Balkis MANSOUR
Gide FORMAT
Hamza RIABI
Abdrahamane Mbourou Camara

# TUTORIEL FINAL

## DU PANIER À LA TRÉSORERIE


## STORYTELLING UNIFIÉ 📖 :

Dans notre projet, deux univers fondamentaux du e-commerce se rencontrent pour former une histoire cohérente et pédagogique.

Le premier est celui du **panier d'achat**.
Il accompagne le client dès son intention d'achat : il parcourt le catalogue, sélectionne des produits, ajuste les quantités, supprime des éléments et observe l'évolution du montant total. Le panier agit comme une mémoire temporaire des décisions prises avant toute validation finale.

Mais au moment de conclure, une question essentielle s'impose naturellement :
**le client a-t-il réellement les moyens de finaliser son achat ?**

C'est ici qu'intervient le second univers : le **compte bancaire**.
Il représente la capacité financière du client et impose des règles simples mais

indispensables : validité des montants, solde suffisant, sécurité des opérations. Aucune transaction ne peut aboutir sans son accord.

Notre projet met ainsi en scène la rencontre de deux logiques complémentaires :

- le **panier d'achat**, qui répond à la question :
  *« Que souhaite acheter le client et pour quel montant ? »*
- le **compte bancaire**, qui répond à :
  *« Cet achat est-il financièrement possible et conforme aux règles ? »*

L'objectif n'est pas de construire une plateforme e-commerce complète, mais de proposer un mini-univers cohérent, centré sur des objets clairement définis, des responsabilités bien séparées et des règles explicites.

Les tests automatisés jouent ici un rôle clé : ils garantissent que chaque interaction respecte le scénario établi et que l'histoire reste logique du début à la fin.

Ainsi, derrière une narration accessible et engageante, le projet illustre des principes fondamentaux de conception logicielle : séparation des responsabilités, validation des règles métier et fiabilité par les tests.

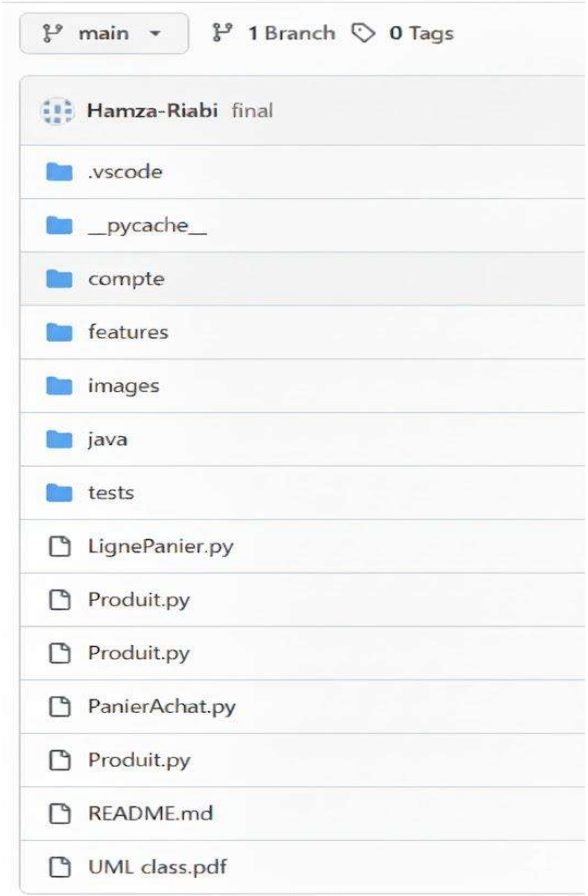
# ETAPE 1 : INTÉGRATION DES DEUX PROJETS DANS UN DÉPÔT UNIQUE

Au départ, nos deux mondes fonctionnaient indépendamment ! D'un côté, le dossier **Panier** développait ses propres routines, et de l'autre, le dossier **Trésorerie** suivait ses propres règles. Mais pour que ces entités puissent collaborer efficacement, il est temps de les rassembler dans un projet unique...tout en préservant l'intégrité de chacun !

- **Harmoniser le dépôt Git.**

Nous avons commencé par rassembler les deux codes dans un projet commun.


- **Résultat de Fusion**




# ETAPE 2: LA COLLABORATION ENTRE LES 2 MONDES

Actuellement, les deux univers fonctionnent de manière indépendante. Nous souhaitons créer une **connexion directe** entre ces mondes, permettant au client de **construire son panier**, de **valider le total de sa commande** et de **simuler le paiement**.

Pour rendre cette étape **pédagogique et interactive**, nous pouvons proposer une action "collaborative" entre le Panier et la Trésorerie :

- **Pré-validation du budget:** vérifier si le total du panier est financièrement réalisable avant toute confirmation.
- **Tentative de débit** (si souhaité) : simuler un paiement pour observer le comportement du système, sans implémenter le paiement réel.

Mais avant tout cela on doit ajouter dans notre classe PanierAchat la méthode suivante :

```
def payer(self, paiement_strategy):
    montant = self.calculer_total()
    paiement_strategy.payer(montant)
```

Ensuite, on doit créer de nouvelles classes

# CB_D

compte > cb_d.py > ...

```python
from compte.compte_bancaire import CompteBancaire


class CB_D(CompteBancaire):

    def retirer(self, montant: float) -> None:
        if self.titulaire is None:
            raise ValueError("aucun titulaire associé au compte")

        if montant <= 0:
            raise ValueError("montant invalide")

        if montant > self.solde:
            raise ValueError("solde insuffisant")

        super().retirer(montant)

```

La classe **`CB_D`** utilise l'héritage pour reprendre automatiquement toutes les fonctionnalités de base de la classe **`CompteBancaire`**, comme la gestion du solde et du titulaire. En héritant de cette classe mère, elle peut ensuite spécialiser son comportement en ajoutant ses propres règles de sécurité lors de l'exécution de la méthode de retrait.

- **Nous avons utilisé le Proxy Pattern.**

  Le Proxy Pattern introduit un intermédiaire entre le client et l'objet réel pour contrôler ou modifier l'accès.

  Dans notre projet : CB_D agit comme un proxy de CompteBancaire, ajoutant des règles de sécurité lors des retraits tout en réutilisant les fonctionnalités de base.

# ETAPE 3: ALLER PLUS LOIN

À long terme, nous ne nous limiterons pas au paiement par compte bancaire. Nous voulons anticiper cette évolution en créant une classe **PaymentStrategy**, conçue pour permettre différentes méthodes de paiement.

Pour l'instant, nous allons créer une sous-classe spécifique qui nous intéresse : **PaymentParCompte**. Cette organisation laisse la porte ouverte à de futures améliorations et à l'ajout de nouvelles stratégies de paiement.

```python
from abc import ABC, abstractmethod

class PaiementStrategy(ABC):

    @abstractmethod
    def payer(self, montant):
        pass
```

```python
from PaiementStrategy import PaiementStrategy

class PaiementParCompte(PaiementStrategy):

    def __init__(self, compte_bancaire):
        self.compte_bancaire = compte_bancaire

    def payer(self, montant):
        self.compte_bancaire.retirer(montant)
```

- **Classe abstraite PaymentStrategy** : définit l'interface commune pour toutes les méthodes de paiement.
- **Sous-classe PaymentParCompte** : implémente la stratégie spécifique de paiement par compte bancaire.

Cette organisation permet :

- D'ajouter facilement de nouvelles méthodes de paiement (`PaymentParCarte`, `PaymentParPaypal`, etc.)

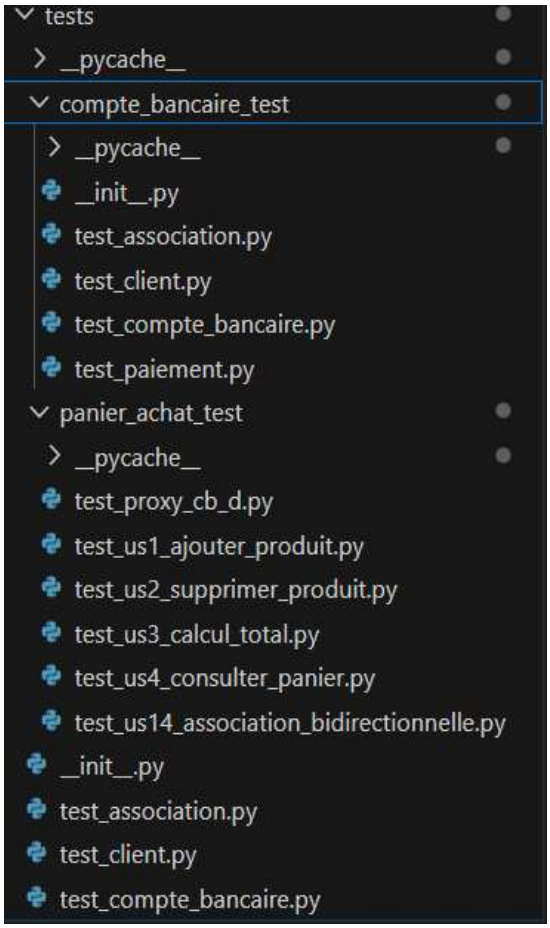
- De changer la stratégie de paiement sans modifier le code du Panier
- De respecter le principe **“ouvert à l’extension, fermé à la modification”**

- **Nous avons utilisé le Strategy Pattern.**

Le **Strategy Pattern** permet de **définir plusieurs comportements ou algorithmes, de les encapsuler dans des classes séparées et de les rendre interchangeables**.
Ici, chaque méthode de paiement (comme `PaymentParCompte`) devient une **stratégie** que le Panier peut utiliser sans connaître ses détails internes.

# ETAPE 4 : PASSER AUX TESTS UNITAIRES ET ASSOCIATIFS

**l’exécution des tests unitaires** avec `pytest`.

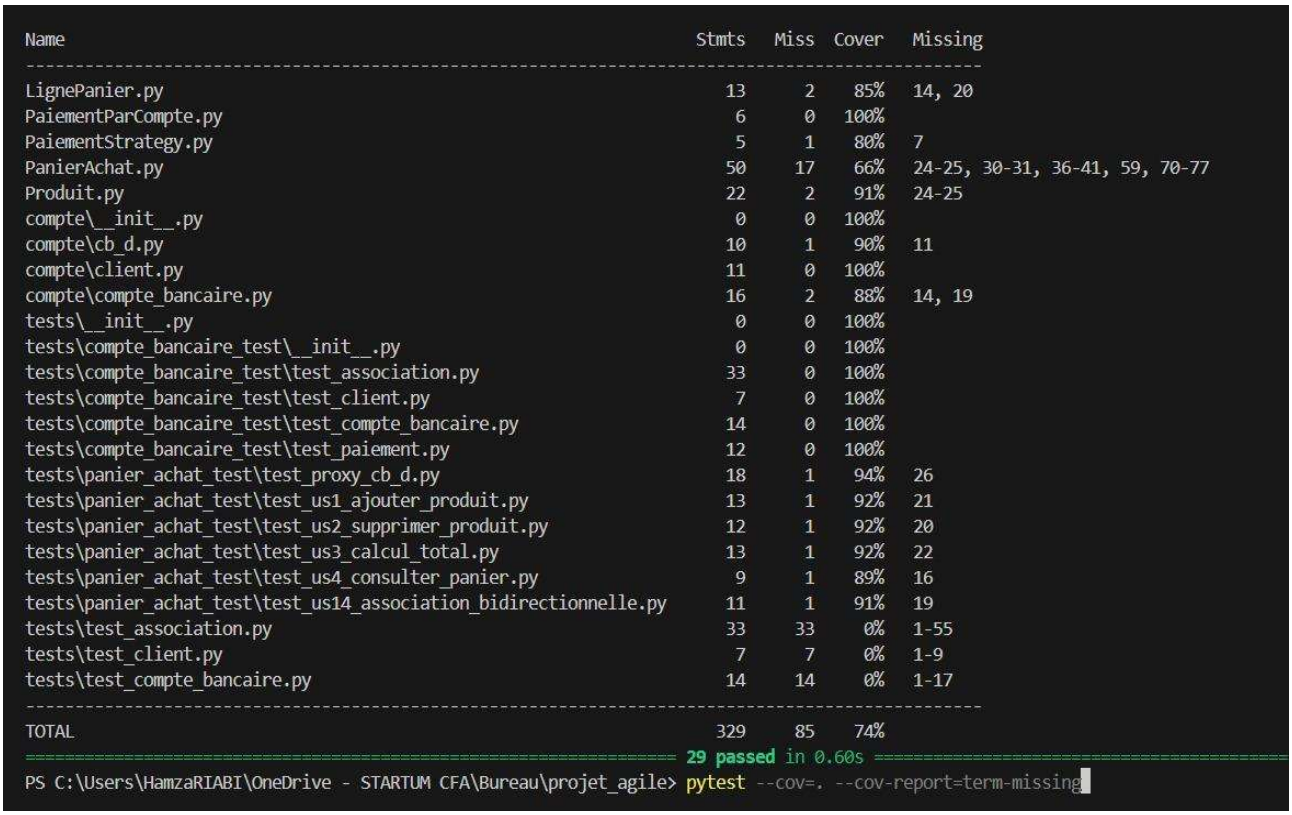


```
tests\compte_bancaire_test\test_association.py ......                                    [ 20%]
tests\compte_bancaire_test\test_client.py .                                              [ 24%]
tests\compte_bancaire_test\test_compte_bancaire.py ...                                   [ 34%]
tests\compte_bancaire_test\test_paiement.py .                                            [ 37%]
tests\panier_achat_test\test_proxy_cb_d.py ...                                           [ 48%]
tests\panier_achat_test\test_us14_association_bidirectionnelle.py .                      [ 51%]
tests\panier_achat_test\test_us1_ajouter_produit.py .                                    [ 55%]
tests\panier_achat_test\test_us2_supprimer_produit.py .                                  [ 58%]
tests\panier_achat_test\test_us3_calcul_total.py .                                       [ 62%]
tests\panier_achat_test\test_us4_consulter_panier.py .                                   [ 65%]
tests\test_association.py ......                                                         [ 86%]
tests\panier_achat_test\test_proxy_cb_d.py ...                                           [ 48%]
tests\panier_achat_test\test_us14_association_bidirectionnelle.py .                      [ 51%]
tests\panier_achat_test\test_us1_ajouter_produit.py .                                    [ 55%]
tests\panier_achat_test\test_us2_supprimer_produit.py .                                  [ 58%]
tests\panier_achat_test\test_us3_calcul_total.py .                                       [ 62%]
tests\panier_achat_test\test_proxy_cb_d.py ...                                           [ 48%]
tests\panier_achat_test\test_us14_association_bidirectionnelle.py .                      [ 51%]
tests\panier_achat_test\test_us1_ajouter_produit.py .                                    [ 55%]
tests\panier_achat_test\test_proxy_cb_d.py ...                                           [ 48%]
tests\panier_achat_test\test_us14_association_bidirectionnelle.py .                      [ 51%]
tests\panier_achat_test\test_proxy_cb_d.py ...                                           [ 48%]
tests\panier_achat_test\test_proxy_cb_d.py ...                                           [ 48%]
tests\panier_achat_test\test_proxy_cb_d.py ...                                           [ 48%]
tests\panier_achat_test\test_us14_association_bidirectionnelle.py .                      [ 51%]
tests\panier_achat_test\test_us1_ajouter_produit.py .                                    [ 55%]
tests\panier_achat_test\test_us2_supprimer_produit.py .                                  [ 58%]
tests\panier_achat_test\test_us3_calcul_total.py .                                       [ 62%]
tests\panier_achat_test\test_us4_consulter_panier.py .                                   [ 65%]
tests\panier_achat_test\test_us14_association_bidirectionnelle.py .                      [ 51%]
tests\panier_achat_test\test_us1_ajouter_produit.py .                                    [ 55%]
tests\panier_achat_test\test_us2_supprimer_produit.py .                                  [ 58%]
tests\panier_achat_test\test_us3_calcul_total.py .                                       [ 62%]
tests\panier_achat_test\test_us4_consulter_panier.py .                                   [ 65%]
tests\panier_achat_test\test_us1_ajouter_produit.py .                                    [ 55%]
tests\panier_achat_test\test_us2_supprimer_produit.py .                                  [ 58%]
tests\panier_achat_test\test_us3_calcul_total.py .                                       [ 62%]
tests\panier_achat_test\test_us4_consulter_panier.py .                                   [ 65%]
tests\panier_achat_test\test_us3_calcul_total.py .                                       [ 62%]
tests\panier_achat_test\test_us4_consulter_panier.py .                                   [ 65%]
```

```
tests\panier_achat_test\test_us4_consulter_panier.py .                                   [ 65%]
tests\panier_achat_test\test_us4_consulter_panier.py .                                   [ 65%]
tests\test_association.py ......                                                         [ 86%]
tests\test_client.py .                                                                   [ 89%]
tests\test_compte_bancaire.py ...                                                        [100%]

========================================== 29 passed in 0.16s ==========================================
PS C:\Users\manso\Downloads\projet_agile>
```

# ETAPE 5 : COUVERTURE DU CODE



```
Name                                                                    Stmts   Miss  Cover   Missing
-----------------------------------------------------------------------------------------------------
LignePanier.py                                                             13      2    85%   14, 20
PaiementParCompte.py                                                        6      0   100%
PaiementStrategy.py                                                         5      1    80%   7
PanierAchat.py                                                             50     17    66%   24-25, 30-31, 36-41, 59, 70-77
Produit.py                                                                 22      2    91%   24-25
compte\__init__.py                                                          0      0   100%
compte\cb_d.py                                                             10      1    90%   11
compte\client.py                                                           11      0   100%
compte\compte_bancaire.py                                                  16      2    88%   14, 19
tests\__init__.py                                                           0      0   100%
tests\compte_bancaire_test\__init__.py                                      0      0   100%
tests\compte_bancaire_test\test_association.py                             33      0   100%
tests\compte_bancaire_test\test_client.py                                   7      0   100%
tests\compte_bancaire_test\test_compte_bancaire.py                         14      0   100%
tests\compte_bancaire_test\test_paiement.py                                12      0   100%
tests\panier_achat_test\test_proxy_cb_d.py                                 18      1    94%   26
tests\panier_achat_test\test_us1_ajouter_produit.py                        13      1    92%   21
tests\panier_achat_test\test_us2_supprimer_produit.py                      12      1    92%   20
tests\panier_achat_test\test_us3_calcul_total.py                           13      1    92%   22
tests\panier_achat_test\test_us4_consulter_panier.py                        9      1    89%   16
tests\panier_achat_test\test_us14_association_bidirectionnelle.py          11      1    91%   19
tests\test_association.py                                                  33     33     0%   1-55
tests\test_client.py                                                        7      7     0%   1-9
tests\test_compte_bancaire.py                                              14     14     0%   1-17
-----------------------------------------------------------------------------------------------------
TOTAL                                                                     329     85    74%
============================== 29 passed in 0.60s ==============================
PS C:\Users\HamzaRIABI\OneDrive - STARTUM CFA\Bureau\projet_agile> pytest --cov=. --cov-report=term-missing
```

- La plupart des tests unitaires et scénarios “user stories” sont exécutés et passent ✅
- Les fichiers PaiementParCompte.py et client.py sont entièrement couverts.
- La couverture globale de **74%** est un bon point de départ pour un projet pédagogique.

## 💡 Points à améliorer

- Tester les fichiers avec **0% de couverture** pour sécuriser toutes les interactions importantes.
- Compléter les tests de PanierAchat.py, qui est central et a beaucoup de code non testé.
- Eventuellement ajouter des tests pour les cas **d’intégration / association** entre Panier et Compte Bancaire.