Tester pour durer : JUnit 5 sur le code des leçons 07 et 08

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

2026-11-02

Table des matières

1 Pourquoi tester 1
2 Comment exécuter ce TP 2
3 1. Premier test : du rouge au vert 5
4 2. Assertions usuelles et `assertThrows` 7
5 3. Cycle de vie et lisibilité : `@BeforeEach`, `@DisplayName` 8
6 4. Un test, cent jeux : `@ParameterizedTest` 9
7 5. Tester le DAO de la leçon 08 sur H2 en mémoire 10
8 6. Mini-cycle TDD guidé 12
9 7. Compte rendu 13
10 Conclusion 14

1 Pourquoi tester

Les deux leçons précédentes ont posé des **contrats** : une méthode qui retourne une collection retourne une collection vide, jamais `null` (leçon 07) ; un DAO écrit contre l'interface `Connection` fonctionne inchangé sur PostgreSQL et sur H2 (leçon 08). Un contrat non vérifié est une promesse : il se dégrade silencieusement à chaque modification, et sa violation est découverte par l'utilisateur final, pas par l'auteur. C'est là l'essentiel de la **dette de test** : le code qui fonctionne aujourd'hui n'est pas garanti de fonctionner demain après refactorage.

Un **test unitaire** transforme une promesse en fait vérifiable : c'est du code qui exerce du code. La dernière question de vérification de la leçon 08 demandait qui découvre le `null` renvoyé par `chercherParNom` — la réponse de ce TP est : un test, à la milliseconde où la régression est introduite, jamais un usager des transports une soirée d'hiver.

Nous utiliserons **JUnit 5**, le standard de fait des tests en Java. Deux différences avec un projet Maven classique, liées au support de cours :

1. les tests s'exécutent **dans ce carnet**, sans terminal ni `mvn test` — un petit lanceur programmatique fait le travail (cellule de préparation, unique partie « invisible » de ce TP) ;
2. un test **rouge n'interrompt rien** : le lanceur capture les échecs et les affiche. Une sortie `Failures: 1` est un résultat pédagogique normal, pas une panne du carnet.

2 Comment exécuter ce TP

- **En local** : ouvrez ce fichier dans le conteneur du cours (noyau JShell/IJava) et exécutez les cellules **dans l'ordre**, de la préparation à la fin. Chaque point de contrôle doit afficher `Failures: 0`.
- **En CI** : une poussée sur `develop` déclenche le rendu Quarto du pipeline de cours, qui exécute toutes les cellules ; la page rendue porte les mêmes lignes `Tests run: …` que vous obtenez en local. Le rendu est la preuve de passation.

La cellule suivante résout les dépendances (JUnit 5, lanceur de plateforme, H2 — mêmes versions que la leçon 08 pour H2) et définit l'utilitaire `executerTests`. Elle est exécutée une fois, en tête ; elle fait exception à notre convention : tout le code est à vous, sauf la plomberie, et reste volontairement compacte.

```
20:12:17.858  [IJava-executor-0]  WARN    o.a.h.c.p.ResponseProcessCookies  --
Invalid cookie header: "set-cookie: __cf_bm=dT_gI7ptJeCTABtsKJPVQDuLXgxqBbixUlNIINpX.kM-1793650337.
YEJmibcBSvKmTVPHNdGiZeeBKdZLs4KCIDoOu_lk9y4BwUOQJQ8YdBbAMDtIOWY9jL62R5WLJ08UyH5yliCoIJ1Eh8Bj_TRHo8
HttpOnly;   SameSite=None;   Secure;   Path=/;   Domain=repo.maven.apache.org;
Expires=Mon, 02 Nov 2026 20:42:17 GMT". Invalid 'expires' attribute: Mon, 02 Nov
2026 20:42:17 GMT
20:12:17.960  [IJava-executor-0]  WARN    o.a.h.c.p.ResponseProcessCookies  --
Invalid cookie header: "set-cookie: __cf_bm=ct6qYWV.n0M7OPJ.9bYoSOSh9uNIG02KOmvbvL6g1Ps-1793650337.
oaMQvHNg9jyOX.rDb7WmJKcsrXZfDG0i8jiYKq82HDXFNlYLSarOKnkSRl0sqPhiNezYdjk1mtfutGCAn8v1KNJYYvGsoY_HjV
HttpOnly;   SameSite=None;   Secure;   Path=/;   Domain=repo.maven.apache.org;
Expires=Mon, 02 Nov 2026 20:42:17 GMT". Invalid 'expires' attribute: Mon, 02 Nov
2026 20:42:17 GMT
20:12:18.017  [IJava-executor-0]  WARN    o.a.h.c.p.ResponseProcessCookies  --
Invalid cookie header: "set-cookie: __cf_bm=qjae.mBpIRz8KY7QiUgQU6FJ94TwmrzjM3o5_9bLDsM-1793650337.
pcVKu66hj1tcLs0jbVNgOtKeb5lBDfFba1yC1elZimqMa8MN8gdue0Hdf8w_5jgSlMxXX9LhJJyFDXPV1_ZEuwUv8kDyqWqphF
HttpOnly;   SameSite=None;   Secure;   Path=/;   Domain=repo.maven.apache.org;
Expires=Mon, 02 Nov 2026 20:42:18 GMT". Invalid 'expires' attribute: Mon, 02 Nov
2026 20:42:18 GMT
20:12:18.071  [IJava-executor-0]  WARN    o.a.h.c.p.ResponseProcessCookies  --
Invalid cookie header: "set-cookie: __cf_bm=nPOi6fEXPUBOwZxmm9xaekfWn1w4yuLFwqTi0p40NU0-1793650338.
MfhOnPGHjsy5KppbMJZkCoyTSyj8RpdduQED0ztJuunXiQoeBjPapj9dI7NrtzP9jJf_Nsw7yBwzlIfUKZkQmi8CiS8_Ev2Pd
HttpOnly;   SameSite=None;   Secure;   Path=/;   Domain=repo.maven.apache.org;
Expires=Mon, 02 Nov 2026 20:42:18 GMT". Invalid 'expires' attribute: Mon, 02 Nov
2026 20:42:18 GMT
```

```
20:12:18.129  [IJava-executor-0]  WARN    o.a.h.c.p.ResponseProcessCookies  --
Invalid cookie header: "set-cookie: __cf_bm=KoApa.SDFLgu0V8_w.UbcXsGczVuhR.NO5t7L9qRsdk-1793650338.
SaUCwLlxoj5QI4.x1E8UiXaOdUYg3T3Jy2rAYdtr5aPsIAPZ3e2Cc3TYtRhc2816OhJ8s0.fs.wvLtacu285lc9PQuTlTAhi5
HttpOnly;   SameSite=None;   Secure;   Path=/;   Domain=repo.maven.apache.org;
Expires=Mon, 02 Nov 2026 20:42:18 GMT". Invalid 'expires' attribute: Mon, 02 Nov
2026 20:42:18 GMT
20:12:18.176  [IJava-executor-0]  WARN    o.a.h.c.p.ResponseProcessCookies  --
Invalid cookie header: "set-cookie: __cf_bm=BZAketRZmk8dpGurft_Ue1AsAaHn53KE0mhFez5I4Bc-1793650338.
HttpOnly;   SameSite=None;   Secure;   Path=/;   Domain=repo.maven.apache.org;
Expires=Mon, 02 Nov 2026 20:42:18 GMT". Invalid 'expires' attribute: Mon, 02 Nov
2026 20:42:18 GMT
20:12:18.226                                                [IJava-executor-0]
WARN  o.a.h.c.p.ResponseProcessCookies -- Invalid cookie header: "set-cookie:
__cf_bm=nx8Woj9QtqhawIKrqRlBGSKFdVXChf4urHy9MlWULyY-1793650338.200041-1.0.1.1-xWCu17k9E1KsBGtw.Io
HttpOnly;   SameSite=None;   Secure;   Path=/;   Domain=repo.maven.apache.org;
Expires=Mon, 02 Nov 2026 20:42:18 GMT". Invalid 'expires' attribute: Mon, 02 Nov
2026 20:42:18 GMT
20:12:18.281  [IJava-executor-0]  WARN    o.a.h.c.p.ResponseProcessCookies  --
Invalid cookie header: "set-cookie: __cf_bm=KSm3zwU9LlXt_z.EUrPlm4MVIrVdjtWRPn74PnI7QCk-1793650338.
X1nveMS4YrMGb.cgkDZcibuXSy1Y2RE7rUjzDDazhTJsj710q2PJB9WYs0bAvQmCTQRuVJF7QiLOm.8fQwNh4SbAAe1tOFWi0
HttpOnly;   SameSite=None;   Secure;   Path=/;   Domain=repo.maven.apache.org;
Expires=Mon, 02 Nov 2026 20:42:18 GMT". Invalid 'expires' attribute: Mon, 02 Nov
2026 20:42:18 GMT
20:12:18.332  [IJava-executor-0]  WARN    o.a.h.c.p.ResponseProcessCookies  --
Invalid cookie header: "set-cookie: __cf_bm=sBr_Gha6lWRVw3C667Zjw5RO9BwJot0D.o6bgK6Tjn0-1793650338.
HttpOnly;   SameSite=None;   Secure;   Path=/;   Domain=repo.maven.apache.org;
Expires=Mon, 02 Nov 2026 20:42:18 GMT". Invalid 'expires' attribute: Mon, 02 Nov
2026 20:42:18 GMT
20:12:18.396     [BasicRepositoryConnector-repo.maven.apache.org-0-0]     WARN
o.a.h.c.p.ResponseProcessCookies -- Invalid cookie header: "set-cookie: __cf_bm=QtYjgTKNuER2eWFcA6g
q_T5aAlQZ4NnEZE3C_V3P.Ts36AFA3EpEV1N6m1IJcxX9dVQEGu1G99VUBeWg2nX5Ro2fY2p21O.BiLJhRA3AlbJ0XhvERkiH
HttpOnly;   SameSite=None;   Secure;   Path=/;   Domain=repo.maven.apache.org;
Expires=Mon, 02 Nov 2026 20:42:18 GMT". Invalid 'expires' attribute: Mon, 02 Nov
2026 20:42:18 GMT
20:12:18.438     [BasicRepositoryConnector-repo.maven.apache.org-0-0]     WARN
o.a.h.c.p.ResponseProcessCookies -- Invalid cookie header: "set-cookie: __cf_bm=.QB8otbmZoNShzEELsw
Q7WoeNFubwpHwm5NRHLKMIo4KhI6KtiFJeNjkw3Y_AwSEeK31S0efqLAlrPLaEfqCy6ACDSbGVGS3JTDAJGMysjBIDSJEeyfK
HttpOnly;   SameSite=None;   Secure;   Path=/;   Domain=repo.maven.apache.org;
Expires=Mon, 02 Nov 2026 20:42:18 GMT". Invalid 'expires' attribute: Mon, 02 Nov
2026 20:42:18 GMT
20:12:18.557     [BasicRepositoryConnector-repo.maven.apache.org-0-0]     WARN
o.a.h.c.p.ResponseProcessCookies -- Invalid cookie header: "set-cookie: __cf_bm=hGH653_gc8FsXb3nTmB
HttpOnly;   SameSite=None;   Secure;   Path=/;   Domain=repo.maven.apache.org;
Expires=Mon, 02 Nov 2026 20:42:18 GMT". Invalid 'expires' attribute: Mon, 02 Nov
```

```
2026 20:42:18 GMT
20:12:18.622     [BasicRepositoryConnector-repo.maven.apache.org-0-0]     WARN
o.a.h.c.p.ResponseProcessCookies -- Invalid cookie header: "set-cookie: __cf_bm=qkUbox2AkSAUg1JFQ7F5
XXGkK_d0Irs5ZYf7oCxX7wd2JR0V_pA0E5Wgoh37pqvDAnmYhFDprqJS45kNcLqGiKJZLZarj8IQVdLflilWGxYq9O6gZzgKAI
HttpOnly;   SameSite=None;   Secure;   Path=/;   Domain=repo.maven.apache.org;
Expires=Mon, 02 Nov 2026 20:42:18 GMT". Invalid 'expires' attribute: Mon, 02 Nov
2026 20:42:18 GMT
20:12:21.874     [BasicRepositoryConnector-repo.maven.apache.org-0-1]     WARN
o.a.h.c.p.ResponseProcessCookies -- Invalid cookie header: "set-cookie: __cf_bm=RAAMS0fvN7J4pgtTURWC
HttpOnly;   SameSite=None;   Secure;   Path=/;   Domain=repo.maven.apache.org;
Expires=Mon, 02 Nov 2026 20:42:18 GMT". Invalid 'expires' attribute: Mon, 02 Nov
2026 20:42:18 GMT
20:12:21.876     [BasicRepositoryConnector-repo.maven.apache.org-0-2]     WARN
o.a.h.c.p.ResponseProcessCookies -- Invalid cookie header: "set-cookie: __cf_bm=aTE4AgDp5LMoAzxbQK3Y
FqEMxk97VyD.vegymLllFt3atTOsMIYVGEoPAVNerUa2wZLZoKdMOJ4wBafS6Yu_RMP38LN_Q1TPf.vhkSVPfeHCQ8dLBX.yxD
HttpOnly;   SameSite=None;   Secure;   Path=/;   Domain=repo.maven.apache.org;
Expires=Mon, 02 Nov 2026 20:42:18 GMT". Invalid 'expires' attribute: Mon, 02 Nov
2026 20:42:18 GMT
20:12:21.931     [BasicRepositoryConnector-repo.maven.apache.org-0-3]     WARN
o.a.h.c.p.ResponseProcessCookies -- Invalid cookie header: "set-cookie: __cf_bm=ZBMJ19WeEB.PuWyYOShw
HttpOnly;   SameSite=None;   Secure;   Path=/;   Domain=repo.maven.apache.org;
Expires=Mon, 02 Nov 2026 20:42:21 GMT". Invalid 'expires' attribute: Mon, 02 Nov
2026 20:42:21 GMT
20:12:22.159     [BasicRepositoryConnector-repo.maven.apache.org-0-4]     WARN
o.a.h.c.p.ResponseProcessCookies -- Invalid cookie header: "set-cookie: __cf_bm=SEwOtsiqEogRHCrjfu4M
bdhoBMQQ.U4RDephay1M.0oFjmuAVEFBVig6MhYoB2uFXnm.YSAuDtepLIBdSe14JXMlPYSnpeSSF2ZuFkyApmSa3EqI0YZv1I
HttpOnly;   SameSite=None;   Secure;   Path=/;   Domain=repo.maven.apache.org;
Expires=Mon, 02 Nov 2026 20:42:22 GMT". Invalid 'expires' attribute: Mon, 02 Nov
2026 20:42:22 GMT
20:12:26.089  [IJava-executor-0]  WARN    o.a.h.c.p.ResponseProcessCookies  --
Invalid cookie header: "set-cookie: __cf_bm=Ce4.qNy6OLnjpHRvJqLd_wen9ZSuXHG8q2tL71EQ_eE-1793650342.
ScIySimAir6eM0KH23AY.ISsVlSXHPOallYCNMu5CGqcEKpzQ2RvJtIUVhTnAj6M2ZWjN03OHH3Om04hs6WPbJBX8ZPi0VEkkz
HttpOnly;   SameSite=None;   Secure;   Path=/;   Domain=repo.maven.apache.org;
Expires=Mon, 02 Nov 2026 20:42:22 GMT". Invalid 'expires' attribute: Mon, 02 Nov
2026 20:42:22 GMT
20:12:26.160                                                  [IJava-executor-0]
WARN  o.a.h.c.p.ResponseProcessCookies -- Invalid cookie header: "set-cookie:
__cf_bm=FQ6grZ9cAnIb.BkW.RX_ssLTJQJw4nBYPoPcN2Y2DeM-1793650346.121726-1.0.1.1-xyVn68RVQkvQsCaKUFzY
HttpOnly;   SameSite=None;   Secure;   Path=/;   Domain=repo.maven.apache.org;
Expires=Mon, 02 Nov 2026 20:42:26 GMT". Invalid 'expires' attribute: Mon, 02 Nov
2026 20:42:26 GMT
```

3 1. Premier test : du rouge au vert

Un flux GTFS realiste contient des noms sales : espaces de bordure, majuscules accidentelles, doubles espaces internes — hérités de saisies manuelles anciennes. Notre unité de production, purement textuelle :

> `NormaliseurNom.normaliser(String)` — coupe les espaces de bordure, réduit les suites d'espaces internes à une seule, met la première lettre en majuscule et le reste en minuscules ; `null` est rejeté par `IllegalArgumentException` ; une entrée sans caractère non-blanc donne une chaîne vide.

On commence par un **squelette** volontairement vide : le code compile, mais ne remplit pas le contrat. (Dans un noyau JShell, un test qui référence une classe inexistante ne compile pas du tout — le rouge « classe absente » n'est pas exécutable ici ; le rouge pédagogique est donc un rouge d'assertion, ce qui est de toute façon le plus fréquent en pratique.)

```
final class NormaliseurNom {
    static String normaliser(String nom) {
        return nom;                       // squelette : aucune normalisation
    }
}
```

Le test est une classe ordinaire dont les méthodes portent l'annotation `@Test`. Chaque méthode commence par le verbe de ce qu'elle affirme sur le contrat :

```
class NormalisationTest {
    @Test
    void espacesDeBordureSupprimes() {
        assertEquals("Pommier", NormaliseurNom.normaliser("  pommier  "));
    }

    @Test
    void seuleLaPremiereLettreEstEnMajuscule() {
        assertEquals("Chêne", NormaliseurNom.normaliser("CHÊNE"));
    }
}
```

Exécutons-les contre le squelette :

```
executerTests(NormalisationTest.class)
```

```
Tests run: 2, Failures: 2, Skipped: 0

Failures (2):
  JUnit Jupiter:$JShell$49$NormalisationTest:espacesDeBordureSupprimes()
    MethodSource [className = 'REPL.$JShell$49$NormalisationTest', methodName =
'espacesDeBordureSupprimes', methodParameterTypes = '']
      ⇒ org.opentest4j.AssertionFailedError: expected: <Pommier> but was: <
pommier  >
     org.junit.jupiter.api.AssertionFailureBuilder.build(AssertionFailureBuilder.java:151)
     org.junit.jupiter.api.AssertionFailureBuilder.buildAndThrow(AssertionFailureBuilder.java:132)
       org.junit.jupiter.api.AssertEquals.failNotEqual(AssertEquals.java:197)
       [...]
                                                                          JUnit
Jupiter:$JShell$49$NormalisationTest:seuleLaPremiereLettreEstEnMajuscule()
    MethodSource [className = 'REPL.$JShell$49$NormalisationTest', methodName =
'seuleLaPremiereLettreEstEnMajuscule', methodParameterTypes = '']
    ⇒ org.opentest4j.AssertionFailedError: expected: <Chêne> but was: <CHÊNE>
     org.junit.jupiter.api.AssertionFailureBuilder.build(AssertionFailureBuilder.java:151)
     org.junit.jupiter.api.AssertionFailureBuilder.buildAndThrow(AssertionFailureBuilder.java:132)
       org.junit.jupiter.api.AssertEquals.failNotEqual(AssertEquals.java:197)
       [...]
```

Deux échecs, deux messages `expected: <Pommier> but was: <  pommier  >` : le test rouge localise la trahison du contrat. C'est l'instant le plus précieux du TP — avant toute correction, l'échec dit exactement ce qui manque. Implantons :

```
final class NormaliseurNom {
    static String normaliser(String nom) {
        if (nom == null) throw new IllegalArgumentException("nom null");
        String propre = nom.strip().replaceAll("\\s+", " ");
        if (propre.isEmpty()) return propre;
        return Character.toUpperCase(propre.charAt(0))
             + propre.substring(1).toLowerCase();
    }
}
```

```
executerTests(NormalisationTest.class)
```

```
Tests run: 2, Failures: 0, Skipped: 0
```

Point de contrôle 1 — sortie : `Tests run: 2, Failures: 0, Skipped: 0`. C'est un cycle **rouge → vert** complet : échec observable, correction minimale, échec devenu succès.

4 2. Assertions usuelles et `assertThrows`

`assertEquals` n'est qu'une assertion parmi d'autres : `assertTrue`/`assertFalse`/`assertNull`/`assertNotNull` et leurs variantes (javadoc `Assertions`), et surtout `assertThrows`, qui affirme qu'un appel **doit** lever une exception — le comportement d'erreur fait partie du contrat, au même titre que le comportement normal. Le test qui suit exerce la branche `null` du squelette corrigé, plus un cas limite honnête (une entrée entièrement blanche donne une sortie blanche, pas une exception) :

```
class NormalisationRobustesseTest {
    @Test
    void nullEstRejete() {
        assertThrows(IllegalArgumentException.class,
                     () → NormaliseurNom.normaliser(null));
    }

    @Test
    void entreeBlancheDonneChaineVide() {
        assertEquals("", NormaliseurNom.normaliser("   "));
    }

    @Test
    void uneListeEntiereSeNormalise() {
        var bruts    = List.of("  pommier ", "CHÊNE", "laurier de  valboisé");
        var attendus = List.of("Pommier", "Chêne", "Laurier de valboisé");
        var obtenus  = bruts.stream().map(NormaliseurNom::normaliser).toList();
        assertEquals(attendus, obtenus);   // assertEquals marche sur les List
    }
}
```

```
executerTests(NormalisationRobustesseTest.class)
```

```
Tests run: 3, Failures: 0, Skipped: 0
```

Point de contrôle 2 — `Tests run: 3, Failures: 0, Skipped: 0`. Remarquez le troisième test : une assertion sur deux collections entières vaut trois assertions ponctuelles, et c'est la leçon 07 (listes triées, comparables) qui rend cela lisible.

5 3. Cycle de vie et lisibilité : `@BeforeEach`, `@DisplayName`

Chaque `@Test` doit être exécutable dans n'importe quel ordre — seul un test **à fraîcheur garantie** est fiable. JUnit crée une **nouvelle instance** de la classe de test avant chaque `@Test` — les champs d'instance sont donc neufs par construction ; un champ `static` lui, est partagé. JUnit offre pour cela des méthodes de cycle de vie, dont `@BeforeEach`, exécutée avant chaque test de la classe (il en existe d'autres : `@AfterEach`, `@BeforeAll`, `@AfterAll`). `@DisplayName` habille classes et méthodes d'un libellé lisible, utile dès qu'on écrit « en français ». Vérifions l'isolation fournie par `@BeforeEach` :

```
@DisplayName("Normaliseur — cycle de vie")
class CycleVieTest {
    private List<String> journal;                 // état privé à CHAQUE test

    @BeforeEach
    void preparer() {
        journal = new ArrayList<>();              // recréé avant chaque @Test
        journal.add("setup");
    }

    @Test
    @DisplayName("le journal est neuf au début de chaque test")
    void journalEstNeuf() {
        assertEquals(List.of("setup"), journal);
       journal.add("traces du test");           // ne sera PAS vu par le suivant
    }

    @Test
    @DisplayName("preuve directe : deux tests, deux univers")
    void secondTestNeRienHerite() {
          assertEquals(List.of("setup"), journal);   // la trace du précédent
a disparu
    }
}
```

```
executerTests(CycleVieTest.class)
```

```
Tests run: 2, Failures: 0, Skipped: 0
```

Point de contrôle 3 — `Tests run: 2, Failures: 0, Skipped: 0`. Expérience à faire en local : passez `journal` en `static`, remplacez dans `preparer()` l'affectation par `if (journal == null) journal = new ArrayList<>();` et prédisez avant d'exécuter — le

second test échouera (la trace du premier survit). Un état partagé entre tests est un défaut de conception des tests, pas du code testé.

6 4. Un test, cent jeux : `@ParameterizedTest`

Écrire un test par cas de normalisation ne tient pas la route. JUnit 5 permet de **paramétrer** un test : le même corps, exécuté pour chaque jeu d'entrées. `@ValueSource` couvre un paramètre, `@CsvSource` plusieurs (avec `quoteCharacter` pour préserver les espaces significatifs) :

```
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;
import org.junit.jupiter.params.provider.ValueSource;

@DisplayName("Normaliseur — tests paramétrés")
class NormalisationParametreeTest {
    @ParameterizedTest(name = "[{index}] «{0}» devient «{1}»")
    @CsvSource(value = {
            "\"  pommier  \",           \"Pommier\"",
            "\"CHÊNE\",                 \"Chêne\"",
            "\"laurier de  valboisé\",  \"Laurier de valboisé\"",
            "\"  GARE ROUTIERE  \",     \"Gare routiere\""
    }, quoteCharacter = '"')
    void normalisationAttendue(String brut, String attendu) {
        assertEquals(attendu, NormaliseurNom.normaliser(brut));
    }

    @ParameterizedTest
    @ValueSource(strings = {"", " ", "\t", " \n "})
    void touteEntreeBlancheDonneVide(String brut) {
        assertEquals("", NormaliseurNom.normaliser(brut));
    }
}
```

```
executerTests(NormalisationParametreeTest.class)
```

```
Tests run: 8, Failures: 0, Skipped: 0
```

Point de contrôle 4 — `Tests run: au moins 8 (4 + 4 + vos cas ajoutés)`, `Failures: 0`. Ajoutez un cas litigieux qui vous passe par la tête (nom à trait d'union, particule…) ; s'il échoue, décidez si c'est le test ou la spécification qui était faux — c'est exactement le travail de l'ingénieur.

7 5. Tester le DAO de la leçon 08 sur H2 en mémoire

Le vrai sujet arrive : peut-on tester du code qui parle à une base de données sans base de données ? La leçon 08 a répondu oui structurellement — le DAO ne dépend que de l'interface `Connection`. Nous rappelons (à recopier mentalement depuis la leçon 08) le modèle et les deux méthodes, `chercherParNom` étant ici la **version corrigée** de fin de leçon :

```
record Stop(String id, String nom, double latitude, double longitude) {}

List<Stop> lireLesArrets(Connection connexion) throws SQLException {
    List<Stop> resultat = new ArrayList<>();
    String sql = "SELECT id, nom, latitude, longitude FROM arrets ORDER BY nom";
    try (Statement stmt = connexion.createStatement();
         ResultSet rs = stmt.executeQuery(sql)) {
        while (rs.next())
            resultat.add(new Stop(rs.getString("id"), rs.getString("nom"),
                         rs.getDouble("latitude"), rs.getDouble("longitude")));
    }
    return resultat;
}

List<Stop> chercherParNom(Connection connexion, String nom) throws SQLException {
    List<Stop> trouve = new ArrayList<>();
    try (PreparedStatement p = connexion.prepareStatement(
             "SELECT id, nom, latitude, longitude FROM arrets WHERE nom = ?")) {
        p.setString(1, nom);
        try (ResultSet rs = p.executeQuery()) {
            while (rs.next())
                trouve.add(new Stop(rs.getString(1), rs.getString(2),
                                    rs.getDouble(3), rs.getDouble(4)));
        }
    }
    return trouve;                                   // jamais null — contrat 07
}
```

Le test d'intégration JDBC suit le même patron que les tests précédents, avec une exigence nouvelle : **une base fraîche par test**, sinon les tests se contaminent (une insertion du test A vue par le test B est un faux vert). Le numéro de base incrémenté à chaque `@BeforeEach` garantit cette isolation ; H2 en mémoire rend chaque cycle quasi instantané :

```
@DisplayName("DAO des arrêts — contrats 07 et 08 sous test")
class ArretDAOTest {
```

```
    private static int sequence = 0;
    // Contre-exemple assumé : le compteur ne partage aucun état OBSERVÉ par le
    // code testé — il alloue des bases disjointes ; ce qu'on interdit, c'est
    // l'état visible entre tests.
    private Connection connexion;

    @BeforeEach
    void creerBaseFraiche() throws SQLException {
        connexion = DriverManager.getConnection(
         "jdbc:h2:mem:tp" + (++sequence) + ";MODE=PostgreSQL;DB_CLOSE_DELAY=-1",
            "sa", "");
        try (Statement stmt = connexion.createStatement()) {
            stmt.execute("""
                CREATE TABLE arrets (
                    id VARCHAR(3) PRIMARY KEY,
                    nom VARCHAR(50) NOT NULL,
                    latitude DOUBLE PRECISION NOT NULL,
                    longitude DOUBLE PRECISION NOT NULL)""");
            stmt.execute("""
                INSERT INTO arrets VALUES
              ('POM','Pommier',43.125,6.032), ('CHE','Chêne',43.130,6.040)""");
        }
    }

    @AfterEach
    void fermer() throws SQLException { connexion.close(); }

    @Test
    @DisplayName("lireLesArrets retourne la liste triée, jamais null")
    void lectureTriee() throws SQLException {
        List<Stop> arrets = lireLesArrets(connexion);
        assertNotNull(arrets);
        assertEquals(List.of("Chêne", "Pommier"),
                     arrets.stream().map(Stop::nom).toList());
    }

    @Test
     @DisplayName("arrêt inconnu : liste vide, jamais null (question 4 de la
leçon 08)")
    void inconnuRenvoieListeVide() throws SQLException {
        List<Stop> res = chercherParNom(connexion, "Laurier");
        assertNotNull(res);
        assertTrue(res.isEmpty());
    }
```

```
    @Test
    @DisplayName("ma base est fraîche : mon insertion n'est visible de personne")
    void isolationParTest() throws SQLException {
        try (Statement stmt = connexion.createStatement()) {
                                stmt.executeUpdate("INSERT  INTO  arrets  VALUES
('TIL','Tilleul',43.11,6.05)");
        }
        assertEquals(3, lireLesArrets(connexion).size());
      // Les tests voisins tournent sur tp<k> distincts : Tilleul n'existe qu'ici.
    }

    @Test
    @DisplayName("connexion fermée : SQLException, pas de plantage silencieux")
    void connexionFermee() throws SQLException {
        connexion.close();
        assertThrows(SQLException.class, () -> lireLesArrets(connexion));
    }
}
```

```
executerTests(ArretDAOTest.class)
```

```
Tests run: 4, Failures: 0, Skipped: 0
```

Point de contrôle 5 — `Tests run: 4, Failures: 0, Skipped: 0`. Interprétez : si vous remettez `return trouve.isEmpty() ? null : trouve;` dans `chercherParNom`, quel test rouge vous le dit, et en combien de millisecondes ? Vérifiez en local, puis restaurez.

8 6. Mini-cycle TDD guidé

À vous de parcourir le cycle complet seul. Spécification en trois lignes :

> `FormateurArret.formatage(Stop s)` — renvoie `« nom (id) »`, par exemple `Pommier (POM)` ; le `null` est rejeté par `IllegalArgumentException` ; le nom est d'abord normalisé (on réutilise `NormaliseurNom`).

Consigne, dans cet ordre précis : **(i)** écrire d'abord `FormateurArretTest` (audacieux, mais vous savez le faire : au moins un `@ParameterizedTest`, un cas `null` via `assertThrows`, un cas où le nom sale du flux d'entrée ressort propre) ; **(ii)** un squelette qui compile mais échoue (`return "?";`) ; **(iii)** exécution rouge — lisez chaque message d'échec ; **(iv)** implantation minimale pour passer vert ; **(v)** exécution finale. Les cellules sont à vous :

```
// (i) Votre classe de test d'abord. Exemple de structure à personnaliser :
// class FormateurArretTest {
//       // le Stop se construit dans le corps : @CsvSource ne fournit que
des chaînes
//     @ParameterizedTest(name = "{0} {1} → {2}")
//     @CsvSource({"POM, pommier , Pommier (POM)", …})
//     void formatageAttendu(String id, String nom, String attendu) {
//         Stop s = new Stop(id, nom, 0, 0);
//         assertEquals(attendu, FormateurArret.formatage(s));
//     }
//     @Test void nullEstRejete() { … }
// }
```

```
// (ii) Squelette rouge :
final class FormateurArret {
    static String formatage(Stop s) {
        if (s == null) throw new IllegalArgumentException("arrêt null");
        return "?";
    }
}
```

```
// (iii) Rouge : rédigez la classe de test en (i), puis décommentez l'appel :
// executerTests(FormateurArretTest.class)
```

```
// (iv) Votre implantation (évoluez le squelette par re-définition, comme en §1)
```

```
// (v) executerTests(FormateurArretTest.class)  →  vert
```

Point de contrôle 6 — votre exécution finale affiche `Failures: 0` et vous avez observé une exécution rouge avec de vrais messages d'échec. Un vert obtenu sans rouge préalable ne prouve rien : c'est le rouge qui atteste que le test sait détecter l'absence de fonctionnalité.

9 7. Compte rendu

Répondez par écrit (une demi-page à deux pages, sans recopier la cellule de préparation) :

1. En quoi l'exécution rouge du §1 vous a-t-elle appris sur `normaliser` quelque chose que la lecture du squelette ne disait pas ? Votre premier rouge était un rouge d'assertion, pas de compilation — pourquoi cette distinction importe-t-elle dans un vrai projet Maven/IDE ?

2. Pourquoi une base fraîche par test en §5 plutôt qu'une base partagée ? Donnez un scénario concret de faux vert ou de faux rouge.
3. Le contrat « jamais `null` » de la leçon 07 : quelles assertions précises le garantissent ici, et que se passerait-il à la prochaine régression ?
4. `assertThrows` plutôt qu'un `try { … } catch { fail(); }` maison : qu'y gagne-t-on ?
5. **Optionnel (bonus)** — Les tests du §5 ne couvrent ni le `PreparedStatement` contre l'injection, ni les transactions, ni le batch de la leçon 08. Choisissez-en un et esquissez le test qui le verrouillerait (description, pas implémentation).
6. Dans un projet de groupe : à quoi sert d'écrire les tests avant ? Quand est-ce un luxe, et quand une nécessité ?

Rendu : le carnet exécuté (les six points de contrôle `Failures: 0` visibles dans la sortie rendue) + les réponses. Pondération : exécution des 6 points de contrôle + réponses Q1–Q4 ; Q5–Q6 bonus. En CI, la branche `develop` rendue tient lieu de preuve d'exécution ; en local, exportez ou rendez le notebook exécuté.

10 Conclusion

Réutilisation (leçon 07), portabilité (leçon 08), maintenabilité (ce TP) : les trois termes de l'arc ne forment qu'une idée. Un contrat qui s'exprime en tests est un contrat que la machine vérifie pour vous — et le code entouré de tels tests est exactement celui qu'on ose réutiliser et modifier. Les limites de ce TP sont celles de l'outil, pas de la discipline : en vrai projet, `mvn test` ou `gradle test` découvre les classes tout seul et la CI les exécute hors rendu ; les mécanismes (découverte, rapports, isolation) que nous avons manipulés à la main sont alors délégués à l'outillage — pour les mêmes raisons pédagogiques qu'ici : rendre le contrat vérifiable à chaque instant.