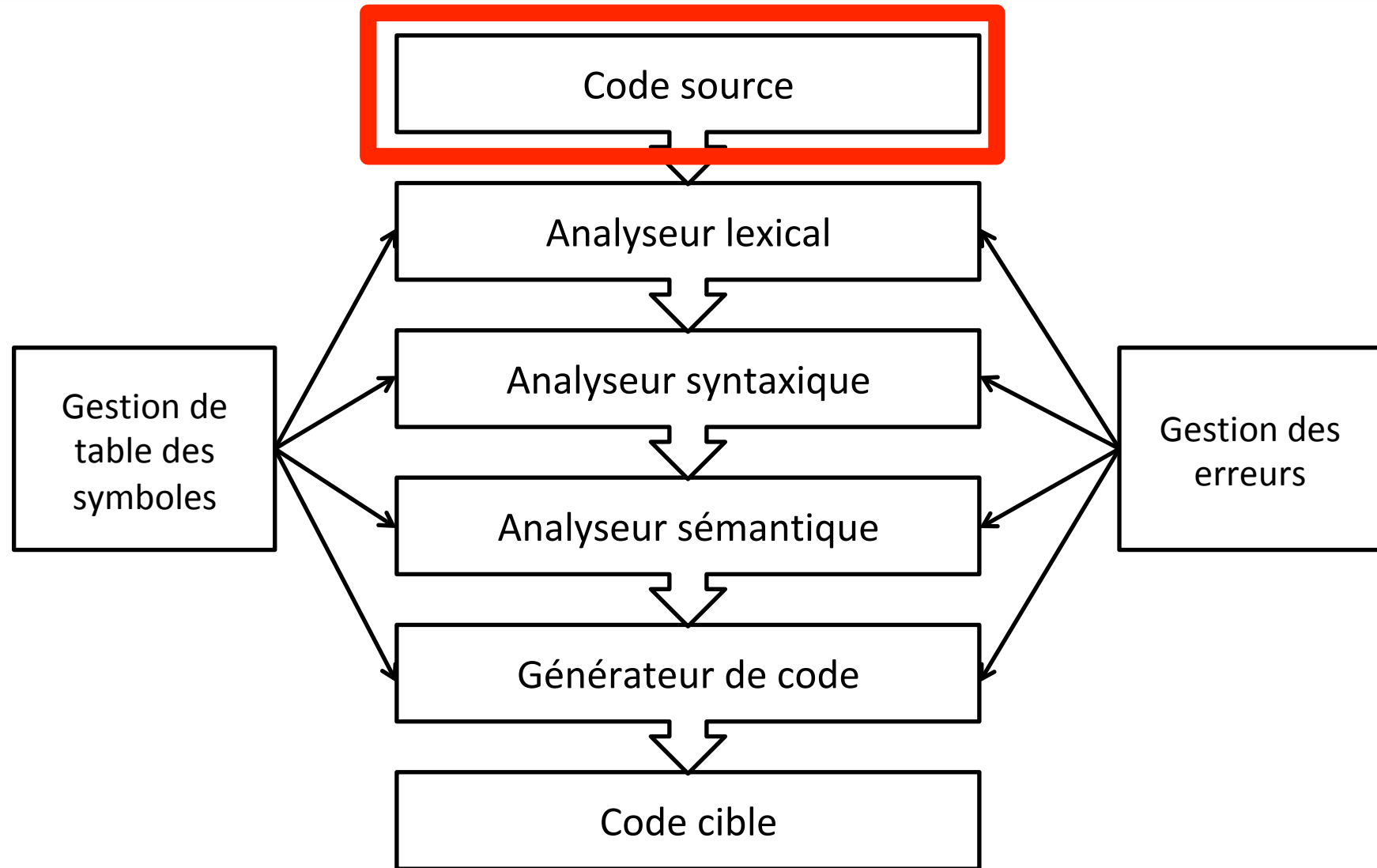
# Compilateur

---



# Compilateur

---



# Programme source

---

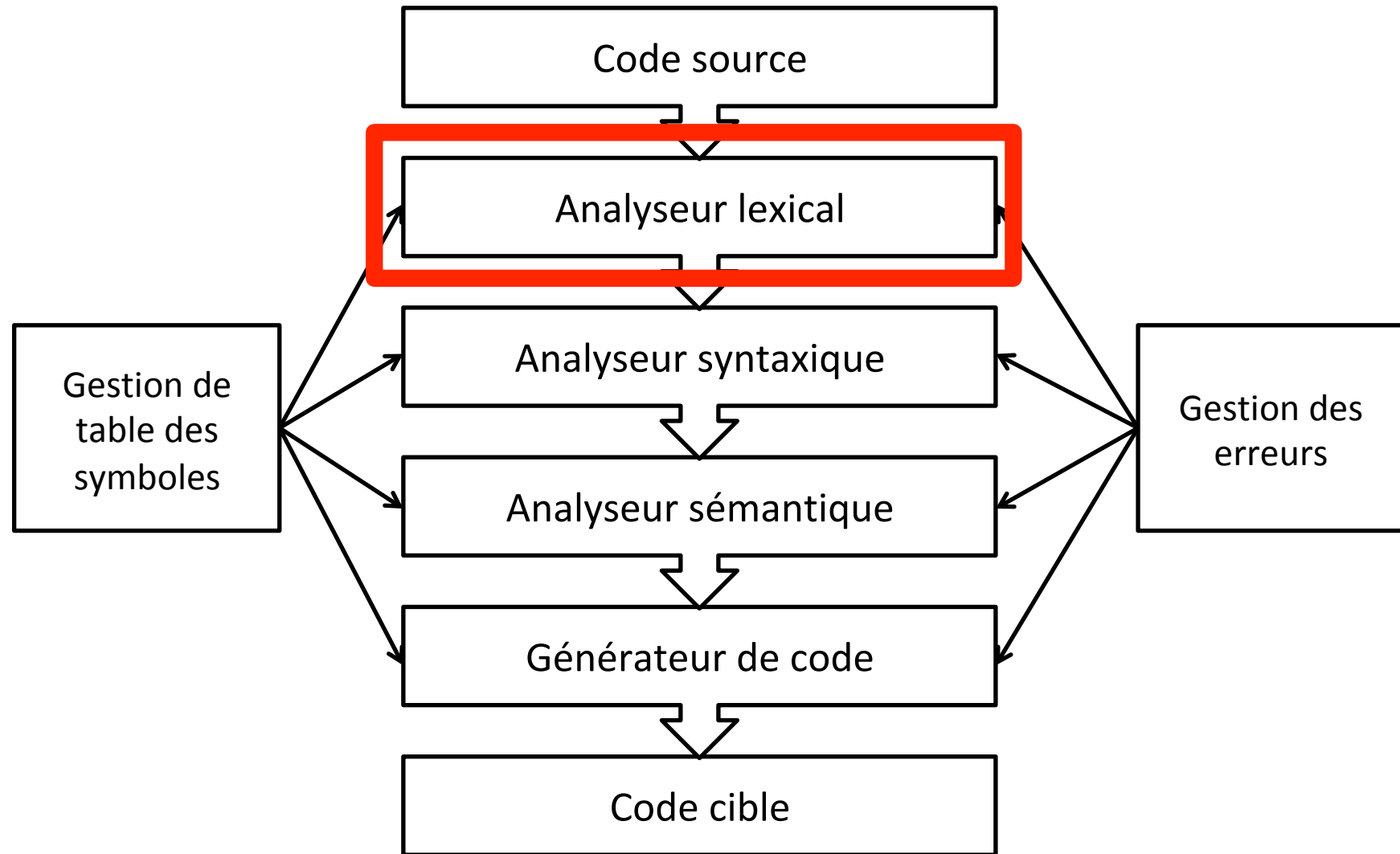
- Décrit dans le document : “Spécification du langage”
- Exemple:

```
// Input map file */
#import "woldExample .map"

// Execute procedure
void execute(){
    left (1);
    down(2);
    right (3);
    up(2);
}
// Main procedure
void main(){
    execute();
    dig();
}
```

# Compilateur

---



# Analyseur lexical

---

- But : vérifier que tous les mots et symboles appartiennent bien au langage
- Exemples de mots (ou lexèmes) :
  - Mots réservés : `if, else, +, *, etc.`
  - Identificateurs : `a, toto, etc.`
  - Constantes : `0, 1, true, false, etc.`
- Peuvent être décrits au moyen d'expressions régulières

# Expression régulière

---

Symbole	Signification
" "	Une chaîne de caractères entourée (éventuellement) de double-quotes représente la chaîne elle-même
[ ]	Une chaîne de caractères entre crochets représente un de ses éléments. Exemple : [abc] représente "a" ou "b" ou "c"
.	tout caractère sauf \n
	Opérateur d'alternance. Exemple : "a" "b" "c" = [abc]
*	Opérateur de répétition : zéro ou plusieurs fois
+	Opérateur de répétition : au moins une fois
?	Opérateur d'occurrence : zéro ou une fois
^ et \$	Début de ligne et fin de ligne
{ }	Opérateur de répétition bornée. Exemple : a{3} = "aaa"

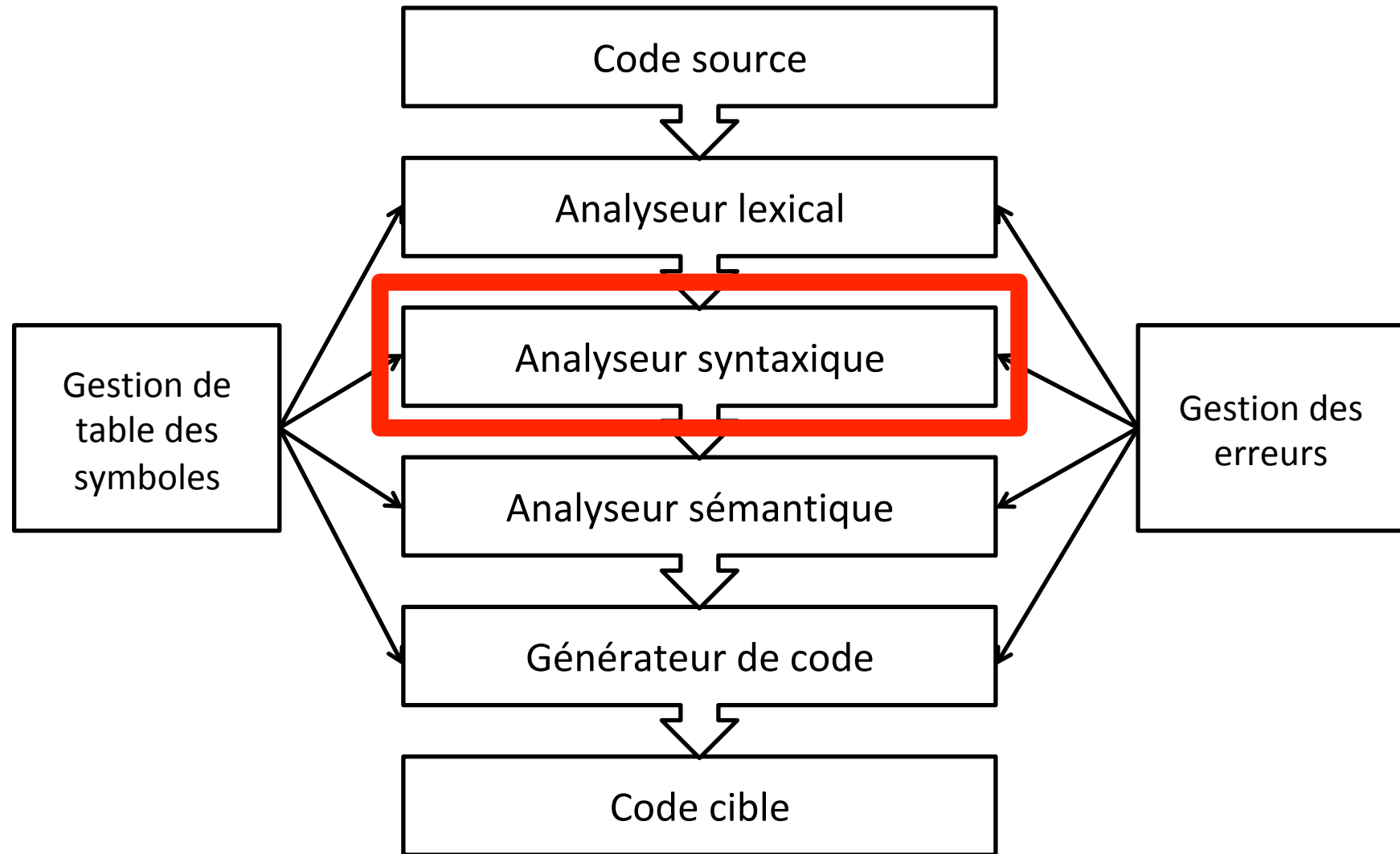
# ANTLR: LexerRules.g4

---

- Une bonne pratique avec ANTLR consiste à regrouper les lexèmes dans un même fichier
- Mots clés :  
IF: 'if';
- Identifiants :  
NAME: LETTER (LETTER | DIGIT)\* ;  
fragment LETTER: 'A'..'Z' | 'a'..'z' | '\_' ;  
fragment DIGIT: '0'..'9' ;
- Espaces :  
NEWLINE: '\r'? '\n' -> skip ;  
WS: [ \t]+ -> skip ;

# Compilateur

---





# Analyseur syntaxique

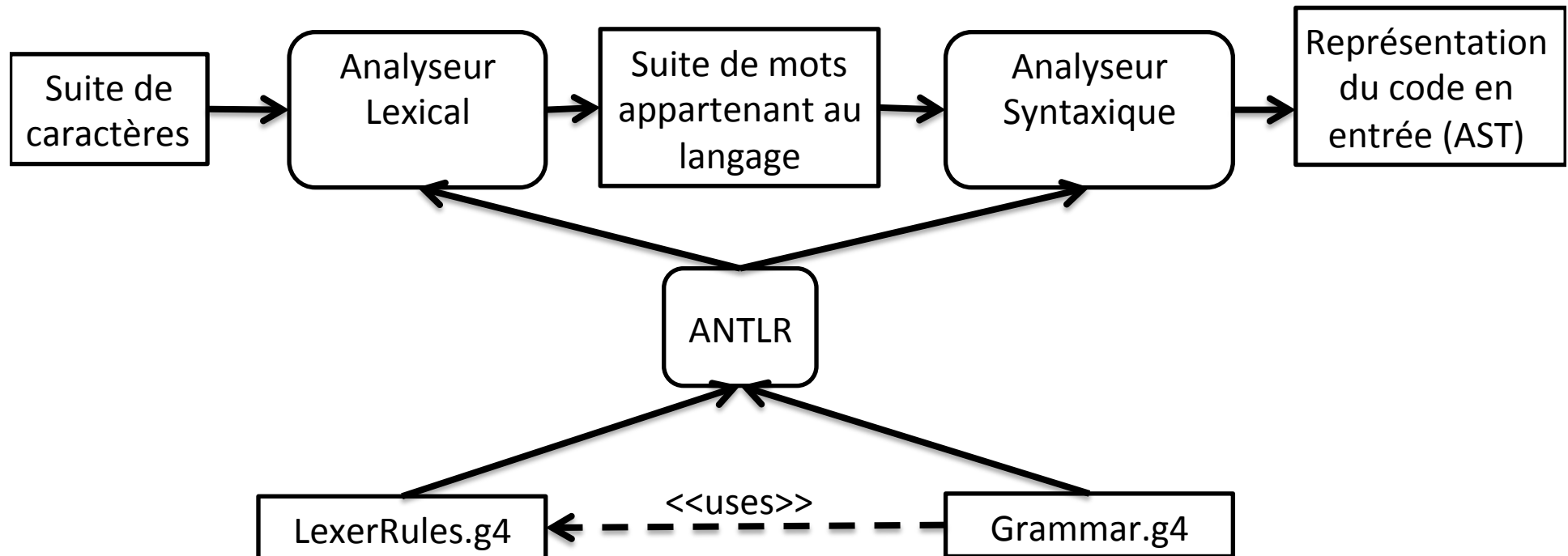
---

- But : vérifier que les enchainements de mots et symboles forment bien des phrases appartenant au langage
- Exemple de phrase :  
if ... else
- Tout comme les mots clés du langage, il est possible d'utiliser un outil pour décrire de manière finie l'ensemble infini des phrases du langage

# Analyseur syntaxique

---

- Construit automatiquement par ANTLR (en même temps que l'analyseur lexical)



## Exemple : Watch.g4

---

```
grammar DEMO;
import LexerRules;

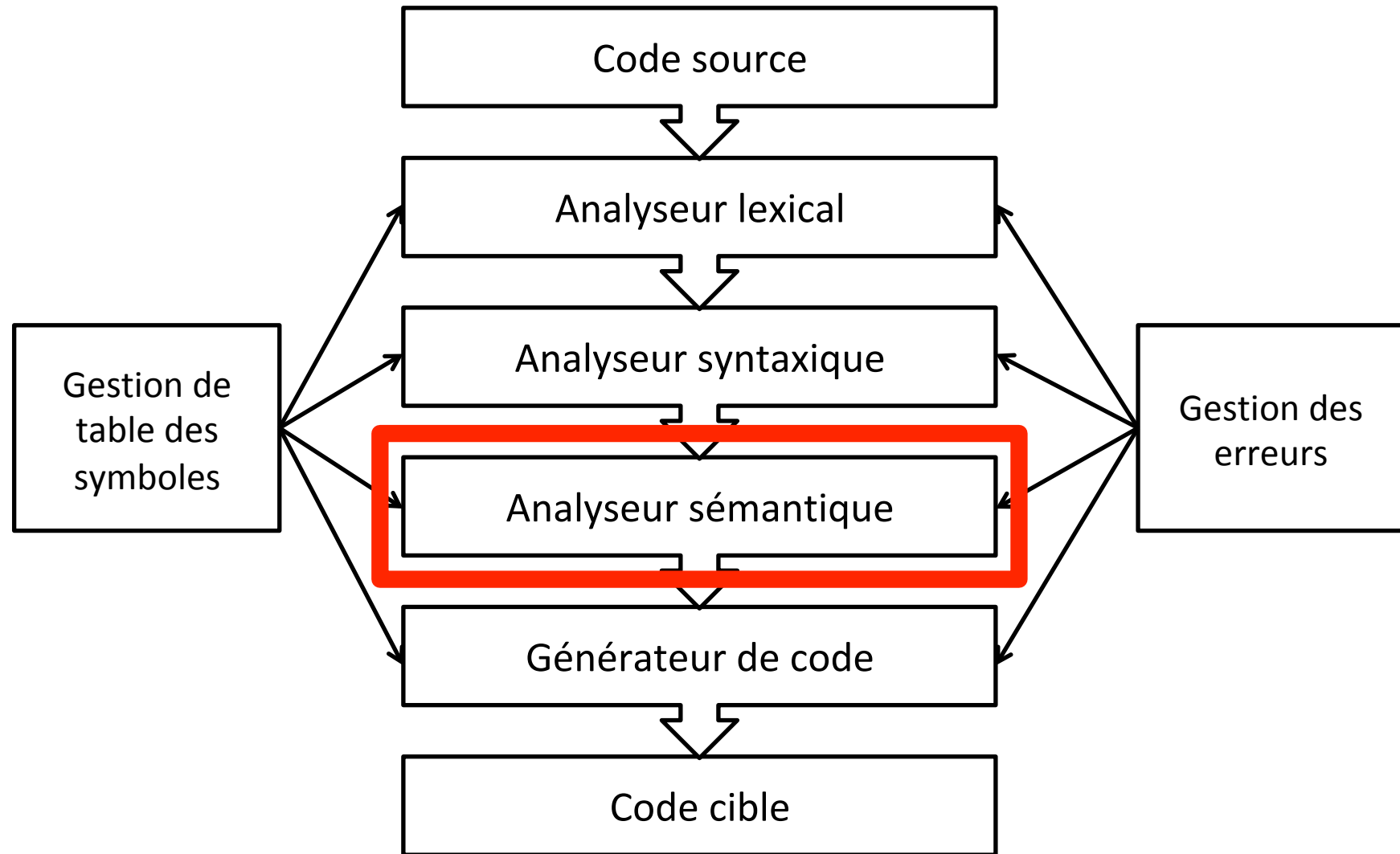
demo: instruction*;

instruction:
    AFFECT LPAR ID COMMA expression RPAR;

expression: NUMBER
| ID
| expression op=(PLUS|MINUS) expression;
```

# Compilateur

---



# Analyseur sémantique

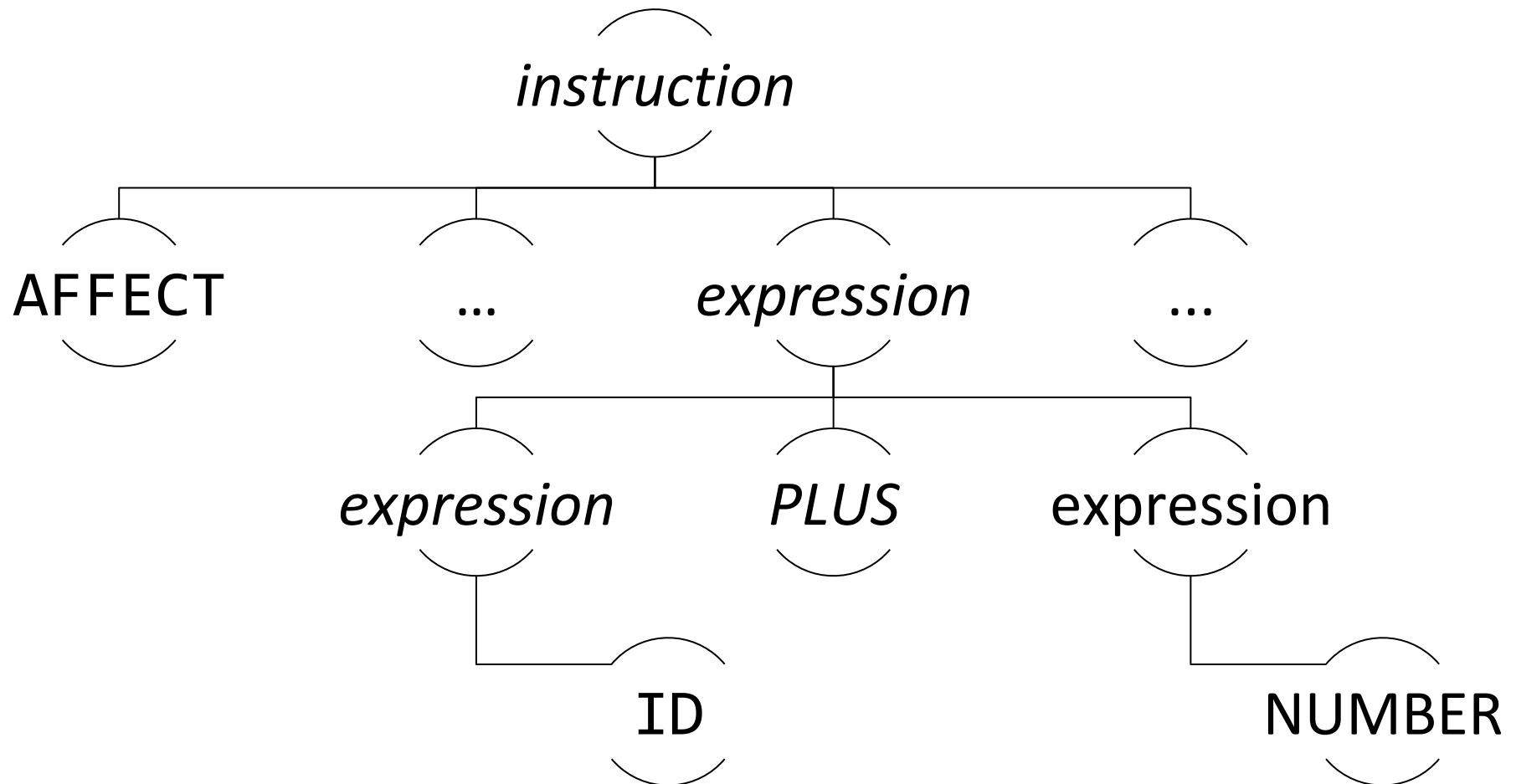
---

- But : Vérifier toutes les autres contraintes
- Exemples :
  - compute x + 2 \* y /\*x et y doivent être entiers et visibles\*/
  - set a to 42 /\*types de l'ExpG doit correspondre au type de l'ExprD\*/

# Arbre syntaxique abstrait (AST)

---

- N-tree représentant le programme compilé



# Table des symboles

---

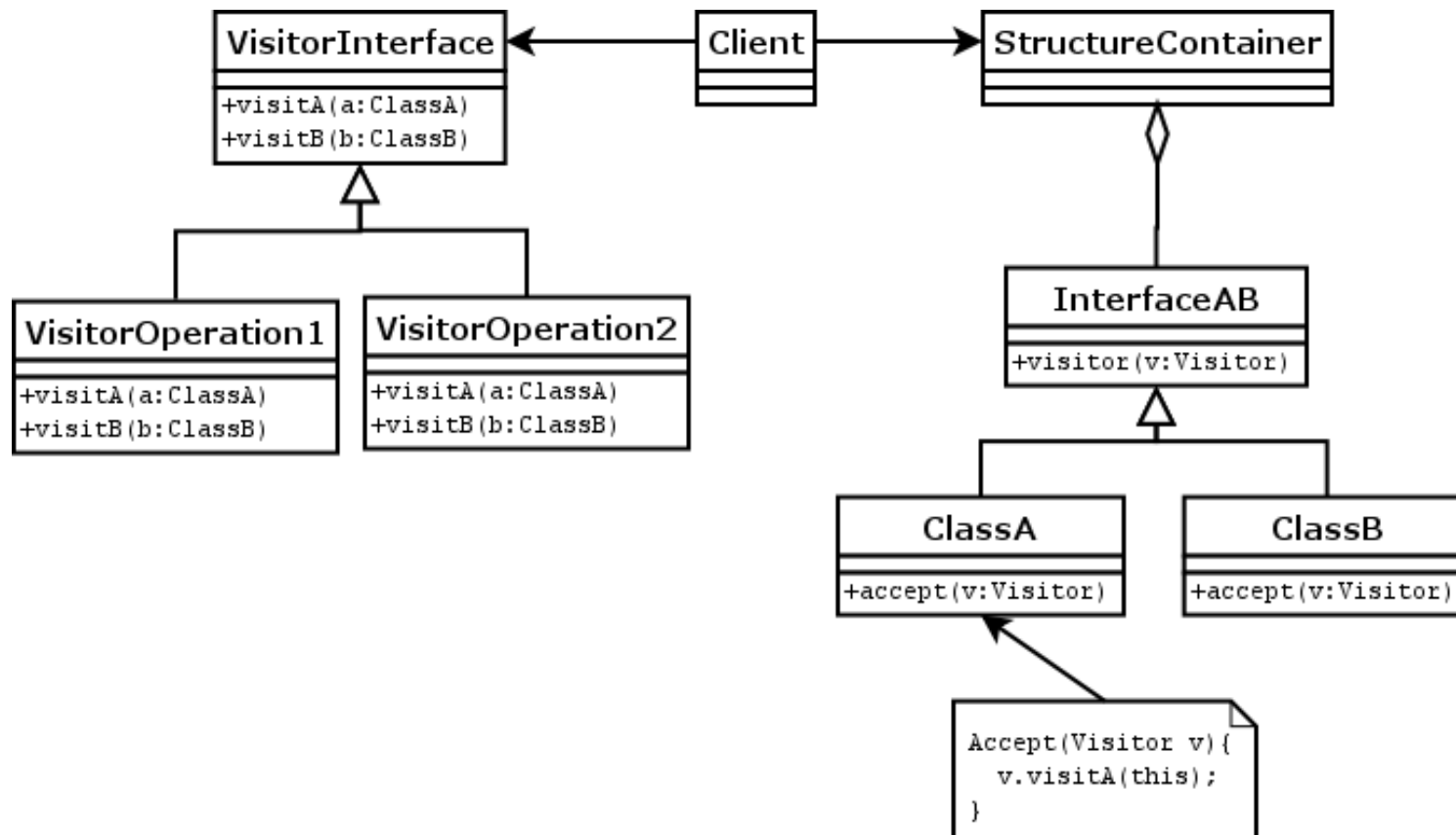
- Représentation des variables (et fonctions) déclarées
- Pour info: un type abstrait de table des symboles est déclaré au chapitre 6 de [5]:

```
public interface Scope {  
    public String getScopeName();  
    public Scope getEnclosingScope();  
    public void define(Symbol sym);  
    public Symbol resolve(String name);  
}
```

```
Scope symTable = new MyScope();
```

# Vérification des types

- En utilisant (par exemple) le *design pattern visitor*

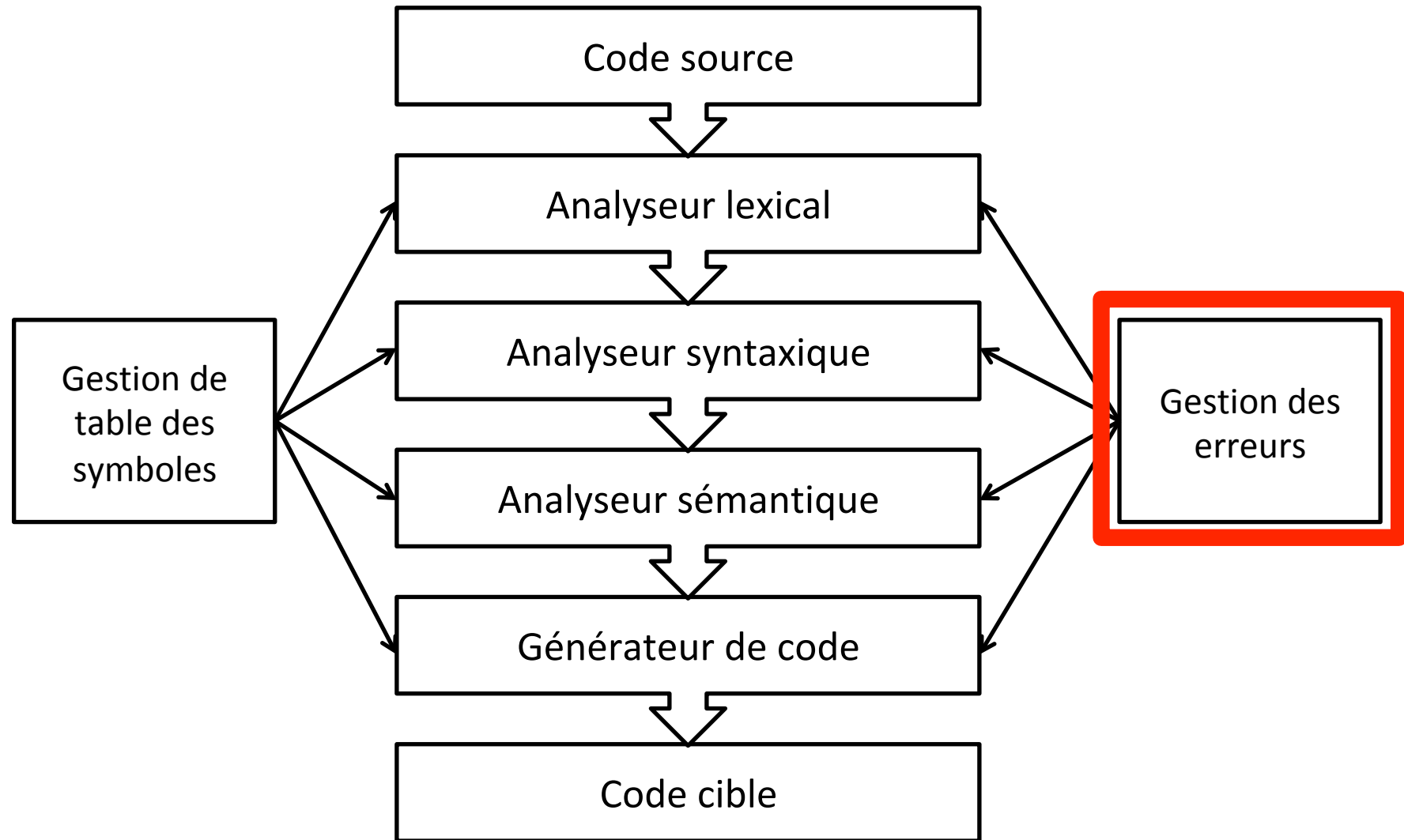


[https://fr.wikipedia.org/wiki/Visiteur\\_\(patron\\_de\\_conception\)](https://fr.wikipedia.org/wiki/Visiteur_(patron_de_conception))



# Compilateur

---



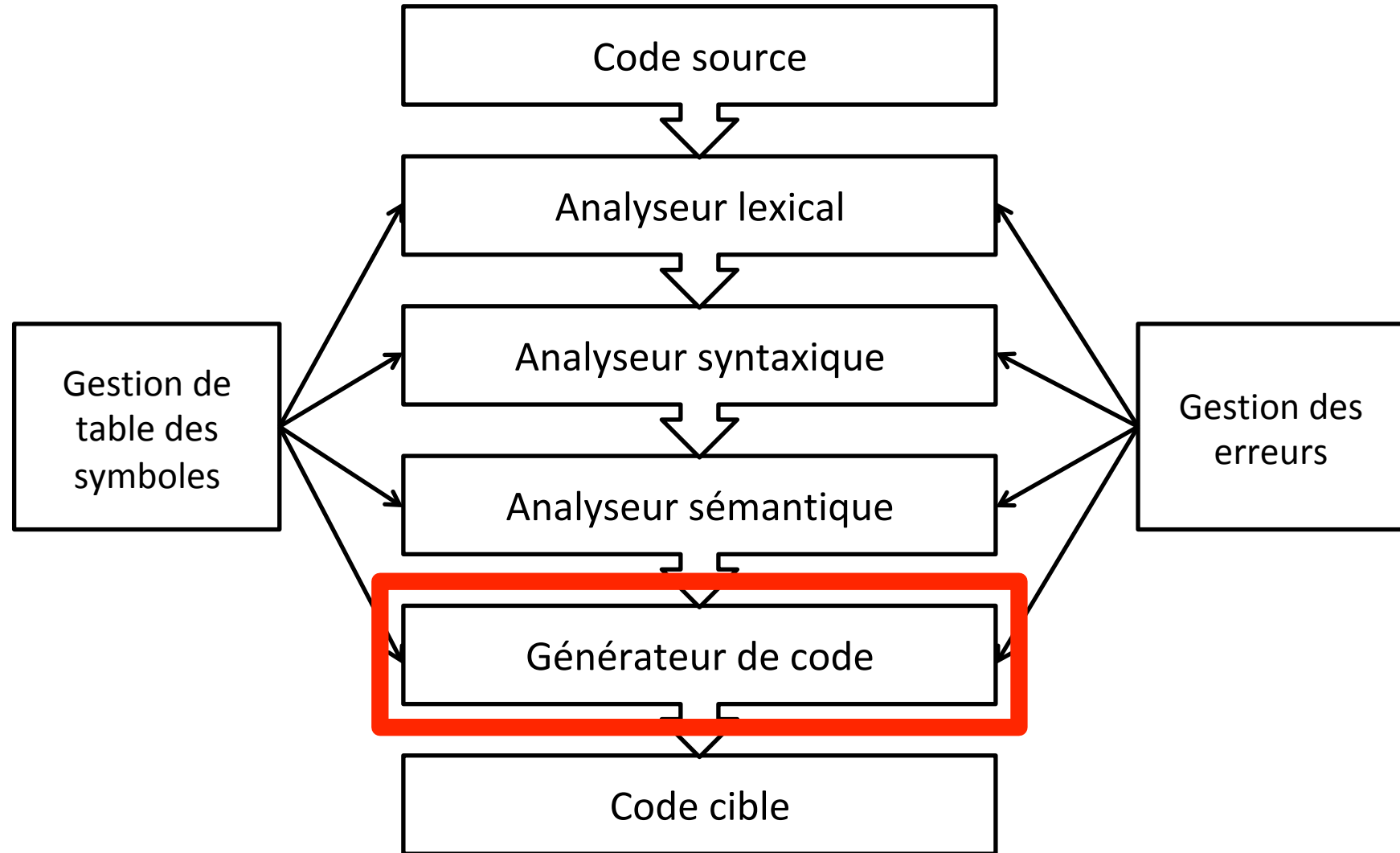
# Gestion des erreurs

---

- Multiples raisons pouvant provoquer l'arrêt du compilateur
- Vivement conseillé d'avoir une interface centralisée de reporting (voir le langage DEMO pour un exemple)
- Utilisation du mécanisme d'exception Java

# Compilateur

---



# Génération de code

---

- But : passer de la représentation interne (valide par rapport à la spécification) au code cible
- Exemples:
  - JVM pour Java
  - CAM pour CAML
  - WAM pour Prolog
  - P-Machine pour Pascal
  - Code binaire pour de l'assembleur
  - Un langage de programmation dans le cas d'un DSL
- Next Byte Codes (NBC Lego) dans notre cas
- Peut se faire en utilisant des librairies de templating (e.g., StringTemplate)
  - Exemple de systèmes utilisant du templating : PHP, JSP, etc.

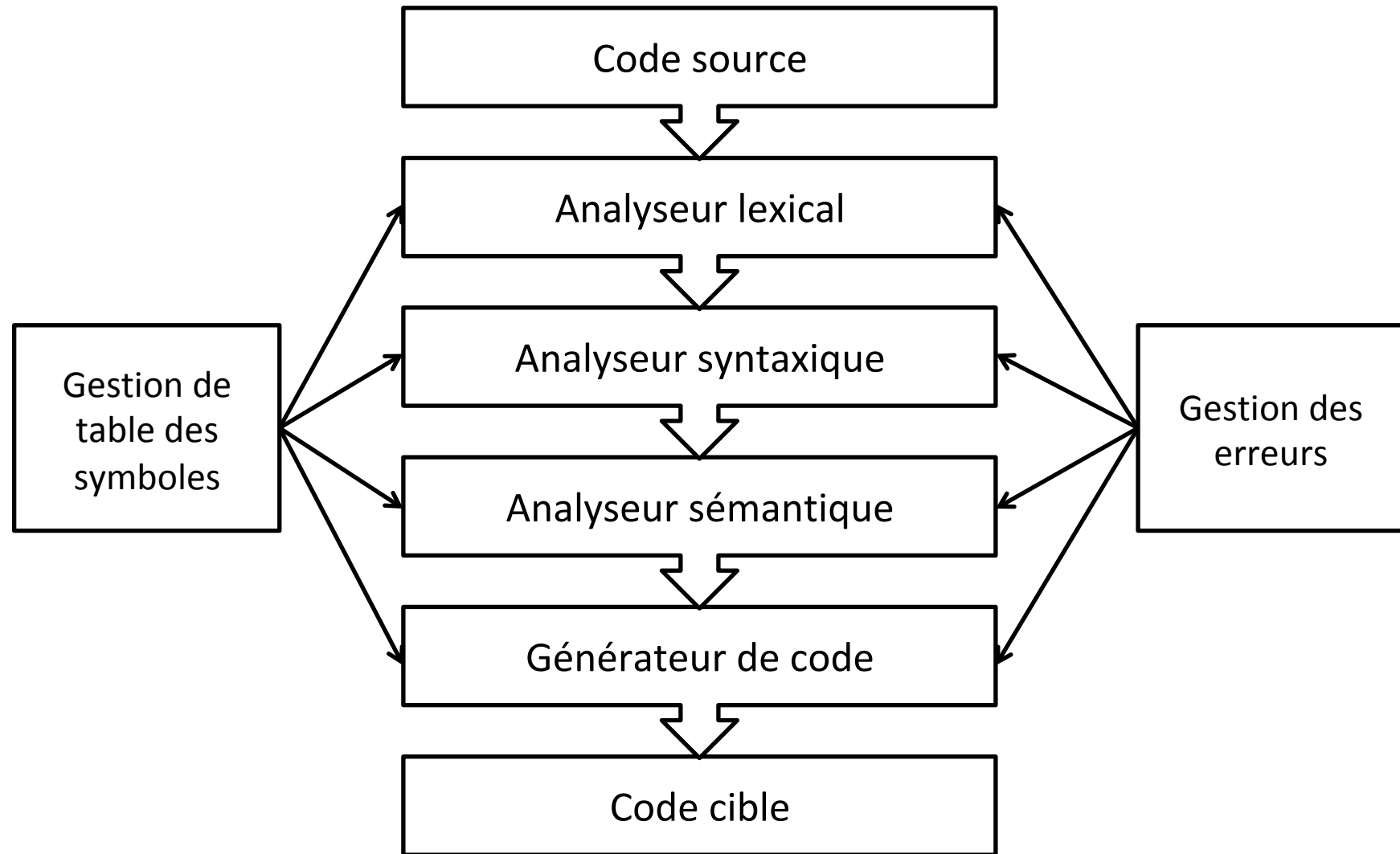
# Traduction des instructions

---

- Cf. Document “NeXT Byte Codes Manual Explanations and Examples.pdf”

# À retenir

---



# Exemple : langage DEMO

---

DEMO ::= Instruction \*

Instruction ::= **affect**( Id, Expr)  
                  | **print**( Expr)  
                  | **read**( Id)

Expr ::= Entier  
          | Id  
          | Expr **+** Expr  
          | Expr **-** Expr  
          | Expr **\*** Expr  
          | Expr **/** Expr  
          | Expr **%** Expr  
          | ( Expr)

$\{(.)\} > \{\%\} > \{*, /\} > \{-, +\}$

git clone

[https://github.com/UNamurCSFaculty/2021\\_SyntaxeSemantique\\_Students](https://github.com/UNamurCSFaculty/2021_SyntaxeSemantique_Students)

# Maven





# Maven 3: Introduction

---

- Project Lifecycle management tool
  - Project description in Project Object Model (pom.xml)
  - Build automation (including unit testing phase)
  - Project modularization
  - Dependency management
  - Deployment automation
  - Reporting (code quality check, documentation generation, website generation, test coverage, etc.)
- Convention over configuration (... like Java EE)  
*“Systems, libraries, and frameworks should assume reasonable defaults. Without requiring unnecessary configuration, systems should “just work”.”* [Maven by example]

# Installation

---

- See <https://maven.apache.org/install.html>
- Use command `mvn -v` to check installation

```
$ mvn -version
Apache Maven 3.1.1
Maven home: /usr/local/apache-maven-3.1.1
Java version: 1.8.0_31, vendor: Oracle Corporation
Java home: /Library/Java/jdk1.8.0_31.jdk/Contents/Home/jre
Default locale: fr_FR, platform encoding: UTF-8
OS name: "mac os x"
```

- Local Maven files are in `~/.m2`
- `~/.m2/settings.xml` contains access configurations
- `~/.m2/repository` contains the local Maven repository

# Maven project structure

---

- myfirstmavenproject/
  - pom.xml
  - src/
    - main/
      - java/
        - » *Here goes your Java source code*
      - resources/
        - » *Here goes other resources (files, etc.) included in the .jar/.war*
    - test/
      - java/
        - » *Here goes the JUnit tests*
      - resources/
        - » *Here goes other resources used by the tests*
    - sub-module-1/
      - pom.xml
      - src/
    - sub-module-2/
      - ...

# Pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <modelVersion>4.0.0</modelVersion>

    <groupId>be.unamur.info</groupId>
    <artifactId>myfirstmavenproject</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <name>My first Maven project</name>
    <description>This is my very first Maven project.</description>

    <packaging>jar</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.7</maven.compiler.source>
        <maven.compiler.target>1.7</maven.compiler.target>
    </properties>

</project>
```

# Pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <modelVersion>4.0.0</modelVersion>

    <groupId>be.unamur.info</groupId>
    <artifactId>myfirstmavenproject</artifactId>
    <version>0.0.1-SNAPSHOT</version>
```

## Identifier

- Maven identifiers have to be unique
- Format: *groupId:artifactId:version*
- GroupId should identify the organisation using Java package naming convention
- ArtifactId should identify the project
- Version should be encoded on 3 number XX.XX.XX
  - “-SNAPSHOT” is used to indicate the currently under development version

# Pom.xml

- Packaging indicates to Maven the type of artifact produced during build
- “jar” indicates a .jar file (executable or not)
- “pom” indicates a maven project without compiled code (used by a parent project)
- “war” indicates a .war file that will be deployed on a web server

```
<packaging>jar</packaging>
```

**Type of the artefact**

```
<properties>  
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
  <maven.compiler.source>1.7</maven.compiler.source>  
  <maven.compiler.target>1.7</maven.compiler.target>  
</properties>
```

```
</project>
```

# Pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
```

- Properties are used as global constants
  - Implicit properties
    - \${basedir} current project root directory
    - \${project.version} current project version
    - \${project.groupId} current project groupId
  - Declared properties
    - Should contain dependencies versions

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.7</maven.compiler.source>
  <maven.compiler.target>1.7</maven.compiler.target>
</properties>
```

**Properties used by project and sub-projects**

# Sub-modules

---

- Parent project (should have pom packaging)

```
<project ...>
  ...
  <packaging>pom</packaging>
  <modules>
    <module>sub-module-1</module>
  </modules>
</project>
```

- Sub-module has to declare parent project

```
<project ...>
  <parent>
    <groupId>be.unamur.info</groupId>
    <artifactId>myfirstmavenproject</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </parent>
  <artifactId>sub-module-1</artifactId>
  ...
</project>
```

- In this case: sub-module-1 version is inherited



# Dependencies

---

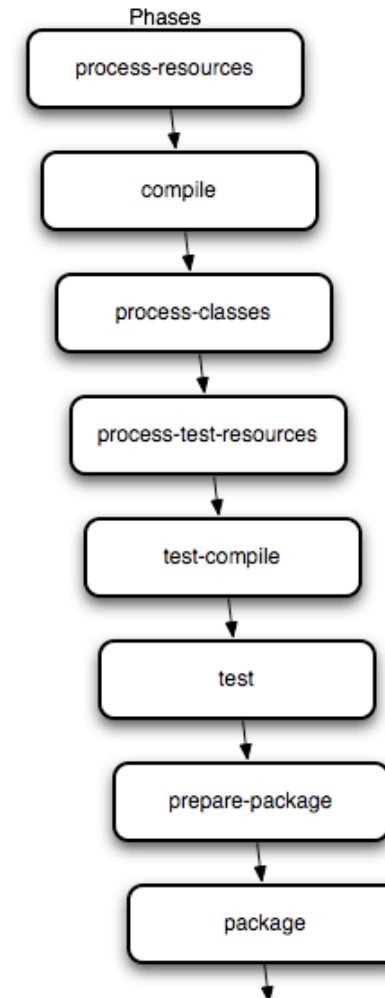
- Maven central repository
  - <http://repo.maven.apache.org/maven2>
  - <http://search.maven.org> (to search an artifact)
- Declare dependency in a Maven project

```
<project ...>
  ...
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>${junit.version}</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

# mvn <phase>

---

- **compile**
  - Compile source files
- **test**
  - Launch JUnit tests in src/test/java/
- **package**
  - Create .jar or .war or ... file in target/
- **install**
  - Install artefact in local Maven repository
- **deploy**
  - Deploy artefact using deployment configuration



Note: There are more phases than shown above, this is a partial list

## mvn <phase>

---

- **compile**
  - Compile source files
- **test**
  - Launch JUnit tests in `src/test/java/`
- **package**
  - Create `.jar` or `.war` or ... file in `target/`
- **install**
  - Install artefact in local Maven repository
- **deploy**
  - Deploy artefact using deployment configuration

### Commands:

```
$ mvn compile
```

```
$ mvn test
```

```
$ mvn package
```

```
$ mvn install
```

```
$ mvn clean
```

```
$ mvn clean package
```

**JUnit**



# JUnit: Introduction

---

- Collection of classes to perform unit testing
- Uses annotations (e.g., @Test)

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class CalculatorTest {
    @Test
    public void evaluatesExpression() {
        Calculator calculator = new Calculator();
        int sum = calculator.evaluate("1+2+3");
        assertEquals(6, sum);
    }
}
```

```
java -cp .;junit-4.12.jar;hamcrest-core-1.3.jar org.junit.runner.JUnitCore CalculatorTest
```

# Anatomy of a Junit test class

---

```
import ...
public class MyClassTest{
    private static final Logger LOG= LoggerFactory.getLogger(MyClassTest.class);

    @Rule
    public TestRule watcher = new TestWatcher() {
        @Override protected void starting(Description description) {
            LOG.info(String.format("Starting test: %s()...", description.getMethodName()));};
    };

    @BeforeClass
    public static void setUpClass() {LOG.info("Setting up before class");}

    @AfterClass
    public static void tearDownClass() {LOG.info("Tearing down after class");}

    @Before
    public void setUp() {LOG.info("Setting up before test");}

    @After
    public void tearDown() {LOG.info("Tearing down after test");}

    @Test
    public void testMyMethod() {assertTrue(true);}
}
```

# Assertions

---

- JUnit
  - `assertTrue(c); assertFalse(c);`  
`assertEquals(expected, actual); assertNull(o);`  
`assertNotNull(o); fail()`
  - `assertThat(T actual,`  
`org.hamcrest.Matcher<T> matcher)`
- Hamcrest matchers (`org.hamcrest.Matchers`)
  - `assertThat(T actual, equalTo(T operand))`
  - `assertThat(T actual, not(T operand))`
  - `assertThat(Iterable<? super T> c, hasItem(T item))`
  - `assertThat(T actual, equalTo(T operand))`
  - `assertThat(T actual, nullValue())`
  - `assertThat(T actual, notNullValue())`

# Hamcrest common matchers

---

- Core
  - anything - always matches, useful if you don't care what the object under test is
- Logical
  - allOf - matches if all matchers match, short circuits (like Java &&)
  - anyOf - matches if any matchers match, short circuits (like Java ||)
  - not - matches if the wrapped matcher doesn't match and vice versa
- Object
  - equalTo - test object equality using Object.equals
  - toString - test Object.toString
  - instanceof, isCompatibleType - test type
  - notNullValue, nullValue - test for null
  - sameInstance - test object identity
- Collections
  - hasEntry, hasKey, hasValue - test a map contains an entry, key or value
  - hasItem, hasItems - test a collection contains elements
  - hasItemInArray - test an array contains an element
- Number
  - closeTo - test floating point values are close to a given value
  - greaterThan, greaterThanOrEqualTo, lessThan, lessThanOrEqualTo - test ordering
- Text
  - equalToIgnoringCase - test string equality ignoring case
  - equalToIgnoringWhiteSpace - test string equality ignoring differences in runs of whitespace
  - containsString, endsWith, startsWith - test string matching



# References



# Support

---

- Webcampus
  - Via le forum
- Livres
  - [1], [3], [5], [6] principalement
- Vos camarades (attention au plagiat)
  - Partage des tests (et uniquement des tests !) est fortement encouragé
- Google (attention au plagiat !!!)
- L'assistant
  - Par mail à l'adresse [james.ortizvega@unamur.be](mailto:james.ortizvega@unamur.be) avec la mention [S&S] dans l'objet
  - Au bureau 432 (pensez à prévenir avant de passer)

# Outils

---

- Utilisation obligatoire de :
  - Maven
    - Ne modifiez pas le format de la ligne de commande du compilateur
    - Ne modifier par le nom (groupId + artifactId) du projet Maven
  - ANTLR
  - Git
  - JUnit
- Mise en place d'un outils d'intégration continue (+ exécution de tests pour les échéances)
- Utilisation éventuelle de :
  - StringTemplate

# Remarques générales

---

- En cas de question, doute :
  - Confronter son point de vue avec celui de son voisin (et argumenter )
  - Faire preuve de bon sens
  - Tester les différentes "alternatives"
  - Poser la question à l'encadrement
    - En oubliant pas de mentionner "[S&S]" dans l'objet du mail et en joignant les fichiers de tests
    - Consultation de vos tests sur le SVN par l'équipe d'encadrement

# Références

---

[1] *S. Chacon and B. Straub. Pro Git*, second edition. Apress, 2015.  
<https://git-scm.com/book/fr/v1>.

[2] *M. Fowler. Domain-specific languages*. Pearson Education, 2010.

[3] *T. O'Brien, J. Casey, B. Fox, J. Van Zyl, J. Xu, T. Locher, D. Fabulich, E. Redmond, and B. Snyder. Maven by Example*. Sonatype, 2015.  
<http://books.sonatype.com/mvnex-book/reference/public-book.html>.

[4] *T. O'Brien, M. Moser, J. Casey, B. Fox, J. Van Zyl, E. Redmond, and L. Shatzer. Maven : The Complete Reference*. Sonatype, 2015. <http://books.sonatype.com/mvnref-book/reference/public-book.html>.

[5] *T. Parr. Language implementation patterns : create your own domain-specific and general programming languages*. The Pragmatic Programmers, 2009.

[6] *T. Parr. The Definitive ANTLR 4 Reference*. The Pragmatic Programmers, 2013.

Sur **Webcampus** (Documents et liens/Projet\_Compilateur)