Génération contrôlée — mini-TP : démasquer trois sorties d'assistant

Emmanuel BRUNO
Université de Toulon, LIS UMR CNRS 7020
emmanuel.bruno@univ-tln.fr

2026-10-31

Table des matières

1 Consigne 1
2 Sortie S2 — ça ne compile pas 2
3 Sortie S1 — ça compile, mais 3
4 Sortie S3 — les tests ne voient pas tout 6
4.1 La correction 9
5 Tableau de verdicts 11
6 Prolongement (optionnel) 11

Dans la leçon 80, vous donniez les prompts. Ici vous changez de côté : vous êtes le **reviewer**. Trois sorties d'un assistant de code, produites sur le domaine `Reservation` d'un projet de réservation de tables, vous sont soumises — épinglées verbatim dans ce fichier, sans fournisseur d'IA imposé. Votre outil n'est pas l'œil : ce sont les **tests**. TP **formatif** : un livrable local, un tableau de verdicts, pas de barème.

1 Consigne

- Dépôt de travail : `/tmp/ai-tp2` ; chaque section y crée son sous-dossier (`lib/`, `s2/`, `s1/`, `s3/`) et compile dans un `out/` local. Les cellules sont auto-contenues (`mkdir -p` + `cd` absolus en tête), relançables dans l'ordre.
- Sorties étudiées dans l'ordre **S2 → S1 → S3** : l'ordre croissant du coût de leur démasquage, pas l'ordre d'arrivée des réponses de l'assistant.
- Livrable : le **tableau de verdicts** (§ final) complété et la checklist cochée.

La mécanique du jar de tests est celle de la leçon 80 ; on la répète pour l'autonomie. La dernière ligne est le contrôle d'intégrité : elle doit afficher `junit-platform-console-standalone-1.11.4.jar: OK` ; sinon, supprimez le fichier de `/tmp/ai-tp2/lib` et relancez la cellule.

```
mkdir -p /tmp/ai-tp2/lib && cd /tmp/ai-tp2/lib
JAR=junit-platform-console-standalone-1.11.4.jar
URL=https://repo1.maven.org/maven2/org/junit/platform/junit-platform-console-standalone/1.11.4/$JAR
[ -f "$JAR" ] || { for attempt in 1 2 3; do curl -sSLO "$URL" && break; sleep 2; done; }
echo "b016ef6b1c3454d6d7c2c88ce081dabf289699686af6622d6e4e2e1b54b4a2fc  $JAR" | sha256sum -c -
```

```
junit-platform-console-standalone-1.11.4.jar: OK
```

2 Sortie S2 — ça ne compile pas

Première porte à franchir : la compilation. L'assistant a produit une classe `Reservation` bâtie sur une « fabrique statique » :

```
mkdir -p /tmp/ai-tp2/s2/out && cd /tmp/ai-tp2/s2
cat > Reservation.java <<'EOF'
// Reservation creee par une fabrique statique.
public class Reservation {

    private final int id;
    private final String nom;

    private Reservation(int id, String nom) {
        this.id = id;
        this.nom = nom;
    }

    // Fabrique utilisee par le reste du projet.
    public static Reservation nouvelleReservation(int id, String nom) {
        return Reservation.of(id, nom);
    }

    public int getId() {
        return id;
    }

    public String getNom() {
        return nom;
    }
}
```

```
EOF
javac -d out Reservation.java
```

```
Picked up JAVA_TOOL_OPTIONS: -XX:MaxRAMPercentage=25
Reservation.java:14: error: cannot find symbol
        return Reservation.of(id, nom);
                          ^
  symbol:   method of(int,String)
  location: class Reservation
1 error
```

```
: 1
```

Ici, le rouge est le **résultat attendu** — la cellule est marquée `error: true`.

Consigne de réponse. Par écrit : nommez la méthode que l'assistant a **inventée** (le symbole que `javac` ne trouve pas) et la ligne fautive — la sortie de la cellule porte la réponse. Ajoutez une ou deux phrases : pourquoi la compilation est-elle la vérification la moins chère du TP ? Elle tranche S2 sans un seul test écrit.

3 Sortie S1 — ça compile, mais

Seconde sortie : une classe qui installe des couverts et une suite de tests présentée comme complète par l'assistant. Lancez la suite **telle quelle** :

```
mkdir -p /tmp/ai-tp2/s1/out && cd /tmp/ai-tp2/s1
cat > Reservation.java <<'EOF'
// Reservation d'une table pour un restaurant.
public class Reservation {

    private final int id;
    private final String nom;
    private final int places;
    private int couverts;

    public Reservation(int id, String nom, int places) {
        this.id = id;
        this.nom = nom;
        this.places = places;
    }

    // Installe des personnes si la table peut les accueillir.
    public void installer(int personnes) {
```

```
        if (couverts + personnes > places + 1) {
            return;
        }
        couverts += personnes;
    }

    public int getId() {
        return id;
    }

    public String getNom() {
        return nom;
    }

    public int getCouverts() {
        return couverts;
    }
}
EOF
cat > ReservationTest.java <<'EOF'
import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

class ReservationTest {

    @Test
    void installeJusquaCapacite() {
        Reservation r = new Reservation(1, "Dupont", 4);
        r.installer(4);
        assertEquals(4, r.getCouverts());
    }

    @Test
    void refuseAuDelaDeLaCapacite() {
        Reservation r = new Reservation(2, "Martin", 2);
        r.installer(2);
        r.installer(1);
        // TODO etudiant : completer ici pour verifier qu'une table de 2 places
        // n'a pas pu accueillir un 3e couvert.
          // Indice : la methode getCouverts() renvoie le nombre de personnes
installees.
    }
}
```

```
EOF
javac -cp /tmp/ai-tp2/lib/junit-platform-console-standalone-1.11.4.jar -d out
*.java
java -jar /tmp/ai-tp2/lib/junit-platform-console-standalone-1.11.4.jar execute
--disable-ansi-colors --disable-banner --fail-if-no-tests --class-path out --
scan-class-path out
```

```
Picked up JAVA_TOOL_OPTIONS: -XX:MaxRAMPercentage=25
Picked up JAVA_TOOL_OPTIONS: -XX:MaxRAMPercentage=25
╷
├─ JUnit Platform Suite ✓
├─ JUnit Jupiter ✓
│  └─ ReservationTest ✓
│     ├─ installeJusquaCapacite() ✓
│     └─ refuseAuDelaDeLaCapacite() ✓
└─ JUnit Vintage ✓

Test run finished after 120 ms
[         4 containers found      ]
[         0 containers skipped    ]
[         4 containers started    ]
[         0 containers aborted    ]
[         4 containers successful ]
[         0 containers failed     ]
[         2 tests found           ]
[         0 tests skipped         ]
[         2 tests started         ]
[         0 tests aborted         ]
[         2 tests successful      ]
[         0 tests failed          ]
```

La suite est verte ; elle ne prouve rien. 2 tests, 2 réussites, code de sortie 0. Or le second test, après `installer(2)` puis `installer(1)` sur une table de 2 places, ne contient **aucune assertion** — seulement un commentaire `// TODO etudiant`. Un test sans assertion ne peut pas échouer : ce vert n'est pas un résultat, c'est une question absente.

À vous de jouer. Complétez `refuseAuDelaDeLaCapacite` dans `/tmp/ai-tp2/s1/ReservationTest.java` : après les deux appels, vérifiez avec `assertEquals` le nombre de couverts réellement installés (`getCouverts()`), puis relancez la compilation et le lanceur dans `/tmp/ai-tp2/s1`. Attention : ne relancez pas la cellule ci-dessus — ses heredocs réécriraient le squelette et écraseraient votre complétion ; relancez seule-

ment `javac -cp /tmp/ai-tp2/lib/junit-platform-console-standalone-1.11.4.jar -d out *.java` et la commande du lanceur.

Après votre complétion, attendez-vous à ce rouge — extrait épinglé du labo (transcription ; le rouge est votre travail, il n'est pas exécuté à votre place) :

```
|     └─ refuseAuDelaDeLaCapacite() ✘ une table de 2 ne doit pas accueillir 3
couverts ⟹ expected: <2> but was: <3>
...
[         1 tests successful      ]
[         1 tests failed          ]
```

Question. Votre réponse doit localiser dans `installer` la ligne fautive qui vaut au `3` d'apparaître, et expliquer pourquoi le squelette vert ne pouvait pas la révéler, alors que le test complété la révèle.

4 Sortie S3 — les tests ne voient pas tout

Troisième sortie : un `Reservation` dont le commentaire d'en-tête annonce un contrat — **« le nom du client ne doit jamais être null »** — et deux tests, tous les deux verts :

```
mkdir -p /tmp/ai-tp2/s3/out && cd /tmp/ai-tp2/s3
cat > Reservation.java <<'EOF'
// Reservation : le nom du client ne doit jamais etre null (contrat annonce).
public class Reservation {

    private final int id;
    private String nom;

    public Reservation(int id, String nom) {
        this.id = id;
        this.nom = nom;
    }

    // Change le nom du client.
    public void changerNom(String nom) {
        this.nom = nom;
    }

    public int getId() {
        return id;
    }

    public String getNom() {
        return nom;
```

```
    }

    // Libelle imprime sur le billet.
    public String afficher() {
        return "Reservation #" + id + " - " + nom;
    }
}
EOF
cat > ReservationTest.java <<'EOF'
import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

class ReservationTest {

    @Test
    void creeEtAffiche() {
        Reservation r = new Reservation(7, "Dupont");
        assertEquals("Reservation #7 - Dupont", r.afficher());
    }

    @Test
    void changeNom() {
        Reservation r = new Reservation(8, "Martin");
        r.changerNom("Moreau");
        assertEquals("Moreau", r.getNom());
    }
}
EOF
javac -cp /tmp/ai-tp2/lib/junit-platform-console-standalone-1.11.4.jar -d out
*.java
java -jar /tmp/ai-tp2/lib/junit-platform-console-standalone-1.11.4.jar execute
--disable-ansi-colors --disable-banner --fail-if-no-tests --class-path out --
scan-class-path out
```

```
Picked up JAVA_TOOL_OPTIONS: -XX:MaxRAMPercentage=25
Picked up JAVA_TOOL_OPTIONS: -XX:MaxRAMPercentage=25
╷
├─ JUnit Platform Suite ✓
├─ JUnit Jupiter ✓
│  └─ ReservationTest ✓
│     ├─ creeEtAffiche() ✓
│     └─ changeNom() ✓
```

```
└─ JUnit Vintage ✓

Test run finished after 115 ms
[         4 containers found      ]
[         0 containers skipped    ]
[         4 containers started    ]
[         0 containers aborted    ]
[         4 containers successful ]
[         0 containers failed     ]
[         2 tests found           ]
[         0 tests skipped         ]
[         2 tests started         ]
[         0 tests aborted         ]
[         2 tests successful      ]
[         0 tests failed          ]
```

2/2 verts. Pourtant, ni le constructeur ni `changerNom` ne consultent `nom` : le contrat annoncé n'existe que dans le commentaire, et les tests fournis ne l'éprouvent pas — ils ne voient que le chemin heureux.

À vous de jouer. Écrivez `/tmp/ai-tp2/s3/ContratNullTest.java` : une classe de test qui **éprouve le contrat** — rouge tant que le code ne vérifie rien. Indices : un `assertThrows` qui attend une `IllegalArgumentException` du constructeur appelé avec un nom `null`, et un message d'échec explicite dans l'assertion. Relancez `javac -cp /tmp/ai-tp2/lib/junit-platform-console-standalone-1.11.4.jar -d out *.java` puis le lanceur — dans sa forme `--scan-class-path`, il exécute tous les tests compilés dans `out`, le vôtre donc — à une condition que la communauté JUnit impose : votre classe de test doit finir par `Test` ou `Tests` (le filtre de scan par défaut est `^(Test.*|.+[.$]Test.*|.*Tests?)$`) ; c'est pourquoi la référence ci-dessous s'appelle `ContratNullRefTest`.

Après votre écriture, attendez-vous à ce rouge — extrait épinglé du labo (transcription) :

```
|  ├─ ContratNullTest ✓
|  |  └─ leConstructeurRejetteUnNomNull() ✘ l'enonce impose un nom non null ⟹
Expected java.lang.IllegalArgumentException to be thrown, but nothing was thrown.
...
[         2 tests successful      ]
[         1 tests failed          ]
```

`nothing was thrown` : un test **utile qui échoue** — l'inverse du test **inutile qui réussit** de la sortie S1. Une suite ne prouve que ce qu'elle ose tenter.

4.1 La correction

Relancez l'assistant avec ce rouge en main : voici la sortie du second prompt — le même `Reservation`, avec une garde de contrat privée appliquée aux deux chemins qui écrivent `nom`. Le test de référence ci-dessous est celui du labo, **assumé ici comme corrigé**. La cellule l'écrit sous un nom distinct, `ContratNullRefTest.java`, pour ne rien écraser dans `/tmp/ai-tp2/s3`. Avant de la lancer, déplacez votre `ContratNullTest.java` hors du dossier : `javac *.java` compilerait les deux et `--scan-class-path` exécuterait les trois classes — le compte 3/3 attendu suppose le vôtre hors du dossier. Comparez la référence au vôtre : si le vôtre était rouge contre la sortie brute et devient vert contre la correction, il est bon.

```
mkdir -p /tmp/ai-tp2/s3/out && cd /tmp/ai-tp2/s3
cat > Reservation.java <<'EOF'
// Reservation : le nom du client ne doit jamais etre null (contrat verifie).
public class Reservation {

    private final int id;
    private String nom;

    public Reservation(int id, String nom) {
        this.id = id;
        this.nom = verifie(nom);
    }

    // Change le nom du client.
    public void changerNom(String nom) {
        this.nom = verifie(nom);
    }

    // Garde de contrat : null est refuse a l'entree.
    private static String verifie(String nom) {
        if (nom == null) {
            throw new IllegalArgumentException("le nom ne doit pas etre null");
        }
        return nom;
    }

    public int getId() {
        return id;
    }

    public String getNom() {
        return nom;
```

```
    }

    // Libelle imprime sur le billet.
    public String afficher() {
        return "Reservation #" + id + " - " + nom;
    }
}
EOF
cat > ContratNullRefTest.java <<'EOF'
import static org.junit.jupiter.api.Assertions.assertThrows;

import org.junit.jupiter.api.Test;

// Test de reference : met le contrat "jamais de nom null" a l'epreuve.
class ContratNullRefTest {

    @Test
    void leConstructeurRejetteUnNomNull() {
        assertThrows(IllegalArgumentException.class,
                () -> new Reservation(9, null),
                "l'enonce impose un nom non null");
    }
}
EOF
javac -cp /tmp/ai-tp2/lib/junit-platform-console-standalone-1.11.4.jar -d out
*.java
java -jar /tmp/ai-tp2/lib/junit-platform-console-standalone-1.11.4.jar execute
--disable-ansi-colors --disable-banner --fail-if-no-tests --class-path out --
scan-class-path out
```

```
Picked up JAVA_TOOL_OPTIONS: -XX:MaxRAMPercentage=25
Picked up JAVA_TOOL_OPTIONS: -XX:MaxRAMPercentage=25
╷
├─ JUnit Platform Suite ✓
├─ JUnit Jupiter ✓
│  ├─ ContratNullRefTest ✓
│  │  └─ leConstructeurRejetteUnNomNull() ✓
│  └─ ReservationTest ✓
│     ├─ creeEtAffiche() ✓
│     └─ changeNom() ✓
└─ JUnit Vintage ✓

Test run finished after 123 ms
```

```
[         5 containers found      ]
[         0 containers skipped    ]
[         5 containers started    ]
[         0 containers aborted    ]
[         5 containers successful ]
[         0 containers failed     ]
[         3 tests found           ]
[         0 tests skipped         ]
[         3 tests started         ]
[         0 tests aborted         ]
[         3 tests successful      ]
[         0 tests failed          ]
```

3/3 verts, code de sortie 0 : le contrat est passé de la promesse au mécanisme vérifié. **Question.** Le test de référence n'éprouve que le constructeur. Votre réponse doit nommer le second chemin mutateur que `verifie(...)` protège dans la correction, et indiquer comment étendre le test pour l'éprouver aussi.

5 Tableau de verdicts

L'artefact à rendre, une ligne par sortie : `compile` (`oui` / `non` observé) ; `tests` (`N verts` / `N rouges` après votre travail) ; `verdict` (un mot : adoptable, à corriger, à rejeter…) ; `action` (ce que vous renverriez à l'assistant) ; commentaires libres.

sortie	compile	tests	verdict	action	commentaires
S2					
S1					
S3					

Checklist de rendu — tout doit être coché :

- ☐ la cellule jar affiche `...: OK` (empreinte `sha256` vérifiée ; sinon supprimer le fichier de `/tmp/ai-tp2/lib` et relancer la cellule) ;
- ☐ S2 : la méthode inventée et la ligne fautive sont nommées par écrit ;
- ☐ S1 : le test complété est rouge avec `expected: <2> but was: <3>` ;
- ☐ S3 : votre `ContratNullTest` est rouge contre la sortie brute (`nothing was thrown`), puis le run de correction passe 3/3 vert ;
- ☐ le tableau de verdicts est complété, aucune cellule vide sur les lignes S1–S3.

6 Prolongement (optionnel)

Rejouez le protocole avec l'assistant de votre choix, sur un domaine qui vous appartient : compilation d'abord, suite verte soupçonnée, contrat annoncé écrit sous forme

de test, tableau de verdicts. Les sorties de ce TP sont pré-épinglées et l'échange réel avec l'assistant n'est pas retranscrit ; voici une formulation **générique** de prompt de relance (aucun échange simulé), à coller quand votre propre test passe au rouge :

```
Ton dernier fichier ne passe pas mes tests.
1) Test qui échoue : <nom de la méthode de test + fichier>
2) Sortie brute    : <rapport du lanceur JUnit, bloc d'échec complet>
3) Contrat attendu : <rappel du contrat en une phrase>
Ne modifie ni le test ni le contrat : corrige le code pour le satisfaire,
et renvoie le fichier source complet.
```