

# Cucumber?

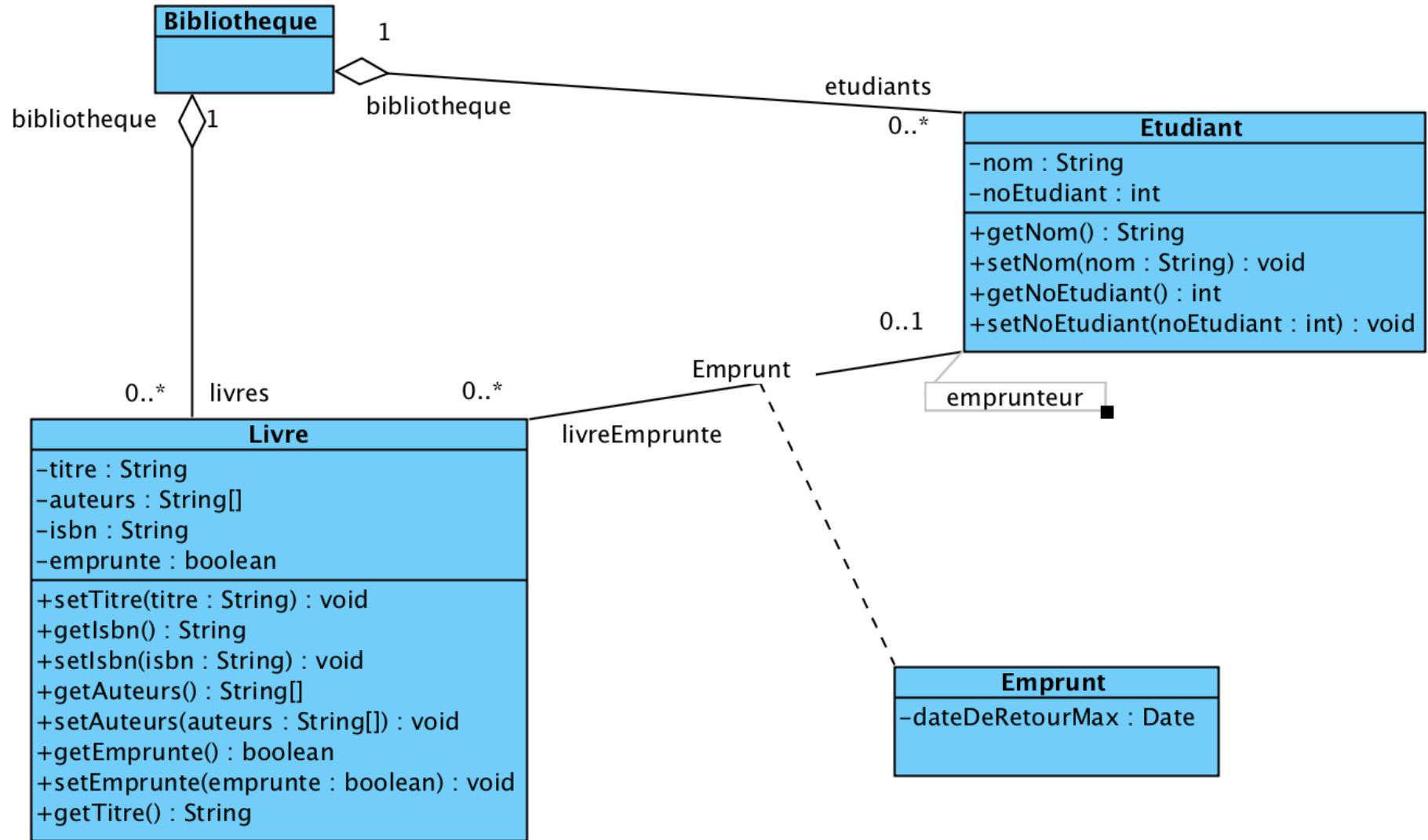




**On veut teste**  
**On veut méca**

# TDD to BDD : Behaviour Driven Development

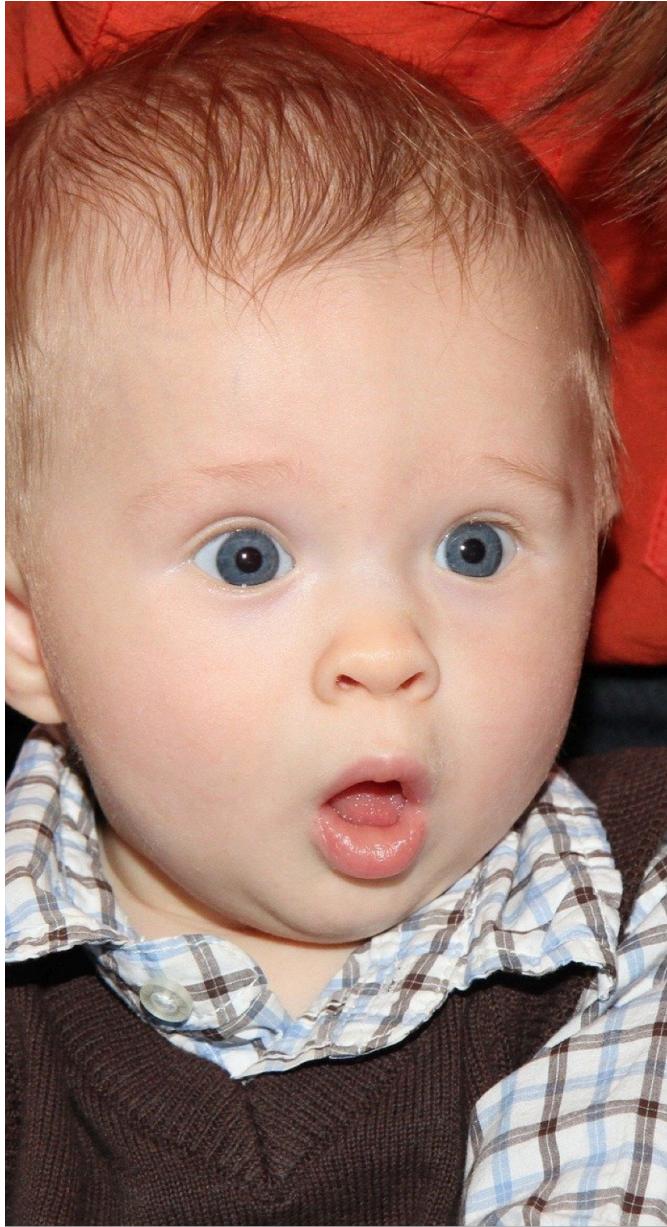
- Test Driven Development
  - Red: implémenter des tests à partir des spécifications, ils échouent
  - Green : implémenter les features et les tests passent
  - Refactor : restructuration, meilleure lisibilité
- **BDD : La spécification en langage naturel devient un test !!!**



# Langage Gherkin

- Un langage spécifique (Domain Specific Language, DSL) créé pour décrire des comportements sans définir comment les implémenter

```
Scénario: emprunt d'un livre
  Quand "Marcel" emprunte le livre "UML pour les nuls"
  Alors Il y a 1 dans son nombre d'emprunts
  Et Il y a le livre "UML pour les nuls" dans un emprunt de la liste d'emprunts
  Et Le livre "UML pour les nuls" est indisponible
```



# **Critères d'acceptation d'une User Story**

**=**  
**=**

## **Langage Gherkin**

# Langage Gherkin

- Un langage spécifique (Domain Specific Language, DSL) créé pour décrire des comportements sans définir comment les implémenter

```
Scenario: a book rental
When "Marcel" rents the book "UML pour les nuls"
Then There is 1 in his number of rentals
And The book "UML pour les nuls" is in a rental in the list of rentals
And The book "UML pour les nuls" is unavailable
```

# Structure de base Gherkin (en)

Feature:

Background:

Scenario:

Given

And

When

Then

Feature: Refund item

Scenario: Jeff returns a faulty microwave

Given Jeff has bought a microwave for \$100

And he has a receipt

When he returns the microwave

Then Jeff should be refunded \$100

<http://dontcodetired.com/blog/post/Gherkin-Cheat-Sheet>

# Exemple

**Feature:** Book rental

**Background:**

```
Given a student of name "Marcel" and with student id 123456
And a book of title "UML pour les nuls"
```

**Scenario: No rental by default**

```
When "Marcel" requests his number of rentals
Then There is 0 in his number of rentals
```

**Scenario: a book rental**

```
When "Marcel" rents the book "UML pour les nuls"
Then There is 1 in his number of rentals
And The book "UML pour les nuls" is in a rental in the list of rentals
And The book "UML pour les nuls" is unavailable
```

slide.com.br



**It's so... Beautiful**

# Et il y a un support pour plusieurs langues, dont le français...

```
#language: fr
Fonctionnalité: Emprunter un livre

Contexte:
Etant donné un etudiant de nom "Marcel" et de noEtudiant 123456
Et un livre de titre "UML pour les nuls"

Scénario: Pas d'emprunt par défaut
Quand "Marcel" demande son nombre d'emprunt
Alors Il y a 0 dans son nombre d'emprunts

Scénario: emprunt d'un livre
Quand "Marcel" emprunte le livre "UML pour les nuls"
Alors Il y a 1 dans son nombre d'emprunts
Et Il y a le livre "UML pour les nuls" dans un emprunt de la liste d'emprunts
Et Le livre "UML pour les nuls" est indisponible
```

nitin.com.br



**It's so... Beautiful**

# Cucumber <https://cucumber.io/>

- Pas vraiment un outil de test
- Plutôt un outil de collaboration, qui comprend la syntaxe Gherkin et peut exécuter les tests sur plein d'implémentation différentes
  - Ruby, Java, Groovy, JavaScript, PHP, C++, Selenium, Spring...
- Pilotage par maven + bon support dans IntelliJ quand il est à jour (génération des étapes, vérification de la couverture des phrases Gherkin)
- Deux versions en Java pour décrire les étapes de test:
  - Compatible Java 6 et plus : par annotations (cette introduction)
  - Compatible Java 8 et plus : par des lambdas

```
public class Livre {  
  
    private String titre;  
    private String[] auteurs;  
    private String isbn;  
    private boolean emprunte;  
    Bibliotheque bibliotheque;  
    Emprunt emprunt;  
  
    public Livre(Bibliotheque biblio) {  
        this.bibliotheque = biblio;  
        emprunte = false;  
    }  
}
```

```
public class Bibliotheque {  
  
    private HashMap<String,Etudiant> etudiants = new HashMap<>();  
    private HashMap<String,Livre> livres = new HashMap<>();  
  
    public Etudiant getEtudiantByName(String nom) {  
        return etudiants.get(nom);  
    }  
  
    public void addEtudiant(Etudiant e) {  
        etudiants.put(e.getNom(),e);  
    }  
  
    public Livre getLivreByTitle(String titre) {  
        return livres.get(titre);  
    }  
  
    public void addLivre(Livre l) {  
        livres.put(l.getTitre(),l);  
    }  
}
```

```
public class Etudiant {  
  
    private String nom;  
    private int noEtudiant;  
    Bibliotheque bibliotheque;  
    Collection<Emprunt> emprunt;  
  
    public Etudiant(Bibliotheque biblio) {  
        this.bibliotheque = biblio;  
        emprunt = new ArrayList<>();  
    }  
}
```

```
public class Emprunt {  
  
    private LocalDate dateDeRetourMax;  
    private Livre livreEmprunte;  
    private Etudiant emprunteur;  
  
    public Emprunt(LocalDate d, Etudiant e, Livre l) {  
        dateDeRetourMax = d;  
        emprunteur = e;  
        livreEmprunte = l;  
    }  
}
```

# Définitions d'étapes en Java

```
public class EmpruntLivreStepdefs implements Fr { // implements En si vos scénarios sont écrits en français  
    Bibliotheque biblio = new Bibliotheque();  
    Etudiant etudiant;  
    Livre livre;  
  
    public EmpruntLivreStepdefs() { // implementation des steps dans le constructeur (aussi pour les autres stepdefs)  
        Etantdonné( expression: "un etudiant de nom {string} et de noEtudiant {int}",  
            (String nomEtudiant, Integer noEtudiant) -> // besoin de refactorer int en Integer  
            {  
                Etudiant etu = new Etudiant(biblio);  
                etu.setNom(nomEtudiant);  
                etu.setNoEtudiant(noEtudiant);  
                biblio.addEtudiant(etu);  
            });  
  
        Et( expression: "un livre de titre {string}", (String titreLivre) -> {  
            Livre liv = new Livre(biblio);  
            liv.setTitre(titreLivre);  
            biblio.addLivre(liv);  
        });  
  
        Quand( expression: "{string} demande son nombre d'emprunt", (String nomEtudiant) -> {  
            etudiant = biblio.getEtudiantByName(nomEtudiant);  
        });  
  
        Alors( expression: "Il y a {int} dans son nombre d'emprunts", (Integer nbEmprunts) -> {  
            assertEquals(nbEmprunts.intValue(), etudiant.getNombreDEmprunt());  
        });  
    }  
}
```

Syntaxe pour expression régulière automatique

Analyse automatique d'arguments

Assertion Junit

```

Quand( expression: "{string} emprunte le livre {string}", (String nomEtudiant, String titreLivre) -> {
    etudiant = biblio.getEtudiantByName(nomEtudiant);
    livre = biblio.getLivreByTitle(titreLivre);
    etudiant.emprunte(livre);
});

Et( expression: "Il y a le livre {string} dans un emprunt de la liste d'emprunts", (String titreLivre) -> {
    assertTrue(etudiant.getEmprunt().stream()
        .anyMatch(emp -> emp.getLivreEmprunte().getTitre().equals(titreLivre)));
});

Et( expression: "Le livre {string} est indisponible", (String titreLivre) -> {
    assertEquals( expected: true, biblio.getLivreByTitle(titreLivre).getEmprunte());
});

```

## Code fonctionnel dans Etudiant

```

public int getNombreDEmprunt() {
    return emprunt.size();
}

public void emprunte(Livre livre) {
    Emprunt e = new Emprunt(LocalDate.now().plusDays(15), e: this, livre);
    livre.setEmprunte(true);
    emprunt.add(e);
}

```

# Même chose en anglais

```
public BookRentalStepdefs() { // implementation des steps dans le constructeur (aussi possiblement dans un autre fichier)
```

```
    Given( expression: "a student of name {string} and with student id {int}",
           (String nomEtudiant, Integer noEtudiant) -> // besoin de refactorer int en Integer
    {
        Etudiant etu = new Etudiant(biblio);
        etu.setNom(nomEtudiant);
        etu.setNoEtudiant(noEtudiant);
        biblio.addEtudiant(etu);
    });
    And( expression: "a book of title {string}", (String titreLivre) -> {
        Livre liv = new Livre(biblio);
        liv.setTitre(titreLivre);
        biblio.addLivre(liv);
    });
}

Then( expression: "There is {int} in his number of rentals", (Integer nbEmprunts) -> {
    assertEquals(nbEmprunts.intValue(), etudiant.getNombreDEmprunt());
});
```

# Encore mieux ? Scenario Outline en Gherkin

Scenario Outline: feeding a suckler cow

Given the cow weighs <weight> kg

When we calculate the feeding requirements

Then the energy should be <energy> MJ

And the protein should be <protein> kg

Examples:

| weight | energy | protein |
|--------|--------|---------|
| 450    | 26500  | 215     |
| 500    | 29500  | 245     |
| 575    | 31500  | 255     |
| 600    | 37000  | 305     |

# Setup technique

- Cucumber 6
- Cucumber-JVM n'est pas trop compatible Junit 5
  - Mais on en fait quand même en passant partiellement par le support Vintage de Junit 4 et un Junit 5.5+
- Le support Cucumber de IntelliJ est **forcément** toujours en retard par rapport aux versions...
  - Deal with it!
  - Mais en ce moment (Aout 2020), ça l'air d'aller.
- **Donc repartez bien du code de**  
<https://github.com/collet/cucumber-demo>

# Collaboration Junit / Cucumber

- Un « faux » test Junit pour configurer Cucumber

```
@RunWith(value = Cucumber.class)
@CucumberOptions(plugin = {"pretty"}, features = "src/test/resources/features")
public class RunCucumberTest { // will run all features found on the classpath
                                // in the same package as this class
}
```

