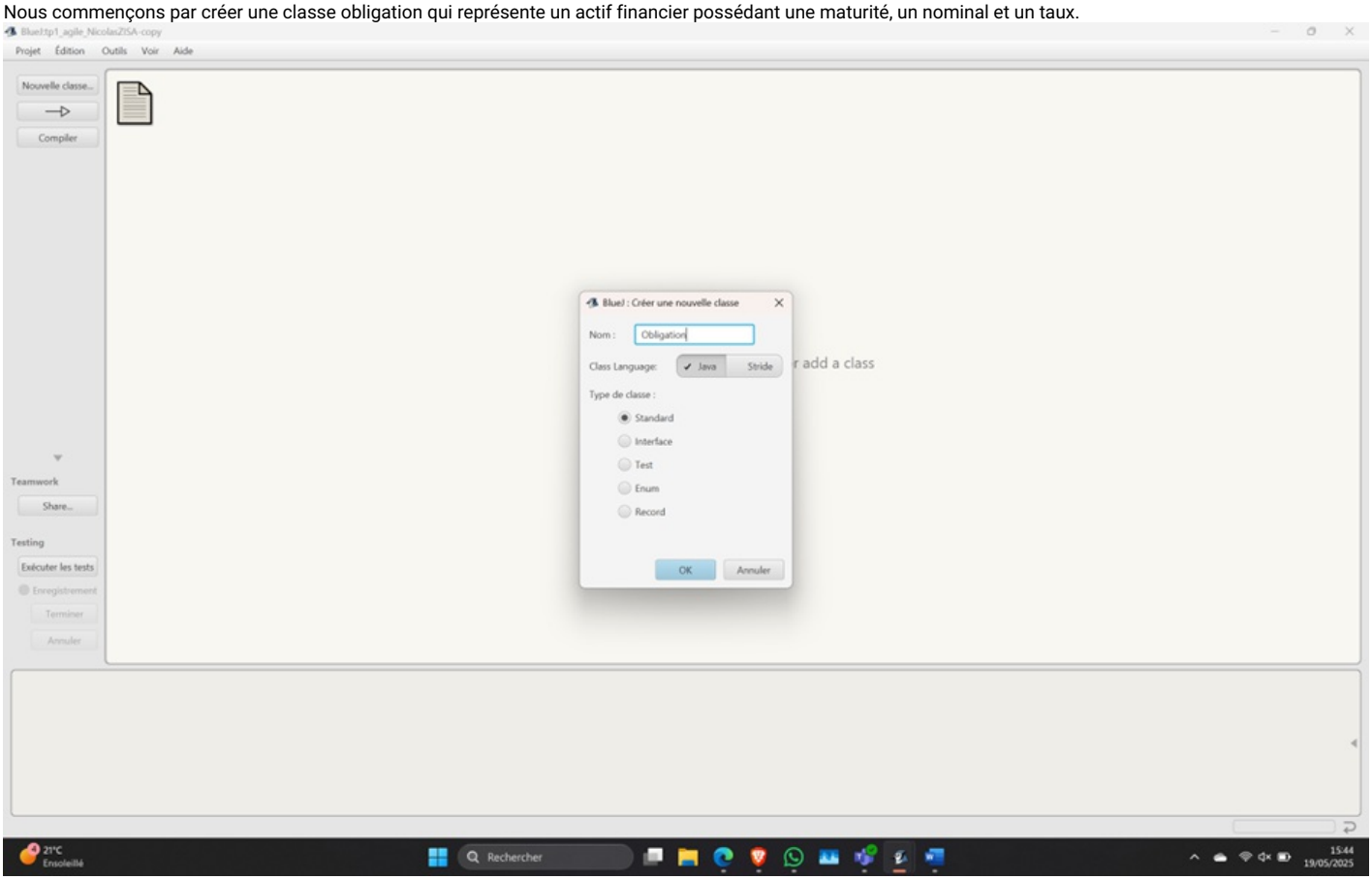
Tutoriel : La programmation orientée-objet pour financiers très très motivés

Courte histoire pour une mise en jambe efficace

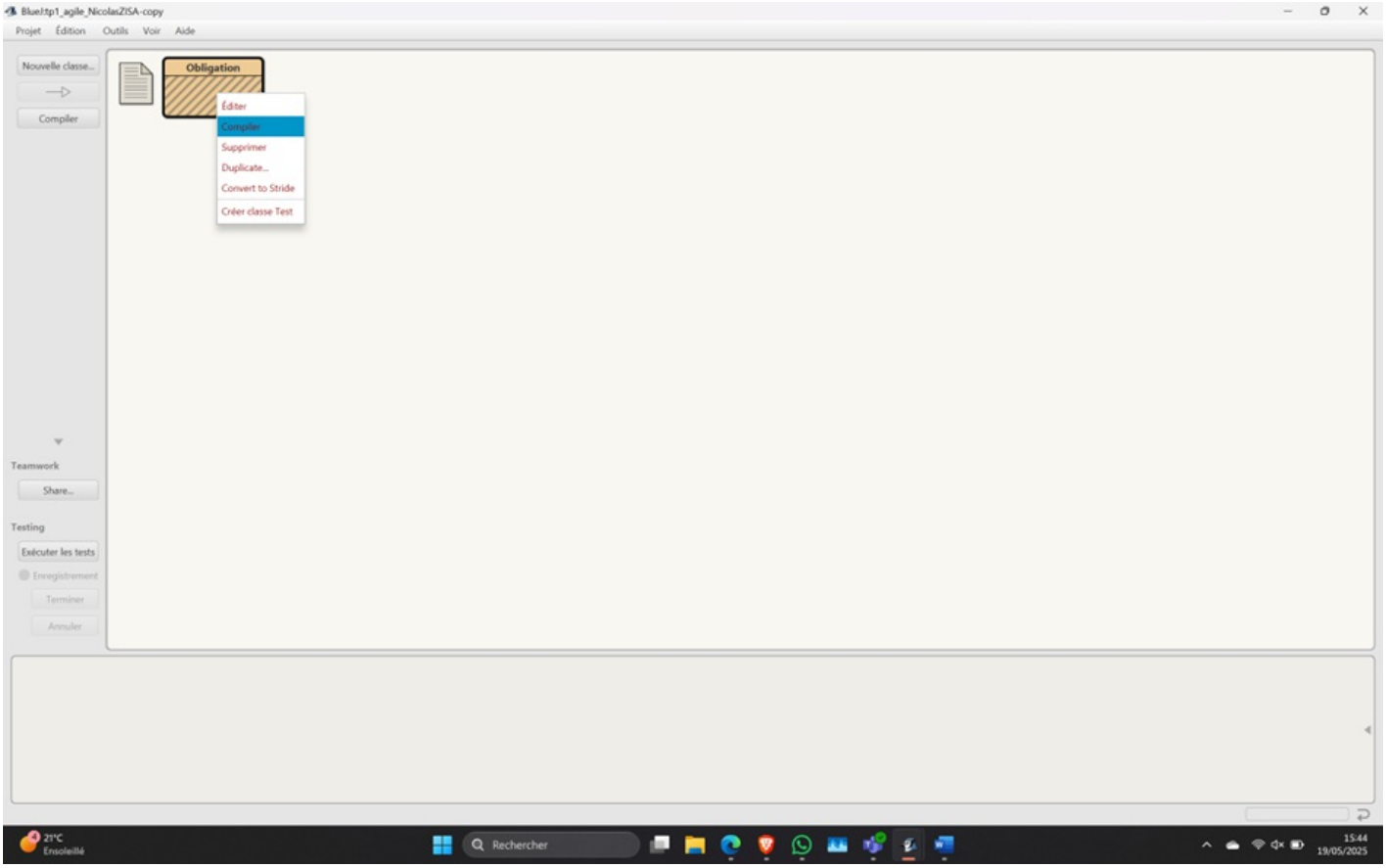
Pierre , financier expert sur le marché des obligations, en avait marre de jongler avec ses tableurs excel chargés de formules et illisibles. Un soir, après un énième café, il se dit : “Stop, j'en peux plus, il me faut un truc qui calcul tout, tout le temps et à ma place !”. Il se rendit sur internet, à la recherche d'un tutoriel qui le ménera vers un avenir radieux. "Très bien, il me suffit d'apprendre à coder, ça ne devrait pas être si difficile" pensa t-il à haute voix. "Mais comment m'assurer que mon code fonctionne, je pourrais perdre des millions d'euros et finir par vendre des paninis à chateaurou" s'inquiéta t-il. Après des heures de recherches, au bord de l'abandon, il découvrit une vidéo youtube lui introduisant le principe des tests Junit, fonctionnels et l'IA. "Eureka !" s'écriat-il. Après une bonne nuit de sommeil, une tasse de café à la main, il s'attela à sa formation.

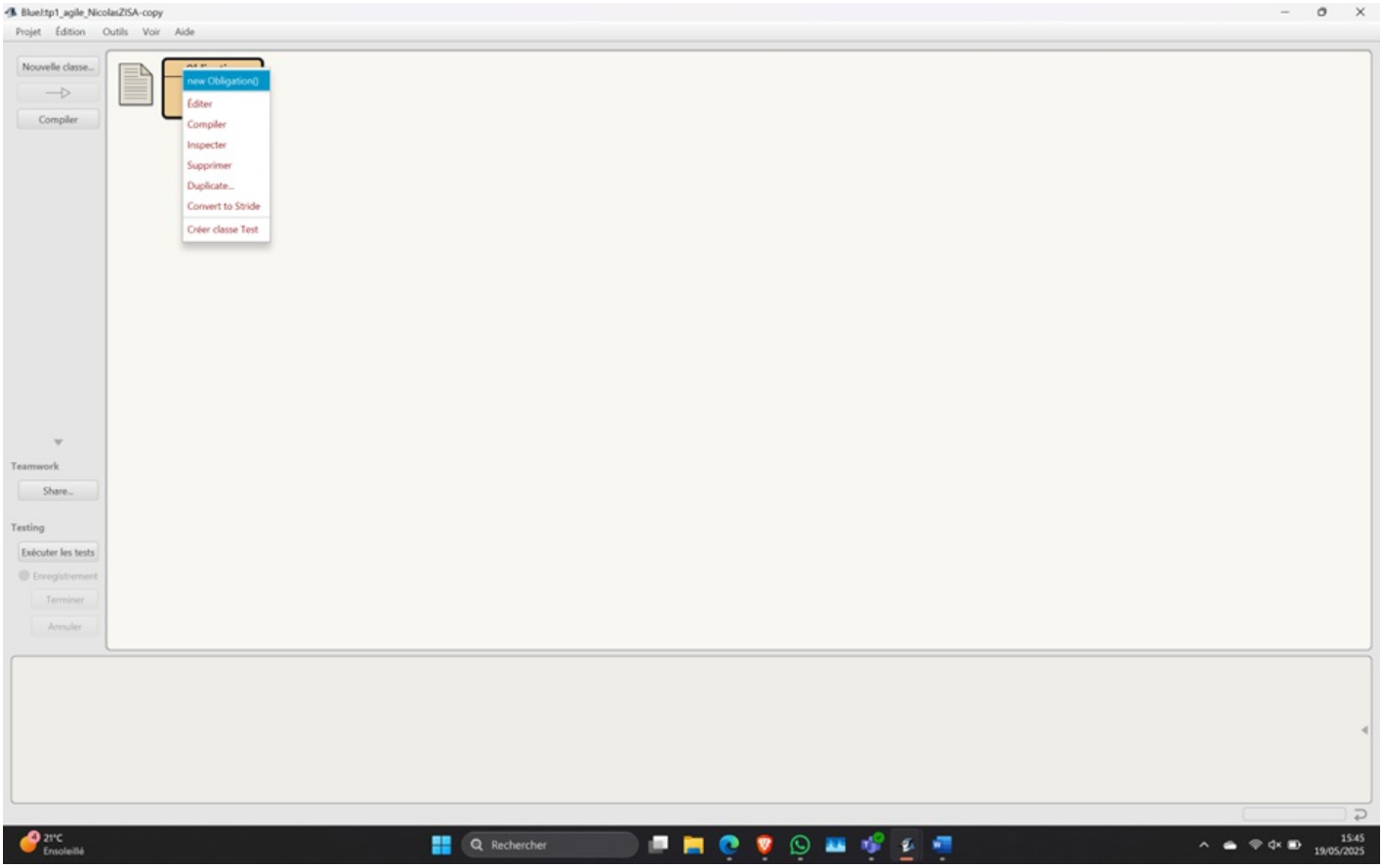
Le tutoriel avec BlueJ

Nous commençons par créer une classe obligation qui représente un actif financier possédant une maturité, un nominal et un taux.



Nous pouvons la compiler et l’instancier même sans schéma clair de sa constitution.



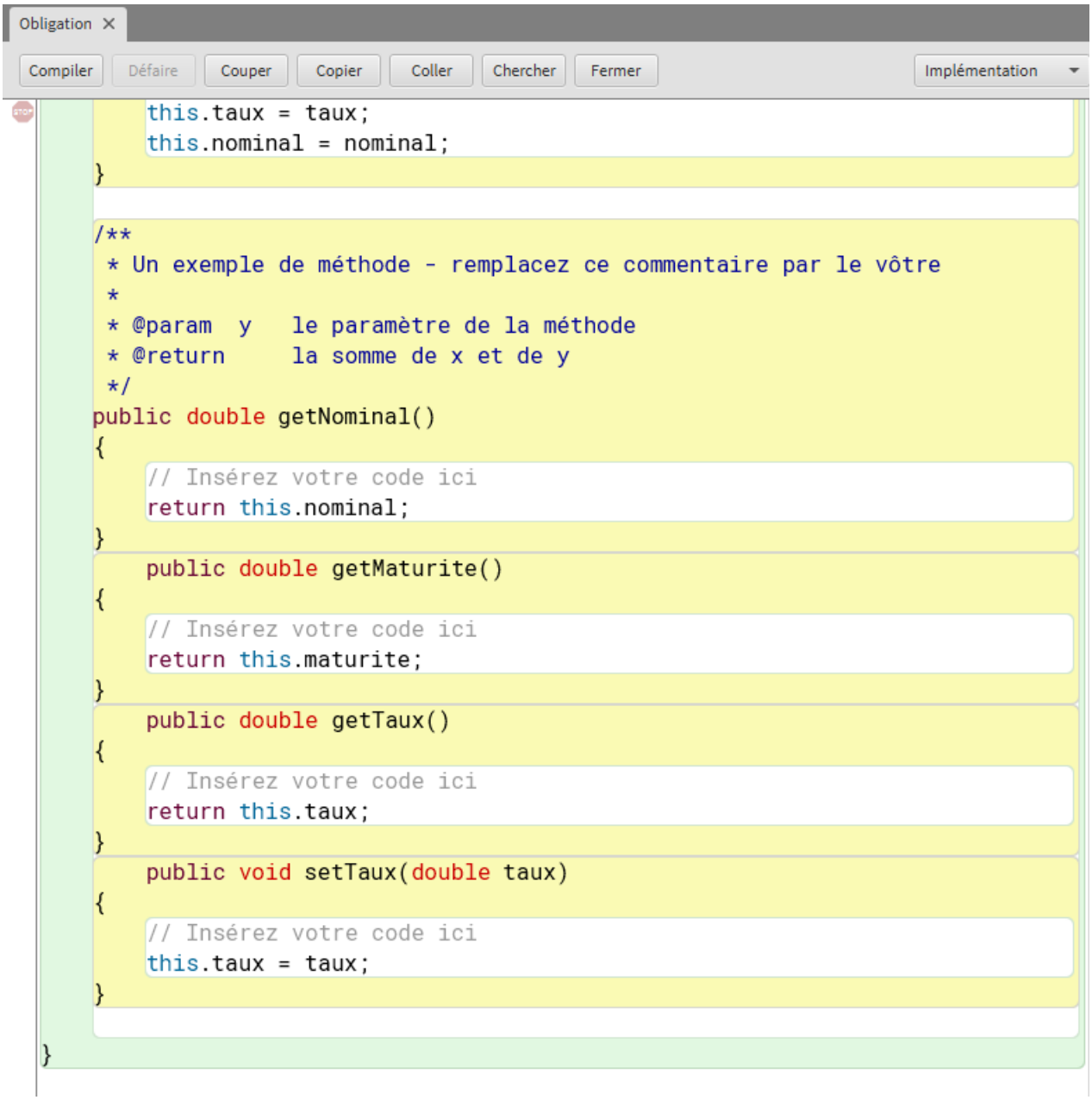


Commençons à mieux définir notre classe.

Obligation ×

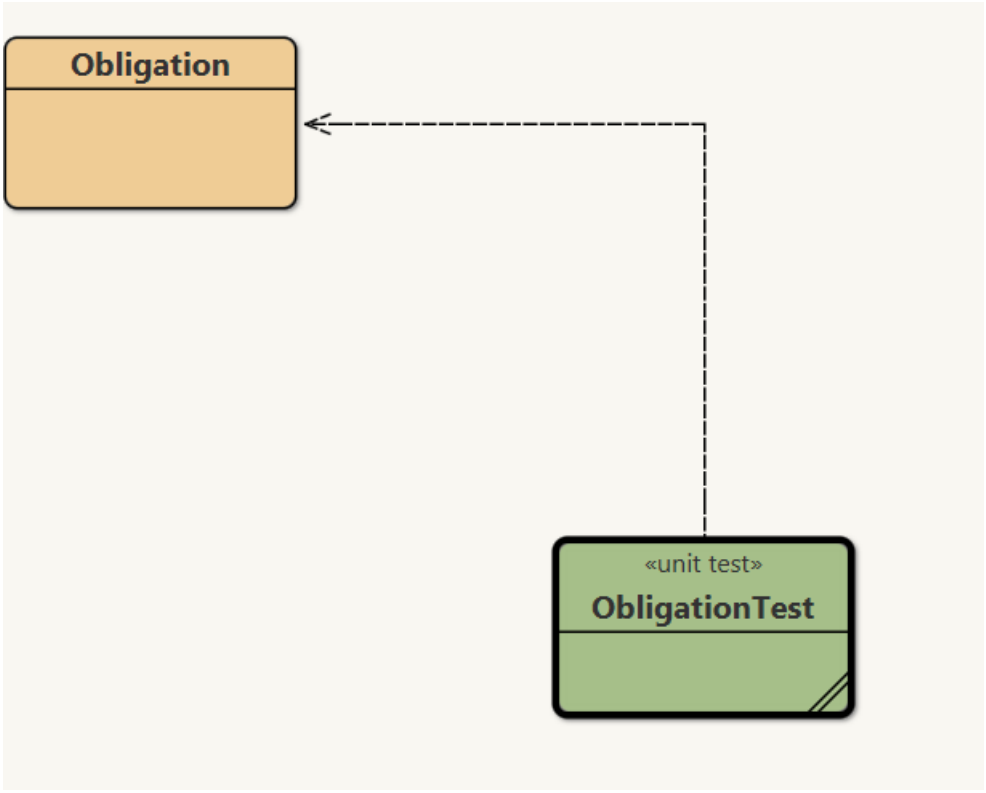
Compiler | Défaire | Couper | Copier | Coller | Chercher | Fermer | Implémentation

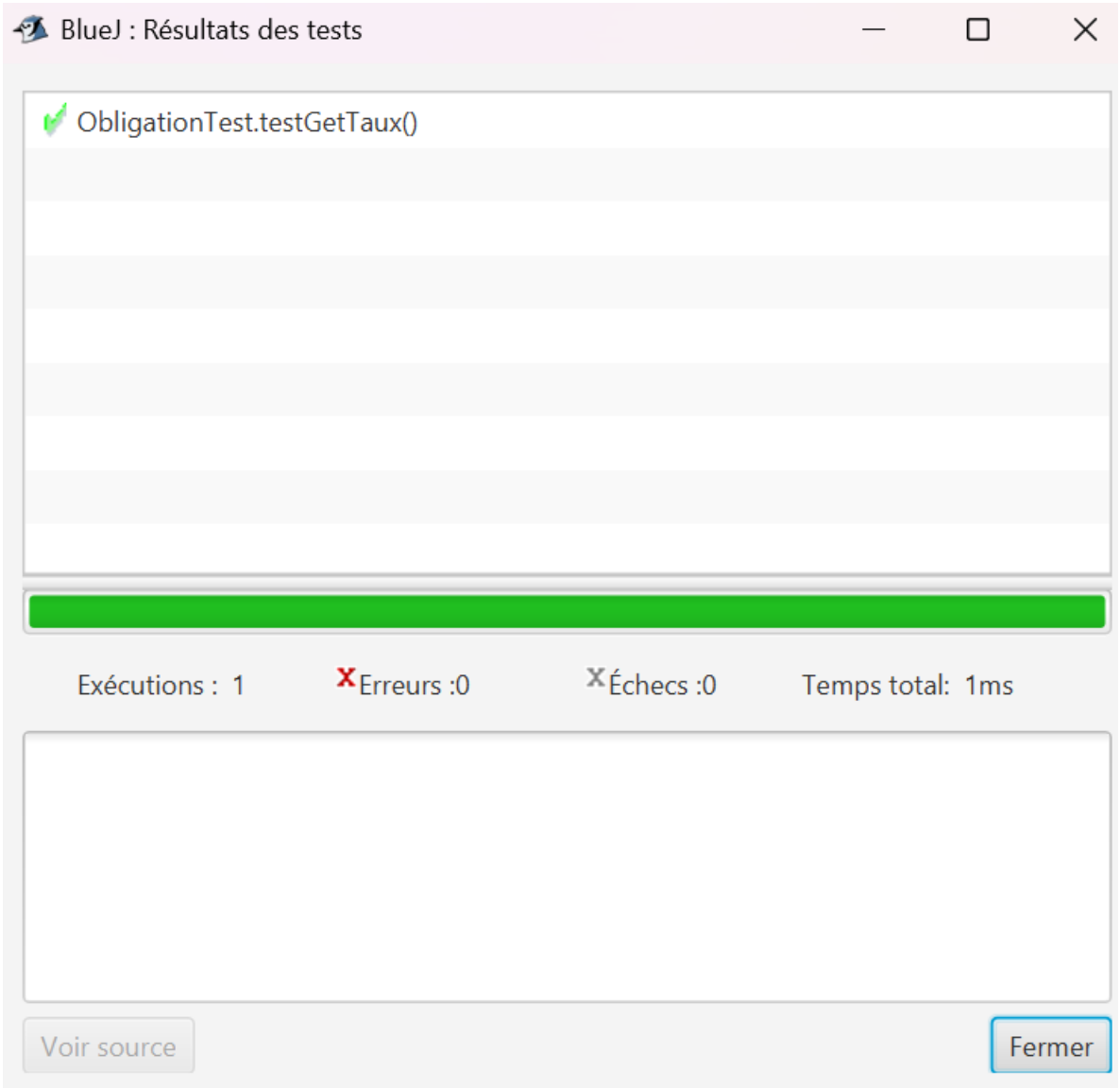
```java
public class Obligation
{
    // variables d'instance - remplacez l'exemple qui suit par le vôtre
    private int maturite;
    private double taux;
    private double nominal;


    /**
     * Constructeur d'objets de classe Obligation
     */
    public Obligation()
    {
        // initialisation des variables d'instance
        this.maturite = 0;
        this.taux = 0;
        this.nominal = 0;
    }
    public Obligation(int maturite, double taux, double nominal)
    {
        // initialisation des variables d'instance
        this.maturite = maturite;
        this.taux = taux;
        this.nominal = nominal;
    }

    /**
     * Un exemple de méthode - remplacez ce commentaire par le vôtre
     *
     * @param  y   le paramètre de la méthode
     * @return     la somme de x et de y
     */
    public double getNominal()
```



Instancions notre classe et essayons nos méthodes GetTaux. getNominal et getMaturite

Maintenant pour notre premier test nous créons une classe test, nous la compilons, nous définissons notre première méthode de test pour la méthode GetTaux.





Maintenant nous créons une nouvelle classe Portefeuille, qui va contenir une obligation à un instant t.

Portefeuille

Compiler | Défaire | Couper | Copier | Coller | Chercher | Fermer | Implémentation

```java
import java.util.List;

public class Portefeuille {
    private List<Obligation> obligations;
    private int annee;

    public Portefeuille(List<Obligation> obligations, int annee) {
        this.obligations = obligations;
        this.annee = annee;
    }

    public double computePosition() {
        double total = 0;
        for (Obligation ob : obligations) {
            if (annee <= ob.getMaturite()) {
                Obligation temp = new Obligation(ob.getNom(), annee, ob.getTaux(
                total += ob.getStrategie().calculer(temp);
            }
        }
        return total;
    }
}
```

Nous allons poursuivre le développement en suivant le pattern strategy, commençons par réaliser notre classe interface StrategieRendement

StrategieRendement

Compiler | Défaire | Couper | Copier | Coller | Chercher | Fermer | Implémentation

```java
/**
 * Décrivez votre interface StrategieRendement ici.
 *
 * @author  (votre nom)
 * @version (un numéro de version ou une date)
 */

public interface StrategieRendement
{
    /**
     * Exemple d'entête de méthode - remplacez ce commentaire par le vôtre
     *
     * @param  y    le paramètre de cette méthode
     * @return        le résultat retourné par exempleDeMethode
     */
    double calculer(Obligation obligation);
}
```

Puis la classe RendementTauxFixe qui l'implèmente

RendementTauxFixe

Compiler | Défaire | Couper | Copier | Coller | Chercher | Fermer | Implémentation

```java
/**
 * Décrivez votre classe RendementTauxFixe ici.
 *
 * @author (votre nom)
 * @version (un numéro de version ou une date)
 */
public class RendementTauxFixe implements StrategieRendement {
    @Override
    public double calculer(Obligation obligation) {
        return obligation.getNominal() * Math.pow(1 + obligation.getTaux(), obli
    }
}
```

Notre Facotry d'obligation

ObligationFactory

Compiler | Défaire | Couper | Copier | Coller | Chercher | Fermer | Implémentation

```java
public class ObligationFactory {
    public static Obligation creerObligation(String nom, int maturite, double taux, double nominal, StrategieRendement strategie) {
        if (strategie == null) {
            strategie = new RendementTauxFixe();
        }
        return new Obligation(nom, maturite, taux, nominal, strategie);
    }
}
```

Et enfin notre Coach IA, si précieux pour développer les meilleures stratégies d'investissement

CoachIA

Compiler | Défaire | Couper | Copier | Coller | Chercher | Fermer

```java
import java.util.*;

public class CoachIA {
    private List<Obligation> obligations;
    private double capital;
    private int nombre;

    public CoachIA(List<Obligation> obligations, double capital, int nombre) {
        this.obligations = obligations;
        this.capital = capital;
        this.nombre = nombre;
    }

    public List<Obligation> entrainerCoach() {
        List<Obligation> sorted = new ArrayList<>(obligations);
        sorted.sort((a, b) -> Double.compare(b.getRendement(), a.getRendement()));

        List<Obligation> selection = new ArrayList<>();
        double solde = capital;

        for (Obligation ob : sorted) {
            if (selection.size() >= nombre) break;
            if (ob.getNominal() <= solde) {
                selection.add(ob);
                solde -= ob.getNominal();
```

Dorénavement, réalisons des tests JUnit sur notre solution

Des Tests sur Obligation

ObligationTest ×

Compiler | Défaire | Couper | Copier | Coller | Chercher | Fermer | Implémentation

```java
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class ObligationTest {
    private Obligation ob;

    @BeforeEach
    public void setUp() {
        ob = new Obligation("SG", 5, 0.01, 1000, new RendementTauxFixe());
    }

    @Test
    public void testCalculRendementFixe() {
        double expected = 1000 * Math.pow(1.01, 5);
        assertEquals(expected, ob.getRendement(), 0.01);
    }

    @Test
    public void testGetTaux() {
        assertEquals(0.01, ob.getTaux(), 0.0001, "Le taux doit être 0.01");
    }
}
```

PortefeuilleTest

Compiler | Défaire | Couper | Copier | Coller | Chercher | Fermer | Implémenta

```java
import java.util.ArrayList;
import java.util.List;

public class PortefeuilleTest {

    private Obligation obl1;
    private Obligation obl2;
    private List<Obligation> obligations;
    private Portefeuille portefeuille;

    @BeforeEach
    public void setUp() {
        StrategieRendement strategie = new RendementTauxFixe();
        obl1 = new Obligation("SG", 8, 0.003, 1500.0, strategie);
        obl2 = new Obligation("BNP", 6, 0.005, 2000.0, strategie);

        obligations = new ArrayList<>();
        obligations.add(obl1);
        obligations.add(obl2);

        portefeuille = new Portefeuille(obligations, 5);
    }

    @Test
    public void testComputePosition() {
        double expectedPosition = obl1.getStrategie().calculer(
            new Obligation("SG", 5, 0.003, 1500.0, obl1.getStrategie())
        ) +
        obl2.getStrategie().calculer(
            new Obligation("BNP", 5, 0.005, 2000.0, obl2.getStrategie())
        );

        double actualPosition = portefeuille.computePosition();
        assertEquals(expectedPosition, actualPosition, 0.01);
    }
}
```

Des Tests sur Portefeuille

Des Tests sur ObligationFactory

ObligationFactoryTest

Compiler | Défaire | Couper | Copier | Coller | Chercher | Fermer | Implémentation

```java
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class ObligationFactoryTest {
    private StrategieRendement strategie;

    @BeforeEach
    public void setUp() {
        strategie = new RendementTauxFixe();
    }

    @Test
    public void testCreerObligationAvecStrategie() {
        Obligation obl = ObligationFactory.creerObligation("AXA", 5, 0.02, 1000,

        assertNotNull(obl);
        assertEquals("AXA", obl.getNom());
        assertEquals(1000 * Math.pow(1 + 0.02, 5), obl.getRendement(), 0.01);
        assertTrue(obl.getStrategie() instanceof RendementTauxFixe);
    }

    @Test
    public void testCreerObligationSansStrategie() {
        Obligation obl = ObligationFactory.creerObligation("CA", 3, 0.01, 2000,

        assertNotNull(obl);
        assertEquals("CA", obl.getNom());
        assertTrue(obl.getStrategie() instanceof RendementTauxFixe);
    }
}
```

Des Tests sur RendementTauxFixe

```java
public class RendementTauxFixeTest {
    private RendementTauxFixe strategie;
    private Obligation obligation;

    @BeforeEach
    public void setUp() {
        strategie = new RendementTauxFixe();
        obligation = new Obligation("BNP", 4, 0.03, 1000, strategie);
    }

    @Test
    public void testCalculerRendement() {
        double attendu = 1000 * Math.pow(1 + 0.03, 4);  // 1000 * (1.03)^4
        double calcul = strategie.calculer(obligation);

        assertEquals(attendu, calcul, 0.01, "Le rendement calculé est incorrec
    }

    @Test
    public void testStrategieInstance() {
        assertTrue(strategie instanceof RendementTauxFixe, "La stratégie n'est
    }
}
```

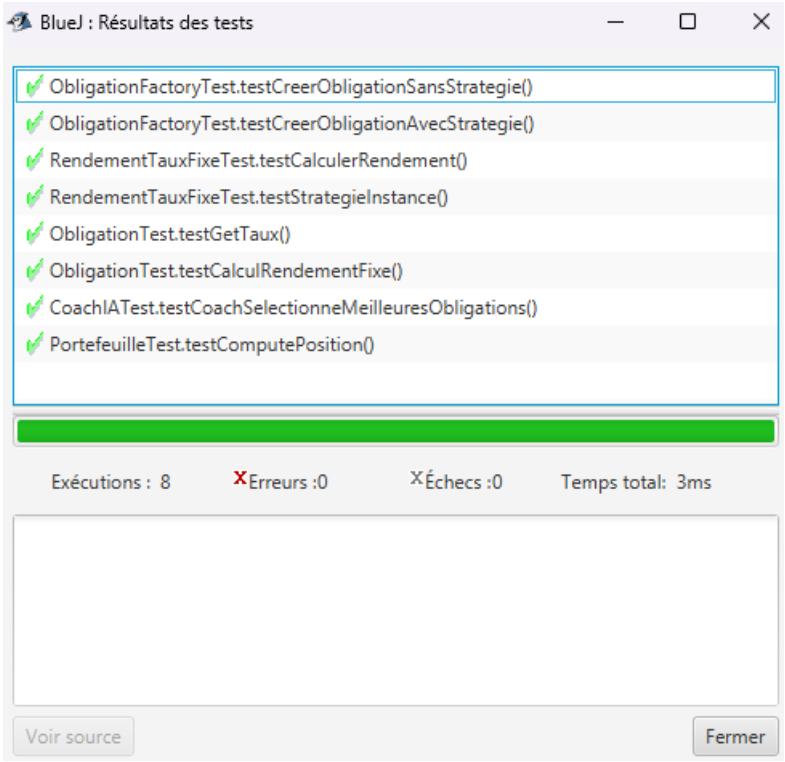
Des Tests sur CoachIA

CoachIATest

Compiler | Défaire | Couper | Copier | Coller | Chercher | Fermer | Implémentation

```
import org.junit.jupiter.api.Test;

import java.util.*;

import static org.junit.jupiter.api.Assertions.*;

public class CoachIATest {

    private List<Obligation> obligations;
    private StrategieRendement strategie;

    @BeforeEach
    public void setup() {
        strategie = new RendementTauxFixe();
        obligations = new ArrayList<>();

        obligations.add(new Obligation("AXA", 10, 0.03, 1500, strategie));
        obligations.add(new Obligation("BNP", 8, 0.02, 2000, strategie));
        obligations.add(new Obligation("SG", 5, 0.01, 1000, strategie));
    }

    @Test
    public void testCoachSelectionneMeilleuresObligations() {
        CoachIA coach = new CoachIA(obligations, 4000, 2);
        List<Obligation> selection = coach.entrainerCoach();



        assertEquals(2, selection.size(), "Doit sélectionner 2 obligations");


        assertEquals("BNP", selection.get(0).getNom(), "1re obligation attendue
        assertEquals("AXA", selection.get(1).getNom(), "2e obligation attendue :
    }
}
```

Il suffit de cliquer sur "Executer les tests" et avec beaucoup d'espoir, tout devrait être vert !



Behave et PyCharm

Il s'agit ici d'implémenter nos classes de BlueJ en Python, pour cela nous allons utiliser l'IDE PyCharm.

```python
class StrategieRendement(Protocol):  2 usages
    def calculer(self, obligation: Obligation) -> float:  2 usages
        ...

class RendementTauxFixe:  2 usages
    def calculer(self, obligation: Obligation) -> float:
        return obligation.nominal * (1 + obligation.taux) ** obligation.maturite

class ObligationFactory:
    @staticmethod
    def creer_obligation(
        nom: str, maturite: int, taux: float, nominal: float,
        strategie: StrategieRendement = None
    ) -> Obligation:
        if strategie is None:
            strategie = RendementTauxFixe()
        return Obligation(nom, maturite, taux, nominal, strategie)
```

```python
@dataclass(order=True)  28 usages
class Obligation:
    nom: str
    maturite: int
    taux: float
    nominal: float
    strategie_rendement: StrategieRendement = field(default_factory=RendementTauxFixe, repr=False, compare=False)
    rendement: float = field(init=False, repr=False, compare=False)

    def __post_init__(self) -> None:
        if self.maturite <= 0:
            raise ValueError("La maturité doit être positive")
        if self.taux < 0:
            raise ValueError("Le taux doit être positif")
        if self.nominal <= 0:
            raise ValueError("Le nominal doit être positif")
        self.rendement = self.calcul_rendement()

    def calcul_rendement(self) -> float:  3 usages (1 dynamic)
        self.rendement = self.strategie_rendement.calculer(self)
        return self.rendement

@dataclass  4 usages
class Portefeuille:
    obligations: List[Obligation]
    annee: int

    def compute_position(self) -> float:  2 usages (1 dynamic)
        total = 0
        for obl in self.obligations:
            if self.annee <= obl.maturite:
                total += obl.strategie_rendement.calculer(
                    Obligation(obl.nom, self.annee, obl.taux, obl.nominal, obl.strategie_rendement)
                )
        return total
```

```python
@dataclass  2 usages
class Financier:
    obligation: Obligation | None = None
    boolean: bool = False

class CoachIA:  5 usages
    def __init__(self, obligations: List[Obligation], capital: float, nombre: int):
        self.obligations = obligations
        self.capital = capital
        self.nombre = nombre

    def entrainer_coach(self) -> List[Obligation]:  3 usages
        candidates = sorted(self.obligations, key=lambda ob: ob.rendement, reverse=True)
        selection: List[Obligation] = []
        solde = self.capital

        for ob in candidates:
            if len(selection) >= self.nombre:
                break
            if ob.nominal <= solde:
                selection.append(ob)
                solde -= ob.nominal
        return selection
```

Dorénavant nous allons effectuer des tests fonctionnels dits de Behavior-Driven Development

Pour cela nous posons une première feature, enregistrer des obligations. Nous y voyons deux scénarios, le premier est le calcul automatique du rendement d'une obligation à maturité, l'autre est une vérification de la validité de l'obligation.

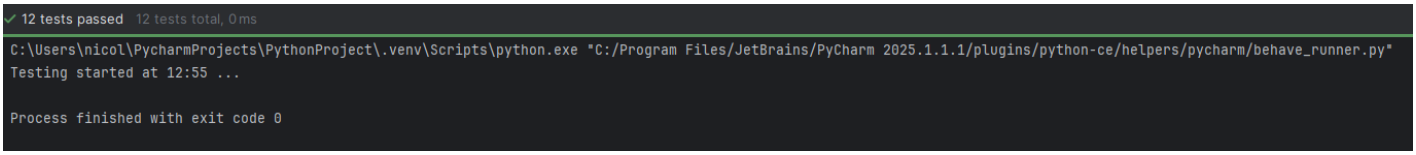
```
Feature: US_0001 - Enregistrer les obligations

En tant que financier
Je veux enregistrer les différentes obligations avec leurs maturités, taux et nominals respectifs.
Ceci afin de développer la vision que j'ai de ma position.

  Scenario Outline: Calcul automatique du rendement à maturité
    Given l'enregistrement d'une obligation avec <nom1>, <maturite1>, <taux1>, <nominal1>
    When l'utilisateur la valide
    Then le <rendement1> est calculé

    Examples:
      | nom1 | maturite1 | taux1 | nominal1 | rendement1 |
      | SG   | 8         | 0.003 | 1500     | 1536.38    |

  Scenario Outline: Refus d'une obligation à maturité, taux ou nominal négatif
    Given l'enregistrement d'une obligation avec <nom1>, <maturite1>, <taux1>, <nominal1>
    When l'utilisateur valide
    Then le système la refuse <isBad>

    Examples:
      | nom1 | maturite1 | taux1 | nominal1 | isBad |
      | SG   | -5        | 0.003 | 1500     | True  |
      | SG   | 8         | -0.01 | 1500     | True  |
      | SG   | 8         | 0.003 | -100     | True  |
```

Il s'agit de faire passer à notre solution un test d'acceptance. Développons un step programme

```python
@given("l'enregistrement d'une obligation avec {nom1}, {maturite1:d}, {taux1:g}, {nominal1:g}")
def step_given_obligation(context, nom1, maturite1, taux1, nominal1):
    context.financier = Financier()
    try:
        obligation = Obligation(
            nom=nom1,
            maturite=maturite1,
            taux=taux1,
            nominal=nominal1
        )
        context.financier.obligation = obligation
        context.financier.boolean = True
    except ValueError:
        context.financier.boolean = False

@when("l'utilisateur la valide")
def step_when_valide(context):
    pass

@when("l'utilisateur valide")
def step_when_valide_short(context):
    pass

@then("le {rendement1:g} est calculé")
def step_then_rendement(context, rendement1):
    assert context.financier.boolean, "L'obligation n'a pas été acceptée"
    rendu = context.financier.obligation.calcul_rendement()
    assert math.isclose(rendu, rendement1, rel_tol=1e-2), (
        f"Attendu {rendement1}, obtenu {rendu}"
    )
```

```python
@then("le système la refuse {isBad:w}")
def step_then_refus(context, isBad):
    attendu = isBad.lower() == "true"
    assert context.financier.boolean != attendu, (
        f"Refus attendu : {attendu}, mais reçu : {not context.financier.boolean}"
    )
```

Il nous suffit d'exécuter notre feature, avec un peu de chance tout sera au vert :)



Reprenons avec de nouvelles features, La deuxieme consiste à calculer la position à un instant t

```gherkin
Feature: US_0002 - Calcul de position à un instant donné

  En tant que financier
  Je veux enregistrer plusieurs obligations dans un portefeuille
  Afin de calculer leur position future

  Scenario: Création d'un portefeuille avec plusieurs obligations
    Given les obligations suivantes sont enregistrées
      | nom | maturite | taux  | nominal |
      | SG  | 8        | 0.003 | 1500    |
      | BNP | 7        | 0.005 | 2000    |
    And le portefeuille est créé pour l'année 2
    When je calcule la position du portefeuille
    Then la position obtenue est 3529.06
```

```python
@given("les obligations suivantes sont enregistrées")
def step_given_obligations(context):
    context.obligations = []
    for row in context.table:
        obl = Obligation(
            nom=row["nom"],
            maturite=int(row["maturite"]),
            taux=float(row["taux"]),
            nominal=float(row["nominal"]),
        )
        context.obligations.append(obl)

@given("le portefeuille est créé pour l'année {annee:d}")
def step_given_portefeuille(context, annee):
    context.portefeuille = Portefeuille(
        obligations=context.obligations,
        annee=annee
    )

@when("je calcule la position du portefeuille")
def step_when_calcul_position(context):
    context.resultat = context.portefeuille.compute_position()

@then("la position obtenue est {valeur:f}")
def step_then_resultat(context, valeur):
    assert math.isclose(context.resultat, valeur, rel_tol=1e-2), (
        f"Attendu {valeur}, obtenu {context.resultat}"
    )
```

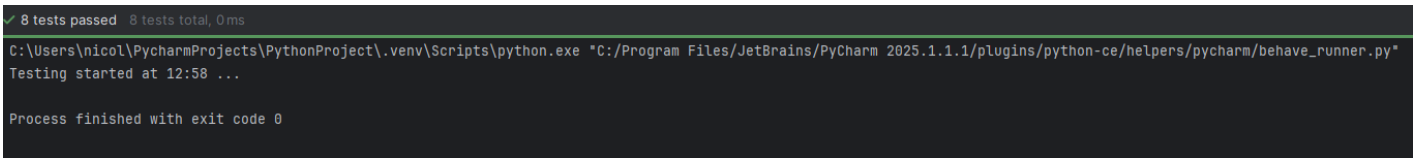
```
✓ 4 tests passed  4 tests total, 0 ms

C:\Users\nicol\PycharmProjects\PythonProject\.venv\Scripts\python.exe "C:/Program Files/JetBrains/PyCharm 2025.1.1.1/plugins/python-ce/helpers/pycharm/behave_runner.py"
Testing started at 12:57 ...

Process finished with exit code 0
```

La troisième à entrainer notre intelligence artificielle

```gherkin
Feature: US_0003 - Entraîner le CoachIA pour sélectionner les meilleures obligations

En tant que financier expérimenté souhaitant maximiser mes investissements,
Je veux qu'un CoachIA m'aide à choisir automatiquement les meilleures obligations en fonction du rendement,
Afin de constituer un portefeuille optimal tout en respectant mes contraintes de capital et de nombre de titres.


  Scenario Outline: Sélection des obligations par rendement décroissant
    Given un CoachIA dispose d'un capital de <capital> euros et souhaite acheter au plus <objectif> obligations
    And les obligations suivantes sont disponibles
      | nom   | maturite | taux    | nominal |
      | <nom1> | <mat1> | <taux1> | <nominal1> |
      | <nom2> | <mat2> | <taux2> | <nominal2> |
      | <nom3> | <mat3> | <taux3> | <nominal3> |
    When le CoachIA sélectionne les obligations
    Then il renvoie <attendu> obligation(s) triée(s) par rendement décroissant

    Examples:
      | capital | objectif | attendu | nom1 | mat1 | taux1 | nominal1 | nom2 | mat2 | taux2 | nominal2 | nom3 | mat3 | taux3 | nominal3 |
      | 4000    | 2        | 2       | SG   | 8    | 0.003 | 1500     | AXA  | 10   | 0.005 | 2000     | CA   | 6    | 0.002 | 1000     |
      | 1500    | 3        | 1       | SG   | 5    | 0.003 | 1500     | AXA  | 10   | 0.005 | 2000     | CA   | 6    | 0.002 | 1000     |
```

```python
@given("un CoachIA dispose d'un capital de {capital:d} euros et souhaite acheter au plus {objectif:d} obligations")
def given_coach_params(context, capital, objectif):
    context.capital = float(capital)
    context.objectif = int(objectif)
    context.obligations = []

@given("les obligations suivantes sont disponibles")
def given_obligations_table(context):
    for row in context.table:
        context.obligations.append(
            Obligation(
                nom=row["nom"],
                maturite=int(row["maturite"]),
                taux=_to_float(row["taux"]),
                nominal=_to_float(row["nominal"]),
            )
        )

@when("le CoachIA sélectionne les obligations")
def when_coach_selects(context):
    coach = CoachIA(context.obligations, context.capital, context.objectif)
    context.selection = coach.entrainer_coach()

@then("il renvoie {attendu:d} obligation(s) triée(s) par rendement décroissant")
def then_verify_selection(context, attendu):
    assert len(context.selection) == attendu, (
        f"Le CoachIA devait renvoyer {attendu} obligation(s), "
        f"mais en a renvoyé {len(context.selection)}."
    )
```

```python
@then("il renvoie {attendu:d} obligation(s) triée(s) par rendement décroissant")
def then_verify_selection(context, attendu):
    assert len(context.selection) == attendu, (
        f"Le CoachIA devait renvoyer {attendu} obligation(s), "
        f"mais en a renvoyé {len(context.selection)}."
    )
    rendements = [obl.rendement for obl in context.selection]
    assert rendements == sorted(rendements, reverse=True), (
        f"Les obligations ne sont pas triées par rendement décroissant : {rendements}"
    )
```


8 tests passed 8 tests total, 0 ms
C:\Users\nicol\PycharmProjects\PythonProject\.venv\Scripts\python.exe "C:/Program Files/JetBrains/PyCharm 2025.1.1.1/plugins/python-ce/helpers/pycharm/behave_runner.py"
Testing started at 12:58 ...
Process finished with exit code 0

On est bon !