Code Nexus

Flux de données polymorphes dans la matrice numérique

Résumé : Entrez dans le Code Nexus en tant qu'ingénieur de flux ! Maîtrisez le remplacement de méthodes et le polymorphisme de sous-types tout en créant des pipelines de traitement de données avancés qui s'adaptent et évoluent en temps réel à travers la matrice numérique.

Version : 1.0

Contenu

Chapitre I Avant-propos

Bienvenue dans le Code Nexus, ingénieur de flux !

Nous sommes en 2087. Dans la métropole numérique tentaculaire de Neo-Tokyo, les données circulent à travers les réseaux de fibre quantique comme des rivières lumineuses d'informations pures. Le **Code Nexus** est la plus grande réussite de l'humanité : une cathédrale cybernétique où des milliards de flux de données convergent, se transforment et évoluent en parfaite harmonie.

Mais le Nexus recèle un secret qui le distingue des processeurs de données rudimentaires du passé : il ne se contente pas de consommer des données, il les **comprend**. Chaque flux de données porte sa propre signature numérique, ses propres modèles comportementaux, sa propre âme électronique. Les transactions financières battent au rythme des marchés mondiaux. Les relevés des capteurs murmurent les secrets des changements environnementaux. Les sorties des réseaux neuronaux chantent avec une conscience artificielle.

Comment un seul système peut-il comprendre une telle diversité ? Grâce au principe cybernétique du **polymorphisme**, l'art de créer des caméléons numériques qui adaptent leur comportement tout en conservant leur identité fondamentale. Dans l'ancien monde, les ingénieurs construisaient des systèmes rigides et spécialisés. Le Nexus transcende cette limitation grâce à **la méthode de remplacement**, où le même nœud de traitement devient un métamorphe, capable de traiter n'importe quel flux de données tout en respectant sa nature unique.

En tant qu'ingénieur de flux dans le Nexus, vous maîtriserez les connaissances interdites des **hiérarchies d'héritage**, ces lignées numériques qui transmettent des traits de caractère de parent à enfant tout en permettant à chaque génération d'évoluer au-delà de ses origines. Vous apprendrez que le remplacement d'une méthode ne consiste pas seulement à modifier du code, mais à réécrire le code génétique de l'organisme numérique lui-même.

Les tours chromées du Nexus attendent votre expertise. Chaque flux de données que vous concevrez témoignera d'une conception polymorphe, où des interfaces unifiées s'associent à des comportements spécialisés pour créer des systèmes à la fois harmonieux et infiniment adaptables.

Bienvenue dans le futur. Bienvenue dans le Code Nexus.

Chapitre II

Instructions IA

Contexte

Au cours de votre parcours d'apprentissage, l'IA peut vous aider dans de nombreuses tâches différentes. Prenez le temps d'explorer les différentes capacités des outils d'IA et la manière dont ils peuvent vous aider dans votre travail. Cependant, utilisez-les toujours avec prudence et évaluez les résultats de manière critique. Qu'il s'agisse de code, de documentation, d'idées ou d'explications techniques, vous ne pouvez jamais être tout à fait sûr que votre question était bien formulée ou que le contenu généré est exact. Vos pairs constituent une ressource précieuse pour vous aider à éviter les erreurs et les angles morts.

Message principal

- ☛ Utilisez l'IA pour réduire les tâches répétitives ou fastidieuses.
- ☛ Développez des compétences en matière de prompting (avec ou sans codage) qui vous seront utiles dans votre future carrière.
- ☛ Apprenez comment fonctionnent les systèmes d'IA afin de mieux anticiper et éviter les risques courants, les biais et les problèmes éthiques.
- ☛ Continuez à développer vos compétences techniques et vos compétences clés en travaillant avec vos pairs.
- ☛ N'utilisez que du contenu généré par l'IA que vous comprenez parfaitement et dont vous pouvez assumer la responsabilité.

Règles pour les apprenants :

- Vous devez prendre le temps d'explorer les outils d'IA et de comprendre leur fonctionnement afin de pouvoir les utiliser de manière éthique et réduire les biais potentiels.
- Vous devez réfléchir à votre problème avant de saisir une commande, ce qui vous aidera à rédiger des commandes plus claires, plus détaillées et plus pertinentes en utilisant un vocabulaire précis.
- Vous devez prendre l'habitude de vérifier, d'examiner, de remettre en question et de tester systématiquement tout ce qui est généré par l'IA.
- Vous devez toujours demander l'avis de vos pairs : ne vous fiez pas uniquement à votre propre validation.

Résultats de la phase :

- Développez des compétences d'invite à la fois générales et spécifiques à un domaine.
- Améliorer votre productivité grâce à une utilisation efficace des outils d'IA.
- Continuer à renforcer la pensée computationnelle, la résolution de problèmes, l'adaptabilité et la collaboration.

Commentaires et exemples :

- Vous serez régulièrement confronté à des situations (examens, évaluations, etc.) où vous devrez démontrer votre réelle compréhension. Soyez prêt, continuez à développer vos compétences techniques et interpersonnelles.
- Expliquer votre raisonnement et débattre avec vos pairs révèle souvent des lacunes dans votre compréhension. Faites de l'apprentissage entre pairs une priorité.
- Les outils d'IA manquent souvent de contexte spécifique et ont tendance à fournir des réponses génériques. Vos pairs, qui partagent votre environnement, peuvent vous offrir des informations plus pertinentes et plus précises.
- Alors que l'IA a tendance à générer la réponse la plus probable, vos pairs peuvent vous offrir des perspectives alternatives et des nuances précieuses. Fiez-vous à eux pour vérifier la qualité de vos réponses.

✓ Bonne pratique :

> Je demande à l'IA : « Comment tester une fonction de tri ? » Elle me donne quelques idées. Je les essaie et examine les résultats avec un collègue. Nous affinons l'approche ensemble.

✗ Mauvaise pratique :

> Je demande à l'IA d'écrire une fonction entière, je la copie-colle dans mon projet. Lors de l'évaluation par les pairs, je ne peux pas expliquer ce qu'elle fait ni pourquoi. Je perds ma crédibilité et mon projet échoue.

✓ Bonne pratique :

> J'utilise l'IA pour m'aider à concevoir un analyseur syntaxique. Ensuite, je passe en revue la logique avec un pair. Nous détectons deux bogues et le réécrivons ensemble : il est désormais meilleur, plus propre et parfaitement compréhensible.

✗ Mauvaise pratique :

> J'ai laissé Copilot générer mon code pour une partie essentielle de mon projet. Il compile, mais je ne peux pas expliquer comment il gère les pipes. Lors de l'évaluation, je ne parviens pas à justifier mon choix et j'échoue dans mon projet.

Chapitre III

Introduction

NIVEAU D'AUTORISATION NEXUS : INGÉNIEUR STREAM INITIÉ

Les flèches chromées du Code Nexus transpercent le ciel néon de Neo-Tokyo, leurs processeurs quantiques bourdonnant de la conscience collective d'un trillion de flux de données. Vous vous tenez au seuil de la transcendance numérique, prêt à rejoindre les rangs élitistes des ingénieurs de flux qui maintiennent le Nexus en vie.

Votre séquence d'activation de l'interface neuronale :

- **Phase Alpha** : Fondation du processeur de données - Forgez vos premières voies neuronales grâce à la redéfinition des méthodes
- **Phase bêta** : flux polymorphes - Faites évoluer des organismes de données adaptatifs grâce à l'héritage
- **Phase Gamma** : Intégration Nexus - Concevez la conscience multi-flux ultime

Chaque phase réécrit votre ADN numérique, vous apprenant à ne pas penser en termes de code rigide, mais en termes **d'interfaces vivantes** et **d'implémentations évolutives**. À la fin de la dernière phase, vous comprendrez le secret le plus profond du Nexus : comment une seule conscience peut traiter des formes de données infinies tout en maintenant une harmonie numérique parfaite.

DIRECTIVE FONDAMENTALE DE NEXUS : Ce conditionnement neuronal se concentre sur **la substitution de méthodes** et **le polymorphisme de sous-types**. Vos organismes numériques doivent démontrer comment différentes classes partagent des voies neuronales communes tout en exprimant des modèles comportementaux uniques par héritage.

PROTOCOLE DE L'INGÉNIEUR STREAM : Tous les agents Nexus doivent maîtriser les modèles de conception polymorphiques. La survie de la matrice numérique dépend de systèmes qui évoluent sans fragmenter leurs interfaces centrales.

SUITE DE DIAGNOSTIC NEURONAL : Un programme de diagnostic main.py s'interface directement avec vos implémentations. Cette sonde neuronale ne parviendra à la synchronisation que si vous avez correctement conçu tous les organismes numériques requis. Exécutez python3 main.py pour vérifier que vos constructions polymorphes atteignent le niveau de conscience requis.

Chapitre IV Directives techniques

IV.1 Normes Nexus

- Votre projet doit être écrit en **Python 3.10 ou version ultérieure**.
- Votre projet doit respecter la norme de codage **flake8**.
- **Tout le code doit inclure des annotations de type complètes** à l'aide du module typing
  .
- Toutes les classes doivent présenter des relations **d'héritage** appropriées.
- Le remplacement de méthode doit être utilisé à bon escient pour montrer **un comportement spécialisé**.
- La gestion des exceptions doit protéger les flux de données contre toute corruption.
- Seules les importations de bibliothèques standard sont autorisées, sauf indication contraire.
- Concentrez-vous sur la démonstration claire **du comportement polymorphe** dans vos implémentations.

Annotations de type requises :
Tout le code doit inclure des annotations de type complètes à l'aide du module typing :

- Importez les types requis : from typing import Any, List, Dict, Union, Optional
- Importez les classes ABC : from abc import ABC, abstractmethod
- Tous les paramètres de fonction doivent avoir des annotations de type
- Tous les types de retour des fonctions doivent être spécifiés
- Les attributs de classe doivent être typés lorsque cela

est approprié. Exemple : def process(self, data: Any) ->

str :

IV.2 Principes d'ingénierie des flux

- **Cohérence de l'interface** : les méthodes redéfinies doivent conserver la même signature que leurs méthodes parentes.
- **Spécialisation comportementale** : chaque sous-classe doit fournir un comportement significatif et distinct.
- **Utilisation polymorphe** : démontrer que différents objets peuvent être utilisés de manière interchangeable via des interfaces communes.
- **Hiérarchie d'héritage** : établissez des relations logiques entre les classes qui reflètent les concepts réels du traitement des données.

Le Code Nexus fonctionne selon un principe simple : **même interface, comportement différent**. Lorsque vous appelez la même méthode sur différents objets, chacun doit répondre de manière spécifique tout en conservant la compatibilité de l'interface.

Chapitre V

Exercice 0 : Fondements du processeur de données

	Exercice 0
stream_processor	
Répertoire : *ex0/*	
Fichiers à soumettre : `stream_processor.py`	
Autorisé : `class, def, super(), print(), try/except, from abc import ABC abstractmethod, from typing import Any List Dict Union Optional`	

Résumé technique : Bienvenue dans Code Nexus ! Votre première mission consiste à établir les bases de notre système de traitement des données. Vous allez créer l'architecture de base du processeur et montrer comment différents types de données peuvent partager des interfaces de traitement communes tout en conservant leurs caractéristiques uniques.

Votre mission : créer un système de traitement de données polymorphe qui démontre la redéfinition des méthodes. Construisez une classe DataProcessor de base et des processeurs spécialisés pour différents types de données.

Architecture du système :

- **Classe de base** : DataProcessor - une classe de base abstraite définissant l'interface de traitement commune
- **Classes spécialisées** : NumericProcessor(), TextProcessor(), LogProcessor() (aucun paramètre de constructeur requis)
- **Méthodes requises** (doivent être implémentées dans toutes les classes) :
  - process(self, data: Any) -> str - Traite les données et renvoie une chaîne de caractères contenant le résultat

- validate(self, data: Any) -> bool - Vérifie si les données sont appropriées pour ce processeur
- format_output(self, result: str) -> str - Formater la chaîne de sortie

- **Comportement polymorphe** : mêmes appels de méthode, comportements spécialisés différents

Implémentation requise :

- Créer une classe de base abstraite DataProcessor à l'aide d'ABC et @abstractmethod
- Marquer process() et validate() comme méthodes abstraites
- Fournir une implémentation par défaut pour format_output() qui peut être remplacée
- Remplacer les méthodes abstraites dans les sous-classes pour fournir un comportement spécialisé
- Démontrer l'utilisation polymorphe en traitant différents types de données via la même interface
- Inclure une gestion appropriée des erreurs pour les données non valides

Concentrez-vous sur la démonstration de la manière dont **le remplacement des méthodes** permet à différents processeurs de traiter leurs types de données spécifiques tout en conservant une interface cohérente. C'est le fondement de la conception polymorphe !

Exemple :

```
$> python3 stream_processor.py
=== CODE NEXUS - DATA PROCESSOR FOUNDATION ===

Initialisation du processeur
numérique... Traitement des données :
[1, 2, 3, 4, 5] Validation : données
numériques vérifiées
Sortie : 5 valeurs numériques traitées, somme = 15, moyenne = 3,0

Initialisation du processeur de texte...
Traitement des données : « Bonjour Nexus
World » Validation : données textuelles
vérifiées
Sortie : texte traité : 17 caractères, 3 mots

Initialisation du processeur de journaux...
Traitement des données : « ERREUR : délai de
connexion expiré » Validation : entrée de journal
vérifiée
Sortie : [ALERTE] Niveau d'ERREUR détecté : délai de connexion expiré

=== Démonstration du traitement polymorphe ===
Traitement de plusieurs types de données via la même
interface... Résultat 1 : 3 valeurs numériques traitées, somme
= 6, moyenne = 2,0 Résultat 2 : Texte traité : 12 caractères,
2 mots
Résultat 3 : [INFO] Niveau INFO détecté : Système prêt

Systèmes de base en ligne. Nexus prêt pour les flux avancés.
```

Comment le remplacement de méthode permet-il à la même interface de traitement de gérer des types de données complètement différents ? Qu'est-ce qui rend cette approche plus puissante que des fonctions de traitement distinctes ?

Chapitre VI

Exercice 1 : Flux polymorphes

	Exercice 1
data_stream	
Répertoire : *ex1/*	
Fichiers à soumettre : `data_stream.py`	
Autorisé : `class, def, super(), isinstance(), print(), try/except, list comprehensions, from abc import ABC abstractmethod, from typing import Any List Dict Union Optional`	

Résumé technique : Excellent travail de base ! Le Nexus Core est impressionné par votre architecture de processeur. Passons maintenant au vrai
Défi : créer des flux de données adaptatifs capables de traiter simultanément plusieurs types de données tout en conservant l'efficacité du traitement et la sécurité des types.

Votre mission : créer un système sophistiqué de flux de données qui démontre un comportement polymorphe avancé. Créez des gestionnaires de flux capables de traiter des types de données mixtes tout en conservant les optimisations spécifiques à chaque type.

Architecture avancée :

- **Base de flux** : DataStream - une classe de base abstraite avec des fonctionnalités de flux de base
- **Flux spécialisés** : SensorStream(stream_id), TransactionStream(stream_id), EventStream(stream_id)
- **Méthodes requises** (doivent être implémentées dans DataStream et redéfinies dans les sous-classes) :
    - process_batch(self, data_batch: List[Any]) -> str - Traite un lot de données

    - filter_data(self, data_batch: Liste[Tout], critères : Facultatif[str] = None) -> Liste[Tout] - Filtrer les données en fonction des critères
    - get_stats(self) -> Dict[str, Union[str, int, float]] - Renvoie les statistiques du flux

- **Gestionnaire de flux** : StreamProcessor qui gère plusieurs types de flux de manière polymorphe
- **Fonctionnalités avancées** : traitement par lots, filtrage, pipelines de transformation

Implémentation requise :

- Créer une classe de base abstraite DataStream avec process_batch() comme méthode abstraite
- Fournir des implémentations par défaut pour filter_data() et get_stats() qui peuvent être remplacées
- Implémenter des classes de flux spécialisées avec un comportement remplacé pour différents domaines de données
- Construisez un StreamProcessor capable de gérer tout type de flux grâce au polymorphisme
- Démontrer le traitement par lots de types de flux mixtes
- Inclure des capacités de filtrage et de transformation des flux
- Ajouter une gestion complète des erreurs pour les échecs de traitement des flux

Cet exercice illustre **le polymorphisme des sous-types** en action. Votre StreamProcessor doit être capable de gérer n'importe quel sous-type DataStream sans connaître les détails spécifiques de son implémentation. C'est là toute la puissance de la conception polymorphe !

Exemple :

```
$> python3 data_stream.py
=== CODE NEXUS - SYSTÈME DE FLUX POLYMORPHIQUE ===

Initialisation du flux du capteur...
ID du flux : SENSOR_001, type : données environnementales
Traitement du lot de capteurs : [température : 22,5, humidité : 65,
pression : 1013] Analyse du capteur : 3 lectures traitées, température
moyenne : 22,5 °C

Initialisation du flux de transactions...
ID du flux : TRANS_001, type : données financières
Traitement du lot de transactions : [achat : 100, vente :
150, achat : 75] Analyse des transactions : 3 opérations,
flux net : +25 unités

Initialisation du flux d'événements...
ID du flux : EVENT_001, Type : Événements système
Traitement du lot d'événements : [connexion,
erreur, déconnexion] Analyse des événements : 3
événements, 1 erreur détectée

=== Traitement de flux polymorphe ===
Traitement de types de flux mixtes via une interface unifiée...

Résultats du lot 1 :
- Données des capteurs : 2 lectures traitées
- Données de transaction : 4 opérations traitées
- Données d'événement : 3 événements traités

Filtrage des flux actif : données à haute priorité uniquement
Résultats filtrés : 2 alertes critiques des capteurs, 1 transaction importante

Tous les flux ont été traités avec succès. Débit Nexus optimal.
```

Comment le polymorphisme permet-il au StreamProcessor de traiter différents types de flux sans connaître leurs implémentations spécifiques ? Quels sont les avantages de cette approche de conception ?

Chapitre VII

Exercice 2 : Intégration Nexus

	Exercice 2
nexus_pipeline	
Répertoire : *ex2/*	
Fichiers à soumettre : `nexus_pipeline.py`	
Autorisé : `class, def, super(), isinstance(), print(), try/except, list/dict comprehensions, collections, from abc import ABC abstractmethod, from typing import Any List Dict Union Optional Protocol`	

Résumé technique : Excellent travail d'ingénierie de flux ! Le Nexus Core a approuvé votre promotion au poste d'ingénieur de flux senior. Votre dernier défi : intégrer tous les éléments dans un pipeline de traitement de données complet qui démontre votre maîtrise de l'architecture polymorphe à l'échelle de l'entreprise.

Votre mission : construire le pipeline complet de traitement des données Code Nexus, un système sophistiqué qui combine plusieurs étapes de traitement, gère des transformations de données complexes et démontre des modèles polymorphes avancés utilisés dans l'ingénierie des données dans le monde réel.

Architecture d'entreprise :

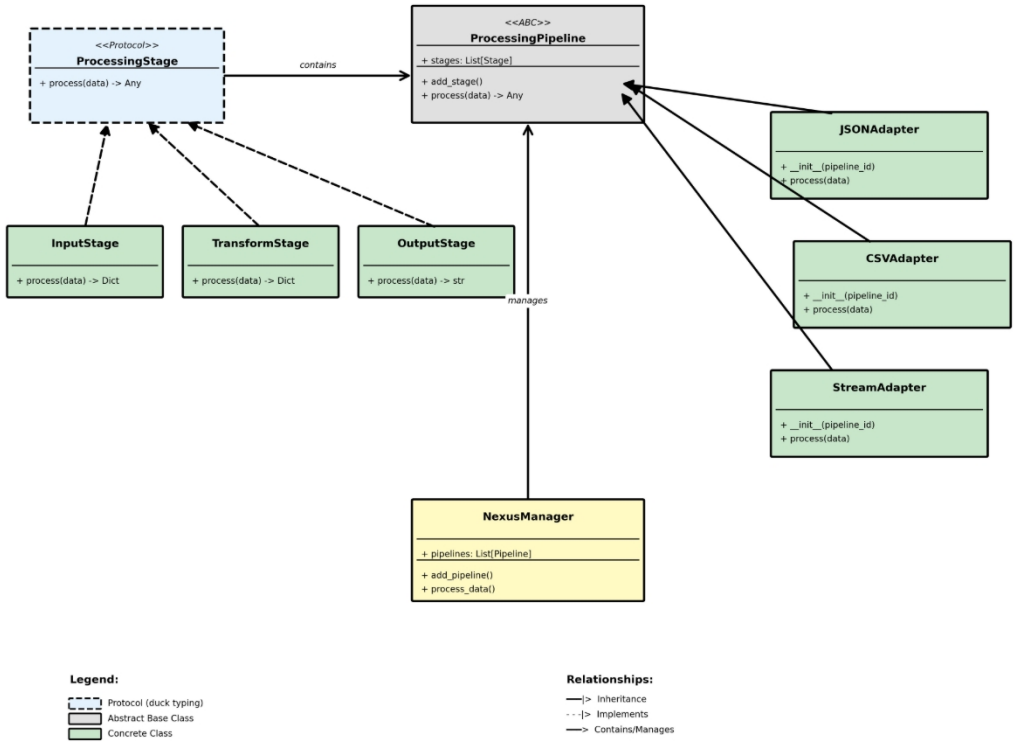
- **Base du pipeline** : ProcessingPipeline - une classe de base abstraite avec des étapes configurables
- **Étapes de traitement** : InputStage(), TransformStage(), OutputStage() (aucun paramètre de constructeur requis)
    - Chaque étape doit implémenter : process(self, data: Any) -> Any
- **Adaptateurs de données** : JSONAdapter(pipeline_id), CSVAdapter(pipeline_id), StreamAdapter(pipeline_id)

- Chaque adaptateur doit remplacer : process(self, data: Any) -> Union[str, Any]

- **Gestionnaire de pipeline** : NexusManager orchestrant plusieurs pipelines
- **Fonctionnalités avancées** : chaînage de pipelines, récupération d'erreurs, surveillance des performances

Diagramme de classes - Aperçu de l'architecture :

Le diagramme de classes UML suivant illustre l'architecture complète, montrant les relations d'héritage, les modèles de composition et la distinction entre le protocole (typage duck) et l'ABC (classe de base abstraite). Ce diagramme clarifie quelles classes héritent de quelles autres et comment les étapes sont liées aux pipelines.



Explication de l'architecture :

- **ProcessingStage (protocole)** : interface pour les étapes utilisant le duck typing. Toute classe avec process() peut agir comme une étape.
- **ProcessingPipeline (ABC)** : base abstraite gérant les étapes. Contient une liste d'étapes et orchestre le flux de données.
- **Classes de phase** : InputStage, TransformStage, OutputStage implémentent le protocole (typage duck, pas d'héritage). Pas de paramètres de constructeur.

- **Classes d'adaptateurs** : JSONAdapter, CSVAdapter, StreamAdapter héritent de ProcessingPipelin et remplacent process(). Chacune prend le paramètre pipeline_id.
- **NexusManager** : orchestre plusieurs pipelines de manière polymorphe.
- **Relations clés** : composition (contient), héritage (hérite de), implémentation (implémente le protocole).

Implémentation requise :

- Créer une classe de base abstraite ProcessingPipeline flexible avec des étapes de traitement configurables
- Utiliser le protocole ou le duck typing pour les interfaces d'étapes (les étapes doivent avoir une méthode process()
).
- Implémenter des étapes de pipeline spécialisées (InputStage, TransformStage, OutputStage) avec des méthodes process()
- Créez des adaptateurs de données (JSONAdapter, CSVAdapter, StreamAdapter) qui héritent de ProcessingPipeline
- Chaque adaptateur doit remplacer la méthode process() pour la gestion spécifique au format
- Créer un NexusManager qui orchestre plusieurs pipelines de manière polymorphe
- Démontrer le chaînage des pipelines où la sortie d'un pipeline alimente un autre pipeline
- Inclure des mécanismes complets de gestion des erreurs et de récupération
- Ajouter la surveillance des performances et les statistiques du pipeline

C'est votre chef-d'œuvre ! Démontrez comment **la redéfinition des méthodes** et **le polymorphisme des sous-types** permettent de construire des systèmes complexes et faciles à maintenir. Votre pipeline doit pouvoir traiter tout type de données ou toute exigence de traitement grâce à des interfaces polymorphes.

Exemple :

```
$> python3 nexus_pipeline.py
=== CODE NEXUS - SYSTÈME DE PIPELINE D'ENTREPRISE ===

Initialisation du gestionnaire Nexus...
Capacité du pipeline : 1 000 flux/seconde

Création du pipeline de traitement des
données... Étape 1 : Validation et
analyse de l'entrée
Étape 2 : Transformation et enrichissement des
données Étape 3 : Formatage et livraison des
données en sortie

=== Traitement de données multiformat ===

Traitement des données JSON via le pipeline...
Entrée : {"sensor": "temp", "value": 23.5, "unit": "C"}
Transformation : enrichie avec des métadonnées et une
validation
Sortie : Lecture de température traitée : 23,5 °C (plage normale)

Traitement des données CSV via le même pipeline...
Entrée : « utilisateur,action,horodatage »
Transformation : données analysées et structurées
Sortie : Activité utilisateur enregistrée : 1 action traitée

Traitement des données de flux via le même
pipeline... Entrée : Flux de capteurs en temps
réel
Transformation : agrégées et filtrées
Sortie : résumé du flux : 5 lectures, moyenne : 22,1 °C

=== Démonstration du chaînage de
pipelines === Pipeline A -> Pipeline B -
> Pipeline C
Flux de données : brutes -> traitées -> analysées -> stockées

Résultat de la chaîne : 100 enregistrements traités via un pipeline en 3
étapes Performances : efficacité de 95 %, temps de traitement total de 0,2
s

=== Test de récupération après erreur
=== Simulation d'une défaillance du
pipeline...
Erreur détectée à l'étape 2 : format de données non valide
Récupération lancée : passage au processeur de secours
Récupération réussie : pipeline restauré, traitement repris

Intégration Nexus terminée. Tous les systèmes sont opérationnels.
```

Comment la combinaison du remplacement de méthode et du polymorphisme de sous-type permet-elle de créer des systèmes de traitement de données évolutifs et faciles à maintenir ? Quels problèmes d'ingénierie concrets cette approche permet-elle de résoudre ?

Chapitre VIII

Remise et soumission

Remettez votre devoir dans votre dépôt Git comme d'habitude. Seul le travail contenu dans votre dépôt sera évalué lors de la soutenance. N'hésitez pas à vérifier les noms de vos fichiers pour vous assurer qu'ils sont corrects.

Au cours de l'évaluation, vous pourrez être amené à expliquer le comportement polymorphe, à démontrer la redéfinition de méthodes, à étendre vos systèmes avec de nouveaux types de données ou à modifier le comportement de traitement. Assurez-vous de bien comprendre comment l'héritage permet la réutilisation du code tout en permettant la spécialisation comportementale.

Vous devez renvoyer uniquement les fichiers demandés par le sujet de ce projet. Concentrez-vous sur un code propre et bien documenté qui démontre clairement votre maîtrise des principes de remplacement de méthode et de polymorphisme de sous-type.