Chapitre 7 : Les Tests

- Pourquoi tester ?
- Tests unitaires et fonctionnels
- Critères de tests
- Junit

Quelques bugs tristement célèbres

- Bug de l'an 2000
  http://en.wikipedia.org/wiki/Year_2000_problem

Problème de format de date

Coût mondial estimé :
Plusieurs milliards de $

Bug de l'an 2000 : la pendule indique janvier 1900 au lieu de janvier 2000

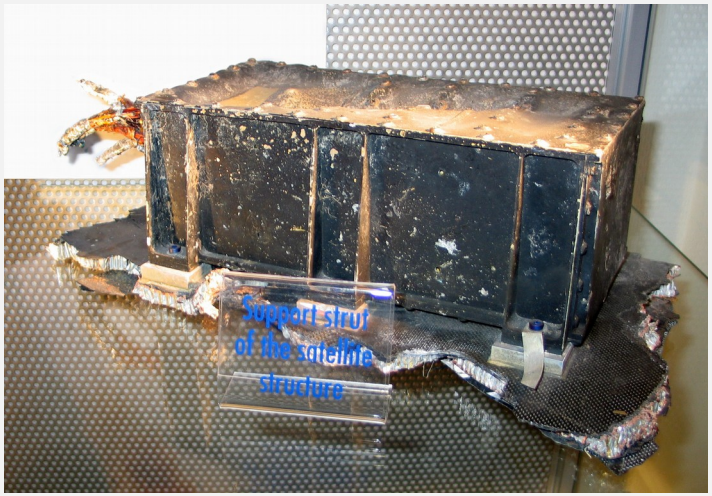
Quelques bugs tristement célèbres

- Sonde mariner 1(1962)
  http://en.wikipedia.org/wiki/Mariner_1

Problème de transcription manuelle dans la spécification du programme de guidage

Quelques bugs tristement célèbres

- Ariane 5 vol 501 (1996)
  http://fr.wikipedia.org/wiki/Vol_501_d%27Ariane_5



Dépassement mémoire

Réutilisation du système de guidage inertiel d'Ariane 4.

Coût : 370 000 000$

Assurer la qualité logicielle

- Logiciels critiques (transport, énergie, santé, militaire…)
  - Bugs inacceptables (vies, coût…)
  - Normes, certifications

- Autres logiciels
  - Niveau de qualité fixé par le client

Plus un bug est détecté tôt, moins il coute cher

Tests

- « Testing is the process of executing a program with the intent of finding errors » *Myers - The Art of Software Testing.*

Vérification et Validation (V&V)

- Verification :

  Le logiciel fonctionne t-il correctement ?

  - « Are we building the product right? »

- Validation:

  Le logiciel répond-il aux besoins ?

  - « Are we building the right product? »

Vérification et Validation (V&V)

- Méthodes de V&V:
  - Tests statiques (revues de code)
  - Tests dynamiques (exécution de code)
  - model checking, preuves formelles…

Test dynamique

- « Program testing can be used to prove the presence of bugs, but never their absence. » *Dijkstra.*

Différents types de tests

- unitaires (chaque méthode de chaque module)
- intégration (composition de méthodes ou de modules)
- système (conforme aux spécifications, performances, sur site)
- utilisateurs (conforme aux besoins)
- non régression (les évolutions du code n'ajoutent pas de bugs)

Critères de sélection

- structurel (tests boite blanche)
  - tests conçus à partir de la structure du code
  - différents critères de couverture du code

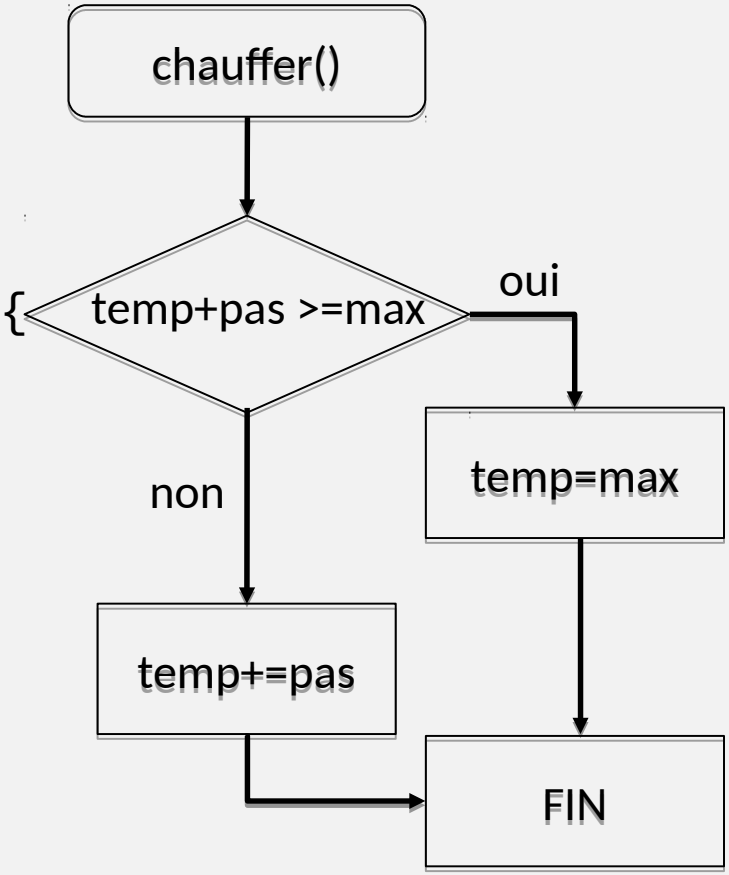
- fonctionnel (tests boite noire)
  - on utilise uniquement l'exécutable
  - tests conçus à partir de la spécification

Ces deux critères de test sont complémentaires

Tests boite blanche

- Graphe de flot de contrôle (programme représenté par un graphe)

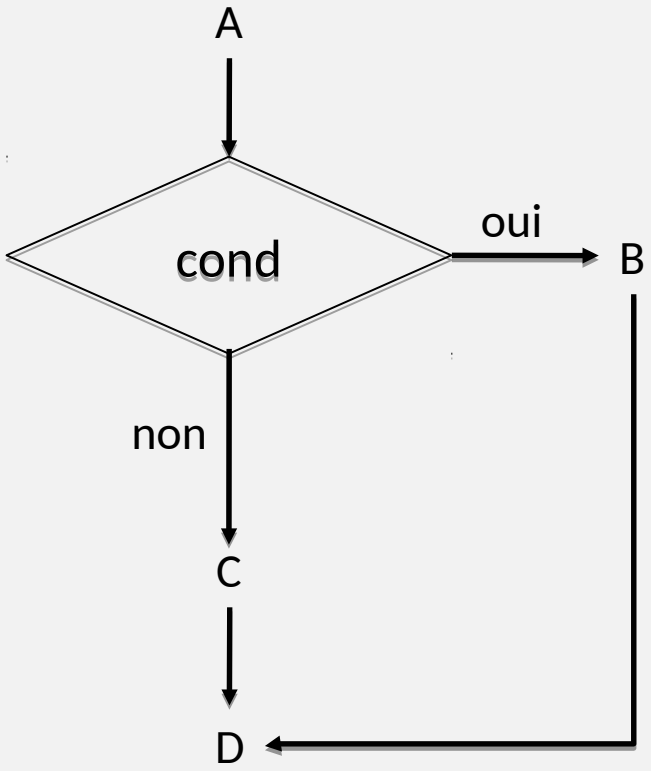
```
public void chauffer() {
    if(this.temp+this.pas>=this.max){
        this.temp = this.max;
    }
    else{
        this.temp += pas;
    }
}
```



Tests boite blanche

- Structure if then else

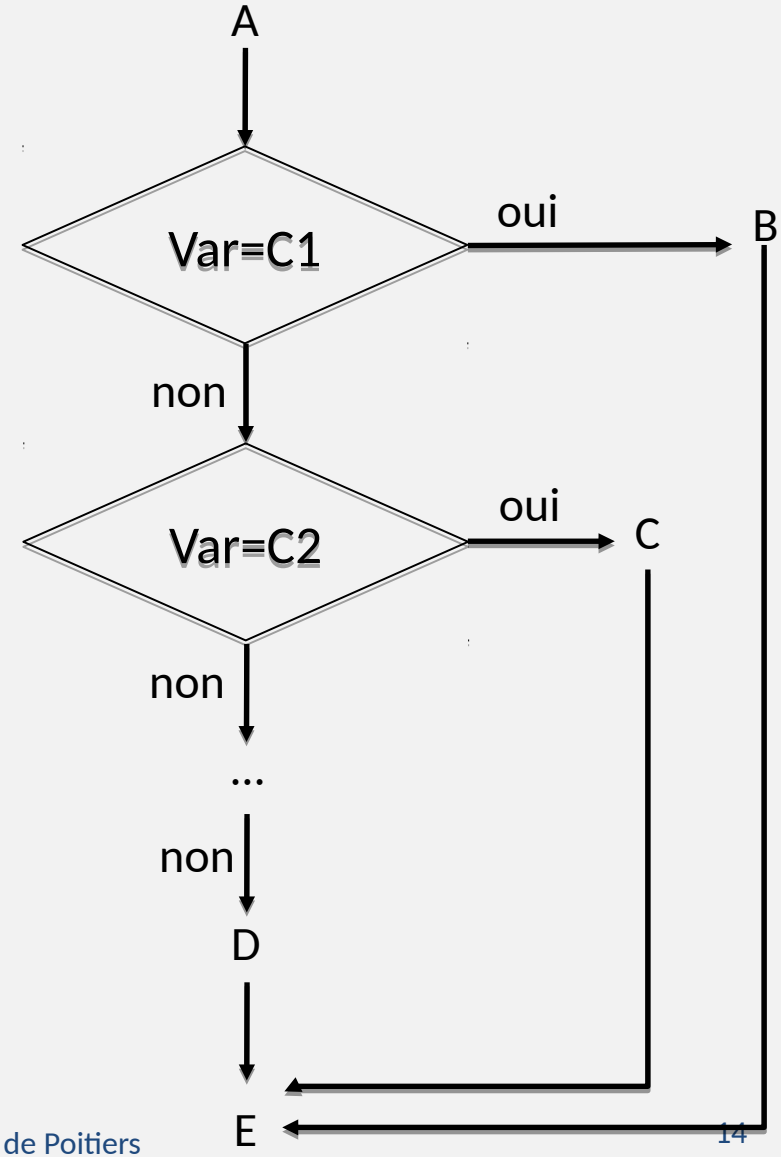
```
A
if(cond) {
    B
}
else{
    C
}
D
```



Tests boite blanche

- Structure switch case

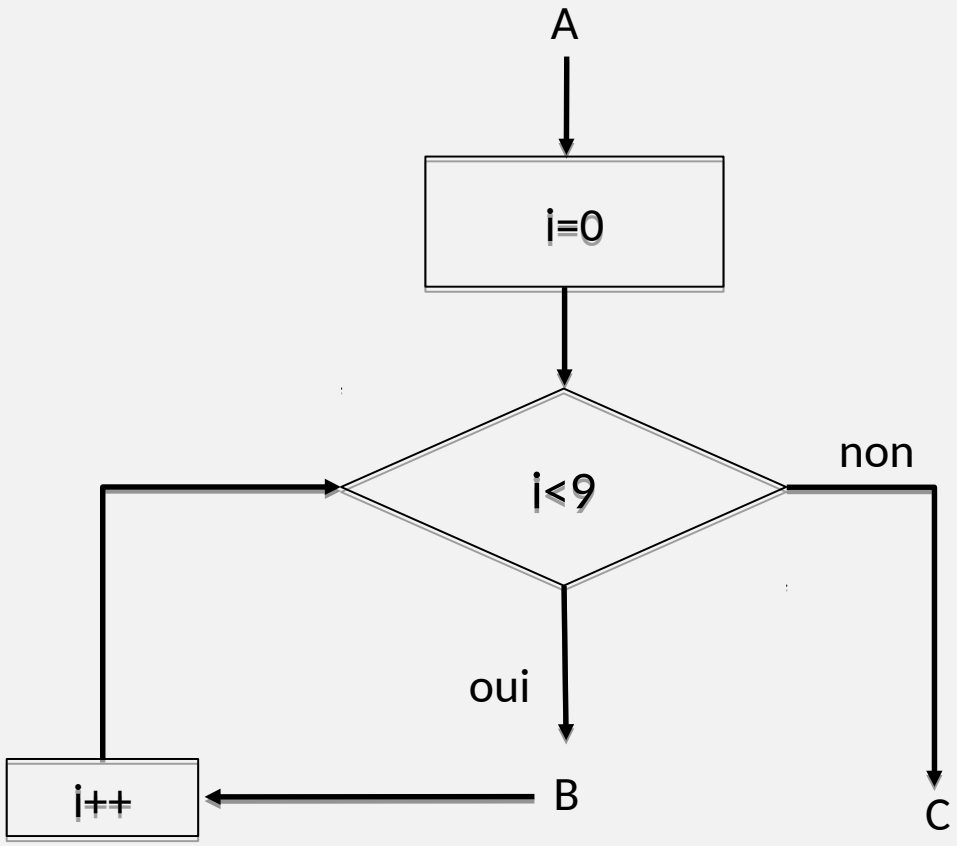
```
A
switch(var) {
    case C1: B;
    case C2: C;
    …
    default: D;
}
E
```



Tests boite blanche

- Structure boucle for

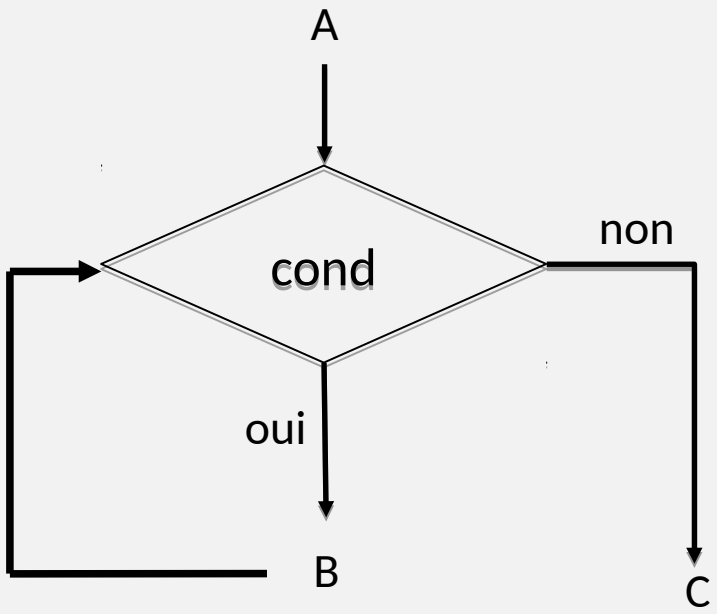
```
A
for(int i=0;i<9;i++){
    B
}
C
```



Tests boite blanche

- Structure boucle while

```
A
while(cond){
    B
}
C
```



Activation d'un chemin

- Prédicat de chemin

But : caractériser les tests qui caractérisent le chemin

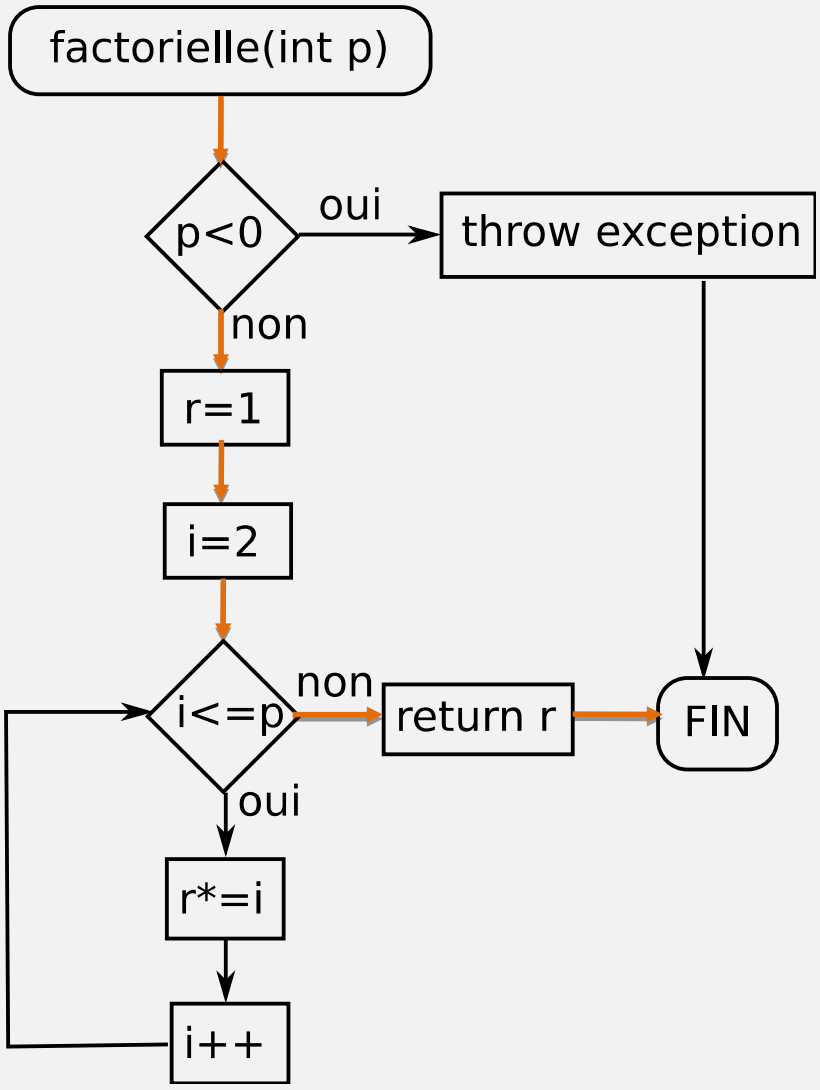
Principe :

condition → prédicat

instruction → substitution

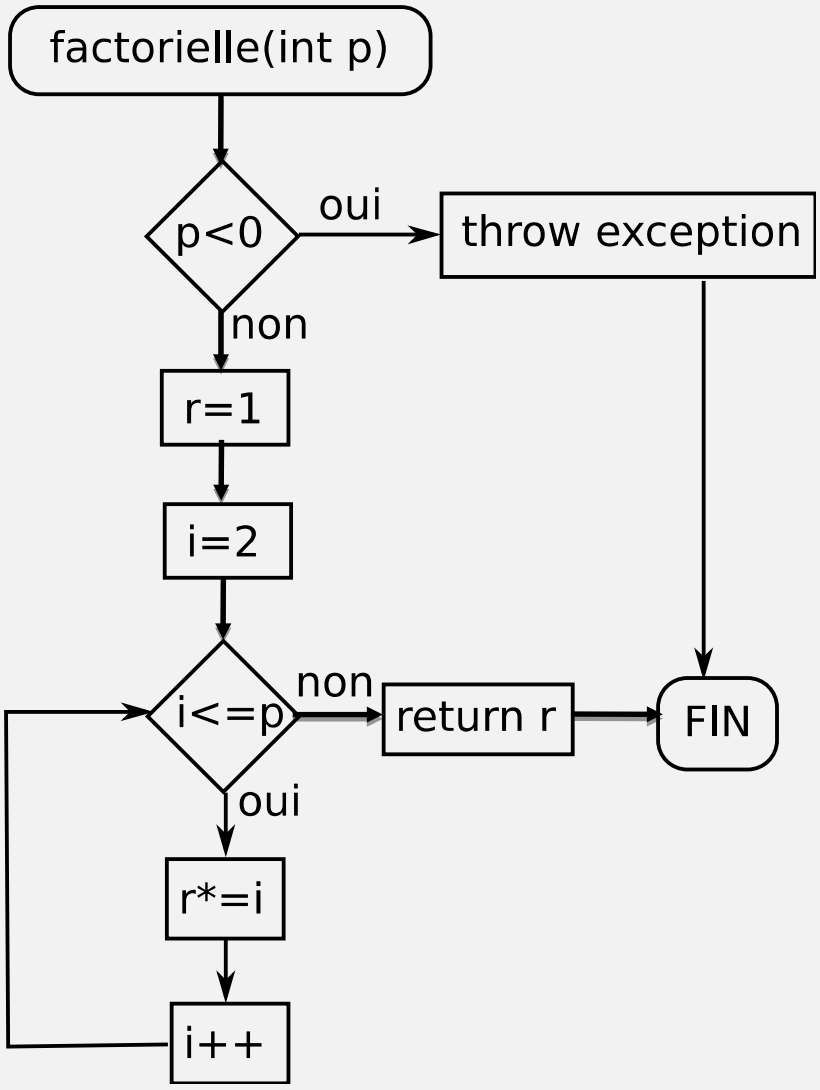
!p<0 && !i<=p [r=1,i=2]

soit p>=0 && p<2



Critères de sélection

- Tous les chemins
  - levée de l'exception
  - ne passe pas dans la boucle
  - passe 1 fois dans la boucle
  - passe 2 fois dans la boucle
  - …



Problème : nombre infini de chemins (et donc de tests)

Critère inutilisable en général

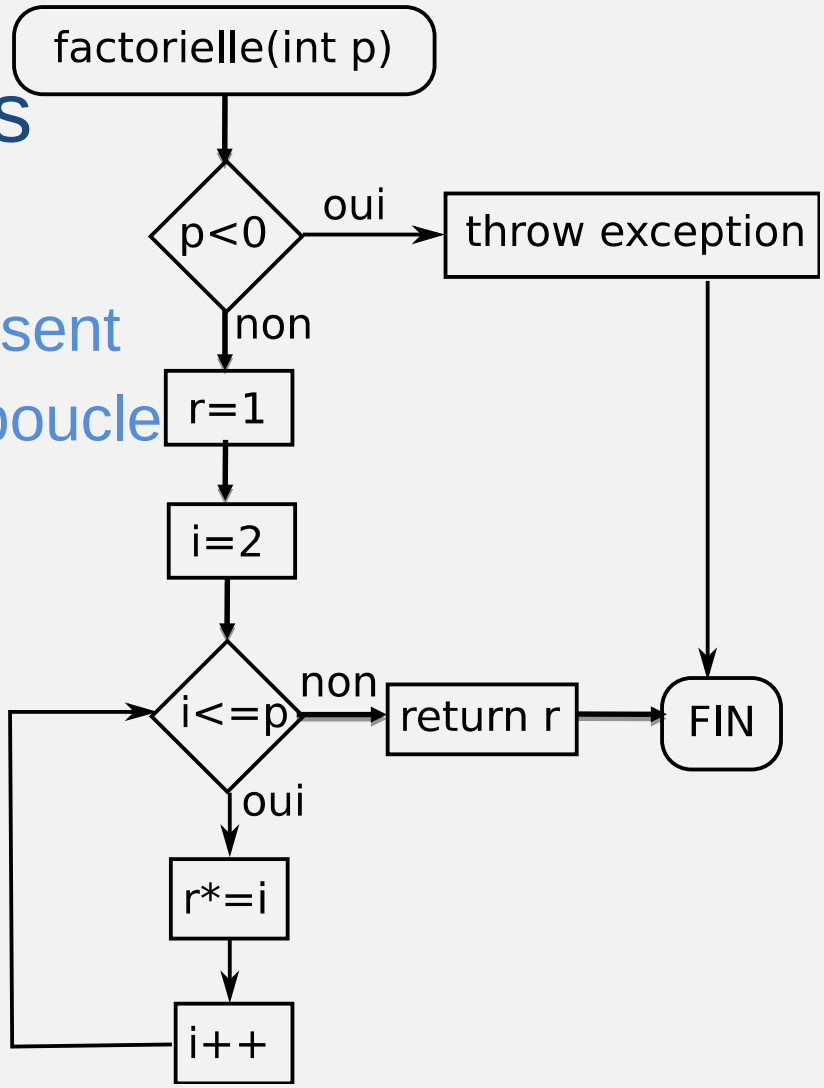
Critères de sélection

- Tous les chemins simples

But : se limiter aux chemins qui passent
1 fois au plus dans chaque boucle

trois chemins simples :

- levée de l'exception
- ne passe pas dans la boucle
- passe 1 fois dans la boucle



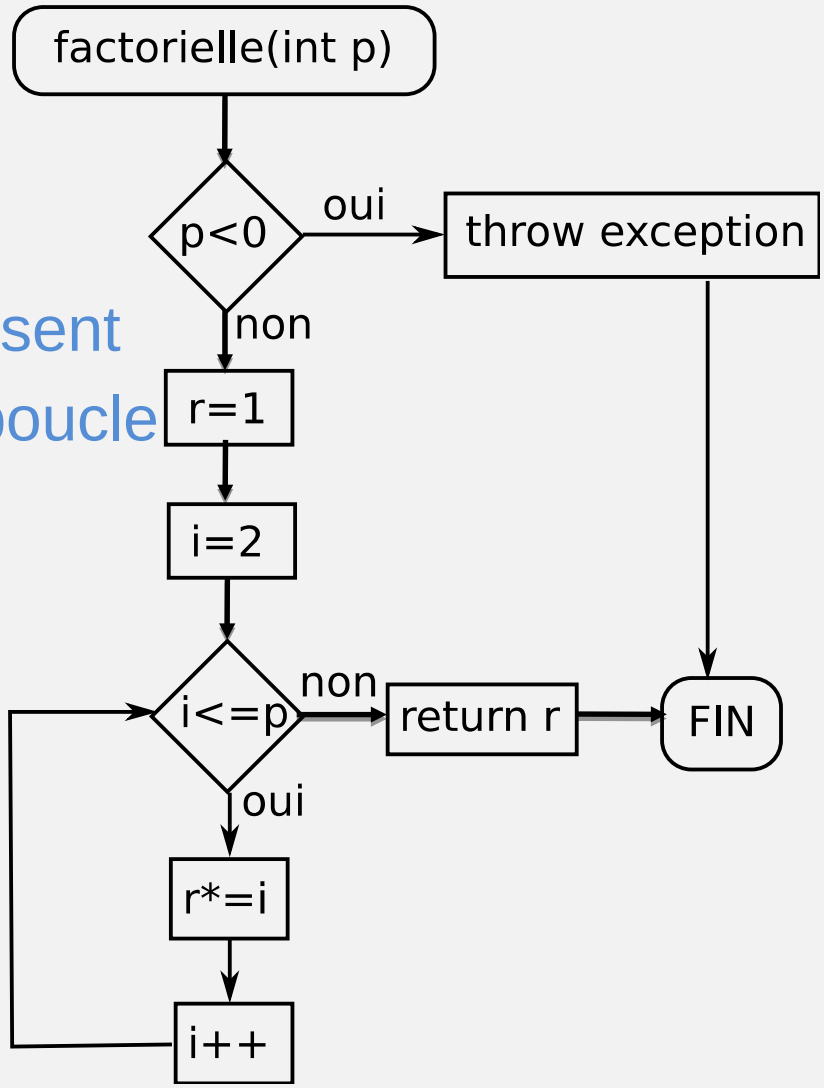
Critères de sélection

- Tous les k-chemins

But : se limiter aux chemins qui passent
k fois au plus dans chaque boucle

cinq 3-chemins :

- levée de l'exception
- ne passe pas dans la boucle
- passe 1 fois dans la boucle
- passe 2 fois dans la boucle
- passe 3 fois dans la boucle



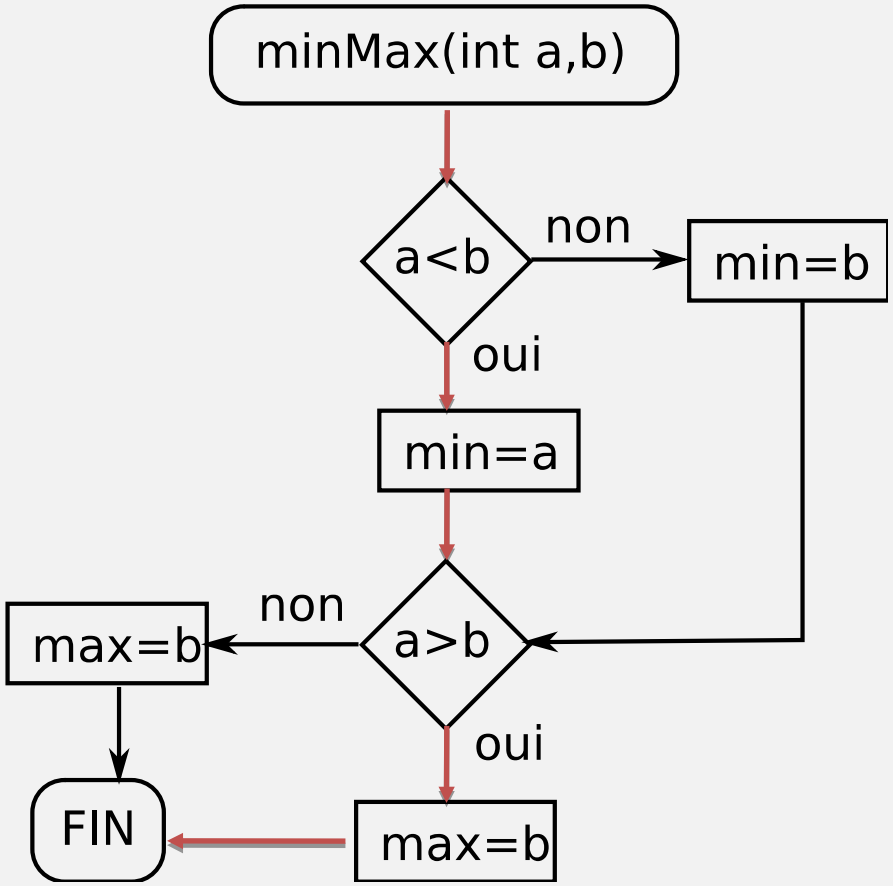
Critères de sélection

- Chemins infaisables

Prédicat de chemin

a<b && a>b

Aucune entrée de test n'active ce chemin !

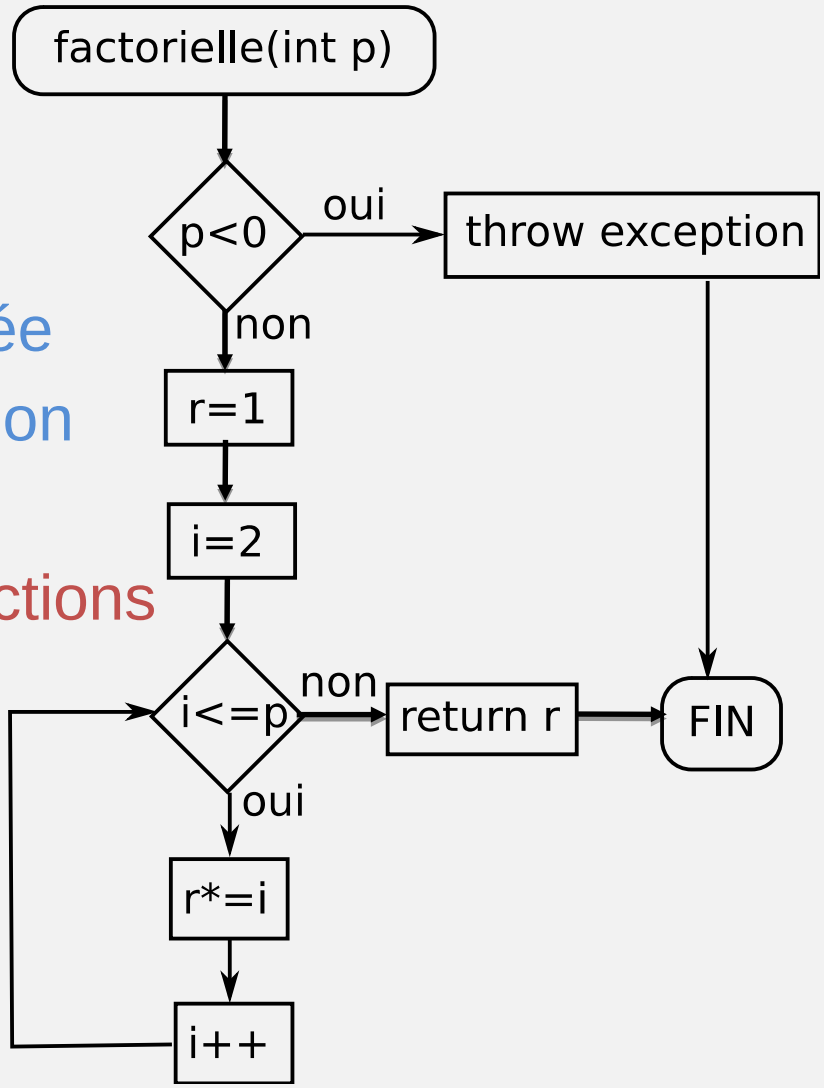


Critères de sélection

- Toutes les instructions

Définition : une instruction est activée lorsque le chemin contient l'instruction

2 chemins activent toutes les instructions
  - levée de l'exception
  - passe 1 fois dans la boucle



Critères de sélection

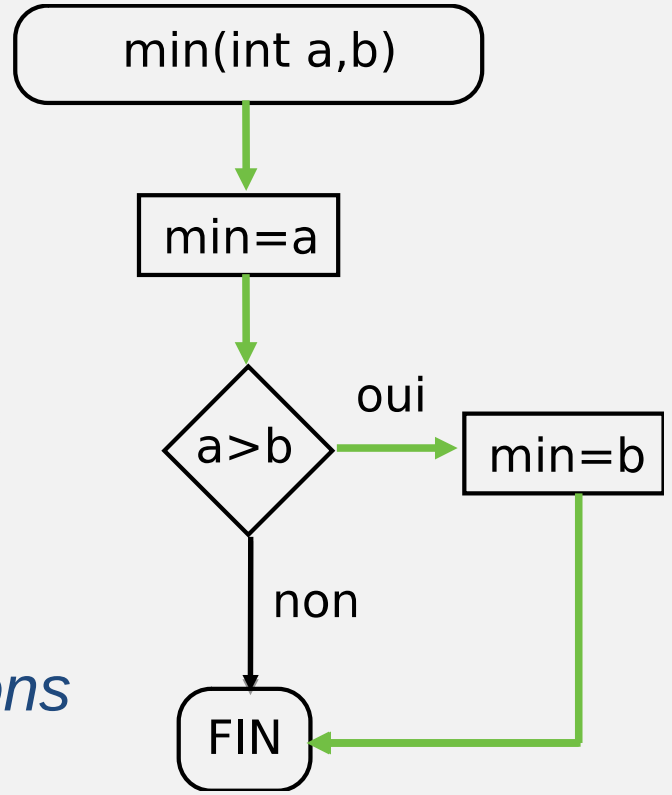
- Toutes les décisions

Définition : couverture des arcs

chemin qui active toutes les instructions

mais pas toutes les décisions !

Critère plus fort que *toutes les instructions*

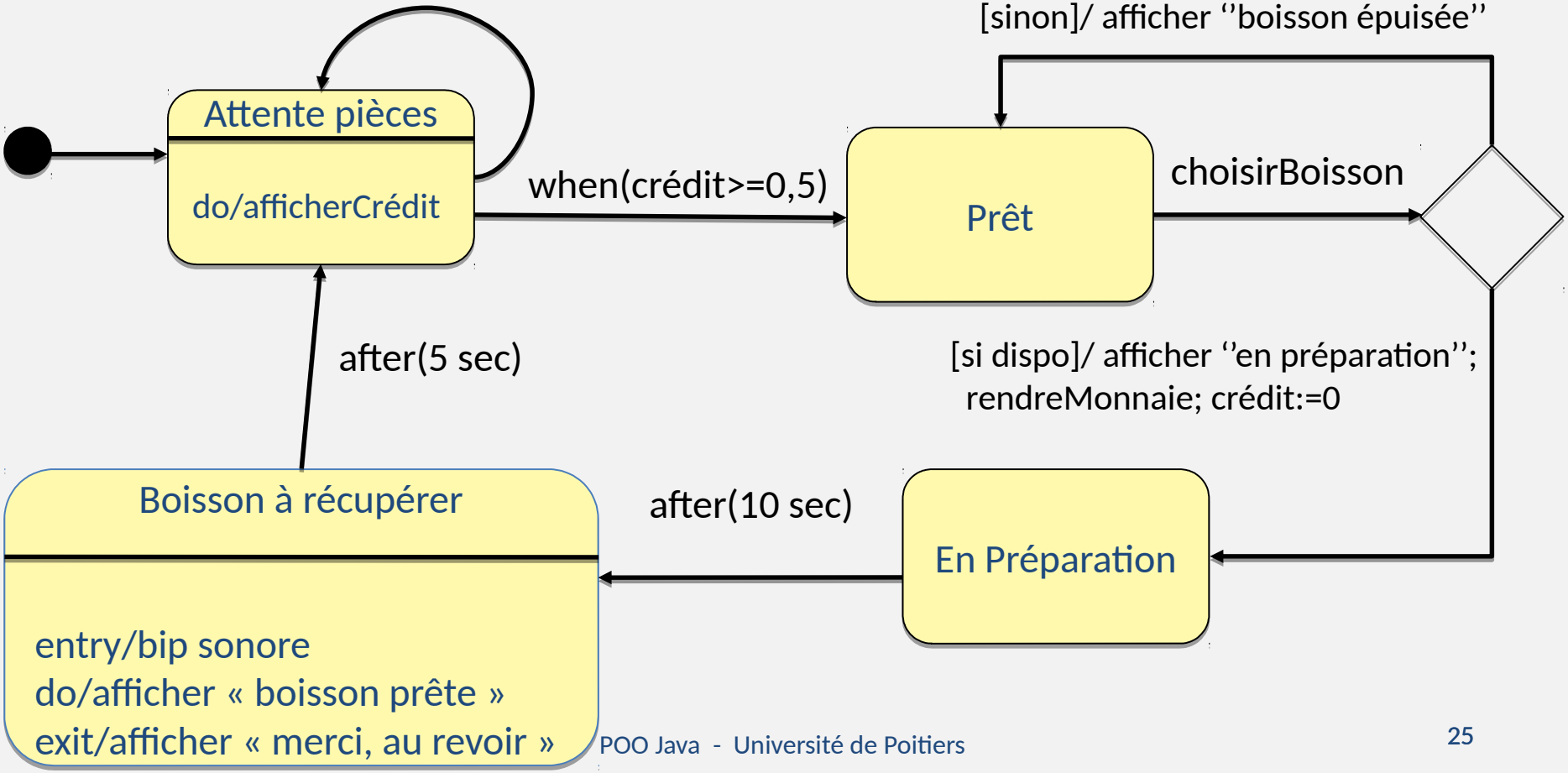


Tests boite noire

- But : couvrir la spécification
  - tests fonctionnels
- Problème : une spécification n'est pas un objet formel
  - pas de critère précis
- Solution : UML propose un cadre clair
  - utilisation des diagrammes

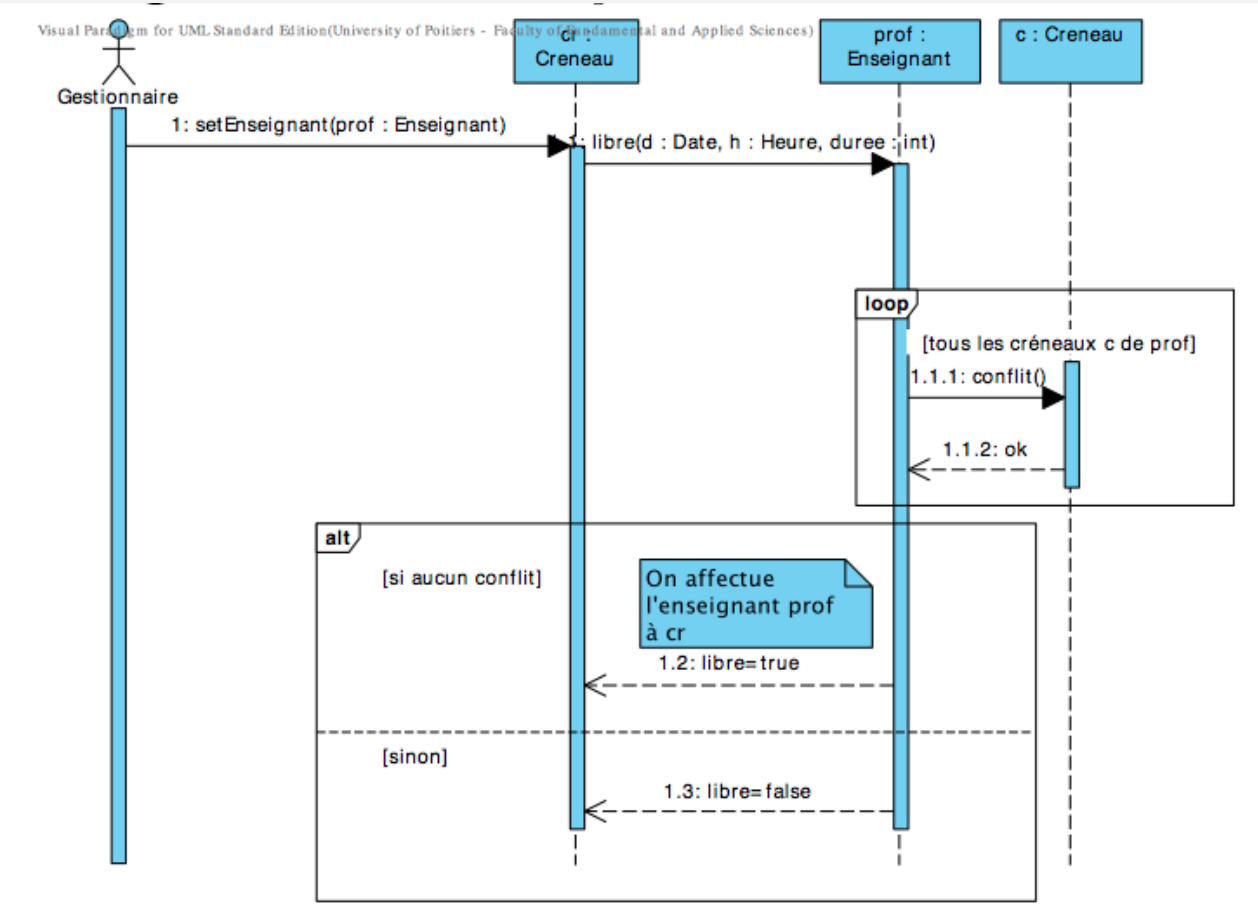
Tests boite noire

- Couverture d'un diagramme d'états
  - Couverture de graphes (états, transitions, chemins…)
  - Couverture des sous diagrammes puis du diagramme



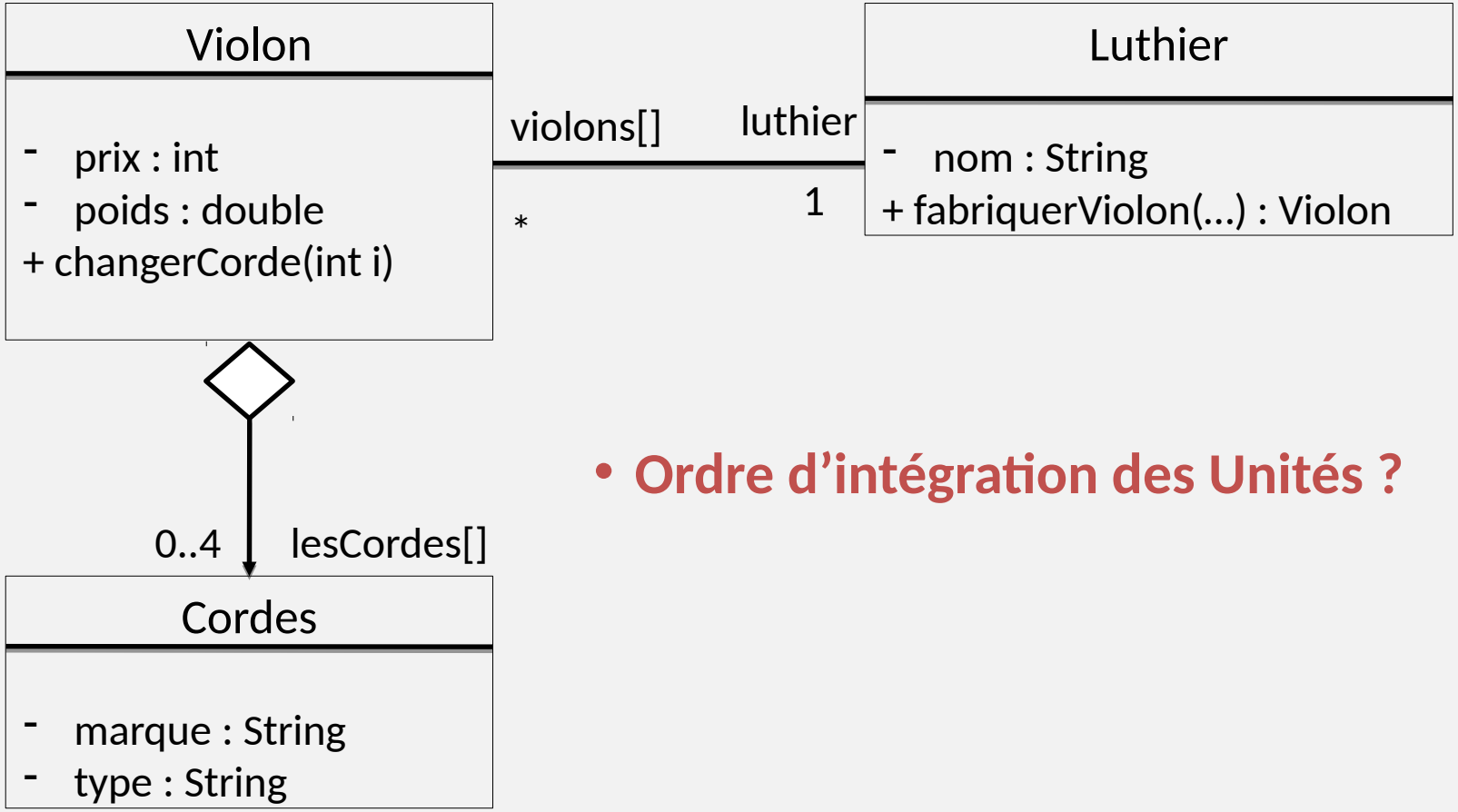
Tests boite noire

- Couverture d'un diagramme de séquences



Tests boite noire

- Couverture d'un diagramme de classes



- **Ordre d'intégration des Unités ?**

Tests unitaires

- Chaque classe est testée unitairement dans une unité de test (avec bouchons si besoin)
  - Critères structurels
  - Critères fonctionnels

- Les bouchons de test : remplacent les unités appelées par les résultats attendus

Tests d'intégration

- Les unités (testées unitairement) sont ensuite assemblées et testées ensemble.

- **Approche Bottom-up :**

  - Tests des unités les plus basses (sans bouchon)
  - Tests des unités plus hautes utilisant les plus basses

  - Avantage : pas de bouchon
  - Inconvénient : localisation des erreurs (bug non détecté lors du test unitaire)

Junit

- Automatisation
  - des tests
  - du dépouillement

cf. TP3