SYSTÈME DE GESTION DE STOCK MULTI-ENTREPÔTS

DOCUMENTATION COMPLÈTE - DIAGRAMMES UML & CAHIER DES CHARGES

PARTIE 1: RÉSUMÉ DE TOUS LES CODES DU PROJET

📁 ***STRUCTURE DU PROJET:***

```
java-multi-entrepots-stock/
|
├── README.md
├── pom.xml
├── STRUCTURE_PROJET_COMPLETE.txt
|
├── src/
|   └── main/java/com/project/stock/
|       ├── models/
|       |   ├── Produit.java
|       |   ├── Entrepot.java
|       |   ├── MouvementStock.java
|       |
|       ├── services/
|       |   ├── ProduitService.java
|       |   ├── EntrepotService.java
|       |   ├── StockService.java
|       |
|       ├── repository/
|       |   ├── ProduitRepository.java
|       |   ├── EntrepotRepository.java
|       |   ├── MouvementRepository.java
|       |
|       ├── utils/
|       |   ├── FileHelper.java
|       |   ├── JsonHelper.java
|       |
```

```
|   ├── exceptions/
|   |   ├── EntrepotNotFound.java
|   |   ├── ProduitNotFound.java
|   |
|   ├── Main.java
|
├── data/                                    (créé automatiquement)
|   ├── produits.json                        ✓ (existe)
|   ├── entrepots.json                       ✓ (existe)
|   └── mouvements.json                      (sera créé lors du premier mouvement)
|
└── docs/
    ├── diagramme_de_classpng.png            ✓ (existe)
    ├── diagrame_sequence_ajouter_produit.png ✓ (existe)
    ├── diagramme_sequence_entre_stock.png   ✓ (existe)
    ├── diagramme_sequence_transfert.png     ✓ (existe)
    ├── diagramme_sequence_consulter_stocker_.png ✓ (existe)
    ├── CAHIER_DE_CHARGE.md                  ✓ (existe)
    ├── CAHIER_DE_CHARGE.html                ✓ (existe)
    ├── DIAGRAMMES_ET_CAHIER_DE_CHARGE.txt   ✓ (existe)
    ├── INSTRUCTIONS_PDF.md                  ✓ (existe)
    ├── README_DIAGRAMMES.md                 ✓ (existe)
    └── cahier_de_charge.pdf                 ✓ (existe)
```

DÉTAIL DES CLASSES:

---

1. MODELS (Classes métiers)

📄 Produit.java

- Attributs:

- String code
- String nom
- String description
- double prixUnitaire

- Méthodes principales:
  - Constructeurs (par défaut et avec paramètres)
  - Getters/Setters pour tous les attributs
  - equals(), hashCode(), toString()

Entrepot.java

- Attributs:
  - String code
  - String nom
  - String adresse
  - double capaciteMax
- Méthodes principales:
  - Constructeurs (par défaut et avec paramètres)
  - Getters/Setters pour tous les attributs
  - equals(), hashCode(), toString()

MouvementStock.java

- Attributs:
  - String id
  - String codeProduit
  - String codeEntrepotSource
  - String codeEntrepotDestination
  - TypeMouvement type (ENUM: ENTREE, SORTIE, TRANSFERT)
  - int quantite

- LocalDateTime dateMouvement
- String commentaire

- Méthodes principales:
  - Constructeurs
  - Getters/Setters
  - equals(), hashCode(), toString()

2. SERVICES (Logique métier)

📄 ProduitService.java

- Méthodes:
  - ajouterProduit()
  - modifierProduit()
  - supprimerProduit()
  - listerProduits()
  - trouverProduit()
  - produitExiste()

📄 EntrepotService.java

- Méthodes:
  - ajouterEntrepot()
  - modifierEntrepot()
  - supprimerEntrepot()
  - listerEntrepots()
  - trouverEntrepot()
  - entrepotExiste()

StockService.java

- Méthodes:
  - enregistrerEntree()
  - enregistrerSortie()
  - transfererProduit()
  - getQuantiteTotale()
  - getQuantiteParEntrepot()
  - getHistoriqueMouvements()
  - getMouvementsParProduit()
  - getMouvementsParDate()
  - getMouvementsParProduitEtDate()
  - getStockActuel()
  - calculerStockActuel() (privée)

3. REPOSITORIES (Persistance)

ProduitRepository.java

- Méthodes:
  - loadProduits() (privée)
  - saveProduits() (privée)
  - add()
  - update()

- delete()
- findByCode()
- findAll()
- exists()

EntrepotRepository.java

- Méthodes:
  - loadEntrepots() (privée)
  - saveEntrepots() (privée)
  - add()
  - update()
  - delete()
  - findByCode()
  - findAll()
  - exists()

MouvementRepository.java

- Méthodes:
  - loadMouvements() (privée)
  - saveMouvements() (privée)
  - add()
  - findById()
  - findAll()
  - findByProduit()
  - findByEntrepot()
  - findByDate()
  - findByProduitAndDate()

4. UTILS (Utilitaires)

FileHelper.java

- Méthodes statiques:

  - readFile(String filePath)

  - writeFile(String filePath, String content)

  - fileExists(String filePath)

JsonHelper.java

- Méthodes statiques:

  - toJson(Object object)

  - listToJson(List<?> list)

  - fromJson(String json, Class<T> clazz)

  - listFromJson(String json, TypeReference<List<T>> typeReference)

5. EXCEPTIONS

ProduitNotFound.java

- Exception personnalisée pour produit non trouvé

EntrepotNotFound.java

- Exception personnalisée pour entrepôt non trouvé

6. MAIN.JAVA (Application principale)

📄 Main.java

- Méthodes principales:
  - main() → Point d'entrée
  - afficherMenuPrincipal()
  - menuGestionProduits()
  - menuGestionEntrepots()
  - menuMouvementsStock()
  - menuTransfert()
  - menuConsultation()
  - menuStatistiques()
  - Méthodes CRUD pour produits
  - Méthodes CRUD pour entrepôts
  - Méthodes de consultation
  - Méthodes de statistiques
  - Méthodes utilitaires (lireString, lireEntier, lireDouble, lireDate)

PARTIE 2: CODE PLANTUML - DIAGRAMME DE CLASSES

========== PACKAGE MODELS ==========

package "com.project.stock.models" {

class Produit {

- String code

- String nom
- String description
- double prixUnitaire
+ Produit()
+ Produit(String, String, String, double)
+ getCode() : String
+ setCode(String)
+ getNom() : String
+ setNom(String)
+ getDescription() : String
+ setDescription(String)
+ getPrixUnitaire() : double
+ setPrixUnitaire(double)
+ equals(Object) : boolean
+ hashCode() : int
+ toString() : String
}

class Entrepot {
- String code
- String nom
- String adresse
- double capaciteMax
+ Entrepot()
+ Entrepot(String, String, String, double)
+ getCode() : String
+ setCode(String)

```
        + getNom() : String
        + setNom(String)
        + getAdresse() : String
        + setAdresse(String)
        + getCapaciteMax() : double
        + setCapaciteMax(double)
        + equals(Object) : boolean
        + hashCode() : int
        + toString() : String
    }

class MouvementStock {
        - String id
        - String codeProduit
        - String codeEntrepotSource
        - String codeEntrepotDestination
        - TypeMouvement type
        - int quantite
        - LocalDateTime dateMouvement
        - String commentaire
        + MouvementStock()
        + MouvementStock(String, String, String, String, TypeMouvement, int, String)
        + getId() : String
        + setId(String)
        + getCodeProduit() : String
        + setCodeProduit(String)
        + getCodeEntrepotSource() : String
```

```
        + setCodeEntrepotSource(String)
        + getCodeEntrepotDestination() : String
        + setCodeEntrepotDestination(String)
        + getType() : TypeMouvement
        + setType(TypeMouvement)
        + getQuantite() : int
        + setQuantite(int)
        + getDateMouvement() : LocalDateTime
        + setDateMouvement(LocalDateTime)
        + getCommentaire() : String
        + setCommentaire(String)
        + equals(Object) : boolean
        + hashCode() : int
        + toString() : String
    }

    enum TypeMouvement {
        ENTREE
        SORTIE
        TRANSFERT
    }

    MouvementStock *-- TypeMouvement
}
```

========== PACKAGE EXCEPTIONS ==========

package "com.project.stock.exceptions" {

class ProduitNotFound {

+ ProduitNotFound(String)

+ ProduitNotFound(String, Throwable)

}

class EntrepotNotFound {

+ EntrepotNotFound(String)

+ EntrepotNotFound(String, Throwable)

}

}

========== PACKAGE REPOSITORY ==========

package "com.project.stock.repository" {

class ProduitRepository {

- List<Produit> produits

- String FILE_PATH

- loadProduits()

- saveProduits()

+ add(Produit)

+ update(Produit)

+ delete(String)

+ findByCode(String) : Produit

+ findAll() : List<Produit>

+ exists(String) : boolean

}

class EntrepotRepository {

- List<Entrepot> entrepots

- String FILE_PATH

- loadEntrepots()

- saveEntrepots()

+ add(Entrepot)

+ update(Entrepot)

+ delete(String)

+ findByCode(String) : Entrepot

+ findAll() : List<Entrepot>

+ exists(String) : boolean

}

class MouvementRepository {

- List<MouvementStock> mouvements

- String FILE_PATH

- loadMouvements()

- saveMouvements()

+ add(MouvementStock)

+ findById(String) : MouvementStock

+ findAll() : List<MouvementStock>

+ findByProduit(String) : List<MouvementStock>

+ findByEntrepot(String) : List<MouvementStock>

+ findByDate(LocalDateTime, LocalDateTime) : List<MouvementStock>

+ findByProduitAndDate(String, LocalDateTime, LocalDateTime) : List<MouvementStock>

}

}

========== PACKAGE SERVICES ==========

package "com.project.stock.services" {

class ProduitService {

- ProduitRepository produitRepository

+ ajouterProduit(String, String, String, double)

+ modifierProduit(String, String, String, double)

+ supprimerProduit(String)

+ listerProduits() : List<Produit>

+ trouverProduit(String) : Produit

+ produitExiste(String) : boolean

}

class EntrepotService {

- EntrepotRepository entrepotRepository

+ ajouterEntrepot(String, String, String, double)

+ modifierEntrepot(String, String, String, double)

+ supprimerEntrepot(String)

+ listerEntrepots() : List<Entrepot>

+ trouverEntrepot(String) : Entrepot

+ entrepotExiste(String) : boolean

}

class StockService {

- MouvementRepository mouvementRepository

- ProduitService produitService

- EntrepotService entrepotService

- Map<String, Map<String, Integer>> stockActuel

- calculerStockActuel()

- ajouterStock(String, String, int)

+ enregistrerEntree(String, String, int, String)

+ enregistrerSortie(String, String, int, String)

+ transfererProduit(String, String, String, int, String)

+ getQuantiteTotale(String) : int

+ getQuantiteParEntrepot(String, String) : int

+ getHistoriqueMouvements() : List<MouvementStock>

+ getMouvementsParProduit(String) : List<MouvementStock>

+ getMouvementsParDate(LocalDateTime, LocalDateTime) : List<MouvementStock>

+ getMouvementsParProduitEtDate(String, LocalDateTime, LocalDateTime) : List<MouvementStock>

+ getStockActuel() : Map<String, Map<String, Integer>>

}

}

========== PACKAGE UTILS ==========

package "com.project.stock.utils" {

class FileHelper {

+ {static} readFile(String) : String

+ {static} writeFile(String, String)

+ {static} fileExists(String) : boolean

}

class JsonHelper {

- {static} ObjectMapper objectMapper

+ {static} toJson(Object) : String

+ {static} listToJson(List<?>) : String

+ {static} fromJson(String, Class<T>) : T

+ {static} listFromJson(String, TypeReference<List<T>>) : List<T>

}

}

========== PACKAGE MAIN ==========

package "com.project.stock" {

class Main {

- {static} Scanner scanner

- {static} ProduitService produitService

- {static} EntrepotService entrepotService

- {static} StockService stockService

- {static} DateTimeFormatter dateFormatter

+ {static} main(String[])

- {static} afficherMenuPrincipal()

- {static} menuGestionProduits()

- {static} menuGestionEntrepots()

- {static} menuMouvementsStock()

- {static} menuTransfert()

- {static} menuConsultation()

- {static} menuStatistiques()

- {static} lireString(String) : String

- {static} lireEntier(String) : int

- {static} lireDouble(String) : double

- {static} lireDate(String) : LocalDateTime

```
    }
}
```

========== RELATIONS ==========

- ProduitService --> ProduitRepository : utilise
- ProduitService --> Produit : manipule
- ProduitService ..> ProduitNotFound : lance

- EntrepotService --> EntrepotRepository : utilise
- EntrepotService --> Entrepot : manipule
- EntrepotService ..> EntrepotNotFound : lance

- StockService --> MouvementRepository : utilise
- StockService --> ProduitService : utilise
- StockService --> EntrepotService : utilise
- StockService --> MouvementStock : manipule
- StockService ..> ProduitNotFound : lance
- StockService ..> EntrepotNotFound : lance

- ProduitRepository --> Produit : persiste
- ProduitRepository --> FileHelper : utilise
- ProduitRepository --> JsonHelper : utilise

- EntrepotRepository --> Entrepot : persiste
- EntrepotRepository --> FileHelper : utilise
- EntrepotRepository --> JsonHelper : utilise

- MouvementRepository --> MouvementStock : persiste
- MouvementRepository --> FileHelper : utilise
- MouvementRepository --> JsonHelper : utilise

- Main --> ProduitService : utilise

- Main --> EntrepotService : utilise
- Main --> StockService : utilise

- MouvementStock --> Produit : référence (codeProduit)
- MouvementStock --> Entrepot : référence (codeEntrepotSource/Destination)

@enduml

PARTIE 3: CODE PLANTUML - DIAGRAMME DE SÉQUENCE

========== DIAGRAMME DE SÉQUENCE 1: AJOUTER UN PRODUIT ==========

```
@startuml Sequence_Ajouter_Produit
    actor Utilisateur
    participant Main
    participant ProduitService
    participant ProduitRepository
    participant FileHelper
    participant JsonHelper
    database "data/produits.json"

    Utilisateur -> Main: Choisir "Ajouter produit"
    Main -> Main: lireString("Code")
    Main -> Main: lireString("Nom")
    Main -> Main: lireString("Description")
    Main -> Main: lireDouble("Prix")
    Main -> ProduitService: ajouterProduit(code, nom, desc, prix)
```

```
    ProduitService -> ProduitRepository: exists(code)
    ProduitRepository -> ProduitRepository: loadProduits()
    ProduitRepository -> FileHelper: readFile("data/produits.json")
    FileHelper -> "data/produits.json": lire fichier
    "data/produits.json" --> FileHelper: contenu JSON
    FileHelper --> ProduitRepository: contenu
    ProduitRepository -> JsonHelper: listFromJson(json)
    JsonHelper --> ProduitRepository: List<Produit>
    ProduitRepository --> ProduitService: false (n'existe pas)
    ProduitService -> Produit: new Produit(code, nom, desc, prix)
    Produit --> ProduitService: produit créé
    ProduitService -> ProduitRepository: add(produit)
    ProduitRepository -> ProduitRepository: produits.add(produit)
    ProduitRepository -> ProduitRepository: saveProduits()
    ProduitRepository -> JsonHelper: listToJson(produits)
    JsonHelper --> ProduitRepository: JSON string
    ProduitRepository -> FileHelper: writeFile("data/produits.json", json)
    FileHelper -> "data/produits.json": écrire fichier
    FileHelper --> ProduitRepository: OK
    ProduitRepository --> ProduitService: OK
    ProduitService --> Main: OK
    Main --> Utilisateur: "✓ Produit ajouté avec succès !"
@enduml

====== DIAGRAMME DE SÉQUENCE 2: ENREGISTRER UNE ENTREE DE STOCK =======
@startuml Sequence_Enregistrer_Entree
    actor Utilisateur
```

```
participant Main
participant StockService
participant ProduitService
participant EntrepotService
participant MouvementRepository
participant FileHelper
participant JsonHelper
database "data/mouvements.json"

Utilisateur -> Main: Choisir "Enregistrer entrée"
Main -> Main: lireString("Code produit")
Main -> Main: lireString("Code entrepôt")
Main -> Main: lireEntier("Quantité")
Main -> StockService: enregistrerEntree(codeProd, codeEntrep, qte, comment)
StockService -> ProduitService: produitExiste(codeProd)
ProduitService -> ProduitRepository: exists(codeProd)
ProduitRepository --> ProduitService: true
ProduitService --> StockService: true
StockService -> EntrepotService: entrepotExiste(codeEntrep)
EntrepotService -> EntrepotRepository: exists(codeEntrep)
EntrepotRepository --> EntrepotService: true
EntrepotService --> StockService: true
StockService -> StockService: UUID.randomUUID()
StockService -> MouvementStock: new MouvementStock(id, codeProd, null,
codeEntrep, ENTREE, qte, comment)
MouvementStock --> StockService: mouvement créé
StockService -> MouvementRepository: add(mouvement)
```

```
    MouvementRepository -> MouvementRepository: mouvements.add(mouvement)
    MouvementRepository -> MouvementRepository: saveMouvements()
    MouvementRepository -> JsonHelper: listToJson(mouvements)
    JsonHelper --> MouvementRepository: JSON string
    MouvementRepository -> FileHelper: writeFile("data/mouvements.json", json)
    FileHelper -> "data/mouvements.json": écrire fichier
    FileHelper --> MouvementRepository: OK
    MouvementRepository --> StockService: OK
    StockService -> StockService: ajouterStock(codeEntrep, codeProd, qte)
    StockService --> Main: OK
    Main --> Utilisateur: "✅ Entrée de stock enregistrée avec succès !"
@enduml

' ========== DIAGRAMME DE SÉQUENCE 3: TRANSFERT ENTRE ENTREPÔTS ==========
@startuml Sequence_Transfert
    actor Utilisateur
    participant Main
    participant StockService
    participant ProduitService
    participant EntrepotService
    participant MouvementRepository
    participant FileHelper
    participant JsonHelper
    database "data/mouvements.json"

    Utilisateur -> Main: Choisir "Transfert entre entrepôts"
    Main -> Main: lireString("Code produit")
```

```
Main -> Main: lireString("Entrepôt source")
Main -> Main: lireString("Entrepôt destination")
Main -> Main: lireEntier("Quantité")
Main -> StockService: transfererProduit(codeProd, codeEntrepSrc, codeEntrepDest, qte, comment)
StockService -> ProduitService: produitExiste(codeProd)
ProduitService --> StockService: true
StockService -> EntrepotService: entrepotExiste(codeEntrepSrc)
EntrepotService --> StockService: true
StockService -> EntrepotService: entrepotExiste(codeEntrepDest)
EntrepotService --> StockService: true
StockService -> StockService: getQuantiteParEntrepot(codeProd, codeEntrepSrc)
StockService -> StockService: vérifier stock disponible
alt Stock suffisant
    StockService -> StockService: UUID.randomUUID()
    StockService -> MouvementStock: new MouvementStock(id, codeProd, codeEntrepSrc, codeEntrepDest, TRANSFERT, qte, comment)
    MouvementStock --> StockService: mouvement créé
    StockService -> MouvementRepository: add(mouvement)
    MouvementRepository -> MouvementRepository: mouvements.add(mouvement)
    MouvementRepository -> MouvementRepository: saveMouvements()
    MouvementRepository -> JsonHelper: listToJson(mouvements)
    JsonHelper --> MouvementRepository: JSON string
    MouvementRepository -> FileHelper: writeFile("data/mouvements.json", json)
    FileHelper -> "data/mouvements.json": écrire fichier
    FileHelper --> MouvementRepository: OK
    MouvementRepository --> StockService: OK
    StockService -> StockService: ajouterStock(codeEntrepSrc, codeProd, -qte)
```

```
        StockService -> StockService: ajouterStock(codeEntrepDest, codeProd, +qte)
        StockService --> Main: OK
        Main --> Utilisateur: "✅ Transfert effectué avec succès !"
    else Stock insuffisant
        StockService --> Main: Exception("Stock insuffisant")
        Main --> Utilisateur: "❌ Erreur: Stock insuffisant"
    end
@enduml

========== DIAGRAMME DE SÉQUENCE 4: CONSULTER LE STOCK ==========
@startuml Sequence_Consulter_Stock
    actor Utilisateur
    participant Main
    participant StockService
    participant MouvementRepository
    participant FileHelper
    participant JsonHelper
    database "data/mouvements.json"

    Utilisateur -> Main: Choisir "Quantité totale par produit"
    Main -> Main: lireString("Code produit")
    Main -> StockService: getQuantiteTotale(codeProd)
    StockService -> StockService: calculerStockActuel()
    StockService -> MouvementRepository: findAll()
    MouvementRepository -> MouvementRepository: loadMouvements()
    MouvementRepository -> FileHelper: readFile("data/mouvements.json")
    FileHelper -> "data/mouvements.json": lire fichier
```

```
"data/mouvements.json" --> FileHelper: contenu JSON
FileHelper --> MouvementRepository: contenu
MouvementRepository -> JsonHelper: listFromJson(json)
JsonHelper --> MouvementRepository: List<MouvementStock>
MouvementRepository --> StockService: List<MouvementStock>
loop Pour chaque mouvement
  StockService -> StockService: calculer stock selon type
  alt Type ENTREE
    StockService -> StockService: ajouterStock(entrepotDest, produit, +quantite)
  else Type SORTIE
    StockService -> StockService: ajouterStock(entrepotSource, produit, -quantite)
  else Type TRANSFERT
    StockService -> StockService: ajouterStock(entrepotSource, produit, -quantite)
    StockService -> StockService: ajouterStock(entrepotDest, produit, +quantite)
  end
end
StockService -> StockService: getQuantiteTotale(codeProd)
StockService --> Main: quantite totale
Main --> Utilisateur: Afficher quantité totale
@enduml
```

PARTIE 4: CAHIER DES CHARGES COMPLET

---

1. INTRODUCTION

1.1. Contexte

Le présent projet consiste à développer une application Java permettant la gestion des produits dans plusieurs entrepôts, en assurant le suivi des quantités, des mouvements d'entrée et de sortie, et du transfert entre entrepôts.

1.2. Objectifs du projet

- Gérer plusieurs entrepôts simultanément
- Gérer un catalogue de produits
- Suivre les mouvements de stock (entrée / sortie)
- Gérer le transfert de produits entre entrepôts
- Consulter l'état des stocks en temps réel
- Générer des statistiques et rapports

1.3. Portée du projet

Application console Java avec interface menu interactif, persistance des données en fichiers JSON, et architecture modulaire respectant les principes de la POO.

---

2. FONCTIONNALITÉS DÉTAILLÉES

2.1. GESTION DES PRODUITS

2.1.1. Ajouter un produit

Description: Permet d'ajouter un nouveau produit au catalogue.

Données requises:

- Code produit (unique, obligatoire)

- Nom du produit (obligatoire)
- Description (optionnelle)
- Prix unitaire (obligatoire, > 0)

Contraintes:

- Le code produit doit être unique
- Le prix doit être positif

Résultat: Produit ajouté et sauvegardé dans data/produits.json

2.1.2. Modifier un produit

Description: Permet de modifier les informations d'un produit existant.

Données requises:

- Code produit (existant)
- Nouveau nom
- Nouvelle description
- Nouveau prix unitaire

Contraintes:

- Le produit doit exister

Résultat: Produit modifié et sauvegardé

2.1.3. Supprimer un produit

Description: Permet de supprimer un produit du catalogue.

Données requises:

- Code produit (existant)

Contraintes:

- Le produit doit exister
- Confirmation requise avant suppression

Résultat: Produit supprimé du catalogue

2.1.4. Lister les produits

Description: Affiche la liste complète de tous les produits.

Résultat: Tableau formaté avec code, nom, description, prix

2.1.5. Rechercher un produit

Description: Recherche un produit par son code et affiche ses détails.

Données requises:

- Code produit

Résultat: Détails complets du produit + quantité totale en stock

2.2. GESTION DES ENTREPÔTS

2.2.1. Ajouter un entrepôt

Description: Permet d'ajouter un nouvel entrepôt.

Données requises:

- Code entrepôt (unique, obligatoire)
- Nom de l'entrepôt (obligatoire)
- Adresse (obligatoire)
- Capacité maximale (obligatoire, > 0)

Contraintes:

- Le code entrepôt doit être unique
- La capacité doit être positive

Résultat: Entrepôt ajouté et sauvegardé dans data/entrepots.json

2.2.2. Modifier un entrepôt

Description: Permet de modifier les informations d'un entrepôt existant.

Données requises:

- Code entrepôt (existant)
- Nouveau nom
- Nouvelle adresse
- Nouvelle capacité maximale

Contraintes:

- L'entrepôt doit exister

Résultat: Entrepôt modifié et sauvegardé

2.2.3. Supprimer un entrepôt

Description: Permet de supprimer un entrepôt.

Données requises:

- Code entrepôt (existant)

Contraintes:

- L'entrepôt doit exister
- Confirmation requise avant suppression
- Avertissement si l'entrepôt contient du stock

Résultat: Entrepôt supprimé

2.2.4. Lister les entrepôts

Description: Affiche la liste complète de tous les entrepôts.

Résultat: Tableau formaté avec code, nom, adresse, capacité max

2.2.5. Rechercher un entrepôt

Description: Recherche un entrepôt par son code et affiche ses détails.

Données requises:

- Code entrepôt

Résultat: Détails complets de l'entrepôt + stock contenu

2.3. MOUVEMENTS DE STOCK

2.3.1. Enregistrer une entrée de stock

Description: Enregistre l'arrivée de produits dans un entrepôt.

Données requises:

- Code produit (existant)
- Code entrepôt (existant)
- Quantité (obligatoire, > 0)
- Commentaire (optionnel)

Contraintes:

- Le produit doit exister
- L'entrepôt doit exister
- La quantité doit être positive

Résultat:

- Mouvement enregistré dans data/mouvements.json
- Stock mis à jour automatiquement

2.3.2. Enregistrer une sortie de stock

Description: Enregistre la sortie de produits d'un entrepôt.

Données requises:

- Code produit (existant)
- Code entrepôt (existant)
- Quantité (obligatoire, > 0)
- Commentaire (optionnel)

Contraintes:

- Le produit doit exister
- L'entrepôt doit exister
- La quantité doit être positive
- Stock disponible suffisant

Résultat:

- Mouvement enregistré
- Stock déduit automatiquement

2.3.3. Historique des mouvements

Description: Affiche l'historique complet de tous les mouvements.

Résultat: Tableau avec ID, Type, Produit, Source, Destination, Quantité, Date, Commentaire

2.4. TRANSFERT ENTRE ENTREPÔTS

2.4.1. Transférer un produit

Description: Transfère des produits d'un entrepôt source vers un entrepôt destination.

Données requises:

- Code produit (existant)
- Code entrepôt source (existant)
- Code entrepôt destination (existant, différent de la source)
- Quantité (obligatoire, > 0)
- Commentaire (optionnel)

Contraintes:

- Le produit doit exister
- Les deux entrepôts doivent exister

- L'entrepôt source et destination doivent être différents
- Stock disponible suffisant dans l'entrepôt source

Résultat:

- Mouvement de type TRANSFERT enregistré
- Stock déduit de l'entrepôt source
- Stock ajouté à l'entrepôt destination

2.5. CONSULTATION

2.5.1. Quantité totale par produit

Description: Affiche la quantité totale d'un produit dans tous les entrepôts.

Données requises:

- Code produit

Résultat: Quantité totale affichée

2.5.2. Quantité par entrepôt

Description: Affiche la quantité d'un produit dans un entrepôt spécifique.

Données requises:

- Code produit
- Code entrepôt

Résultat: Quantité affichée

2.5.3. Liste des mouvements filtrée par produit

Description: Affiche tous les mouvements d'un produit spécifique.

Données requises:

- Code produit

Résultat: Liste filtrée des mouvements

2.5.4. Liste des mouvements filtrée par date

Description: Affiche tous les mouvements dans une période donnée.

Données requises:

- Date de début (format: yyyy-MM-dd HH:mm)
- Date de fin (format: yyyy-MM-dd HH:mm)

Résultat: Liste filtrée des mouvements

2.5.5. Liste des mouvements filtrée par produit et date

Description: Affiche les mouvements d'un produit dans une période donnée.

Données requises:

- Code produit
- Date de début
- Date de fin

Résultat: Liste filtrée des mouvements

2.6. STATISTIQUES & RAPPORTS

2.6.1. Vue d'ensemble du stock

Description: Affiche un tableau récapitulatif du stock par produit et par entrepôt.

Résultat: Tableau matriciel avec produits en lignes, entrepôts en colonnes, total par produit

2.6.2. Stock détaillé par entrepôt

Description: Affiche le stock détaillé de chaque entrepôt.

Résultat: Pour chaque entrepôt, liste des produits avec leurs quantités

2.6.3. Produits en rupture de stock

Description: Identifie et affiche les produits avec stock zéro.

Résultat: Liste des produits en rupture de stock

2.6.4. Top 10 produits

Description: Affiche les 10 produits avec les plus grandes quantités totales.

Résultat: Classement décroissant des produits par quantité totale

2.6.5. Statistiques des mouvements

Description: Affiche un résumé statistique des mouvements.

Résultat:

- Nombre total de mouvements
- Nombre d'entrées (avec quantité totale)
- Nombre de sorties (avec quantité totale)
- Nombre de transferts

2.6.6. Rapport complet

Description: Génère un rapport complet avec toutes les statistiques.

Résultat: Document récapitulatif incluant:

- Statistiques générales
- Vue d'ensemble du stock
- Produits en rupture
- Statistiques des mouvements

---

3. ARCHITECTURE TECHNIQUE

3.1. Structure des packages

com.project.stock/

├── models/ → Classes métiers (Produit, Entrepot, MouvementStock)

├── services/ → Logique métier (ProduitService, EntrepotService, StockService)

├── repository/ → Couche de persistance (fichiers JSON)

├── utils/ → Utilitaires (FileHelper, JsonHelper)

├── exceptions/ → Exceptions personnalisées

└── Main.java → Point d'entrée de l'application

3.2. Technologies utilisées

- Langage: Java 11
- Bibliothèque JSON: Jackson 2.15.2
- Gestion de dépendances: Maven
- Format de données: JSON
- Interface: Console avec menu interactif

3.3. Persistance des données

Les données sont sauvegardées dans des fichiers JSON:

- data/produits.json → Liste des produits
- data/entrepots.json → Liste des entrepôts
- data/mouvements.json → Historique des mouvements

3.4. Principes de conception

- Séparation des responsabilités (Models, Services, Repositories)
- Single Responsibility Principle
- DRY (Don't Repeat Yourself)

- Gestion d'erreurs avec exceptions personnalisées
- Validation des données avant traitement

---

4. CONTRAINTES ET RÈGLES MÉTIER

4.1. Contraintes de données

- Les codes (produit, entrepôt) doivent être uniques
- Les quantités doivent être positives (> 0)
- Les prix doivent être positifs (> 0)
- Les capacités d'entrepôt doivent être positives (> 0)

4.2. Règles métier

- Un produit ne peut pas être supprimé s'il a des mouvements associés (optionnel)
- Une sortie ne peut pas être effectuée si le stock est insuffisant
- Un transfert nécessite un stock suffisant dans l'entrepôt source
- Le stock est calculé automatiquement à partir des mouvements
- Les transferts sont enregistrés comme un seul mouvement de type TRANSFERT

4.3. Gestion des erreurs

- ProduitNotFound: Levée quand un produit n'existe pas
- EntrepotNotFound: Levée quand un entrepôt n'existe pas
- IllegalArgumentException: Levée pour données invalides
- IOException: Gérée pour les opérations de fichiers

---

5. INTERFACE UTILISATEUR

5.1. Menu principal

L'application propose un menu principal avec les options suivantes:

1. Gestion des Produits
2. Gestion des Entrepôts
3. Mouvements de Stock
4. Transfert entre Entrepôts
5. Consultation
6. Statistiques & Rapports
0. Quitter

5.2. Caractéristiques de l'interface

- Menus hiérarchiques clairs
- Messages de confirmation pour les actions importantes
- Affichage formaté en tableaux
- Messages d'erreur explicites
- Validation des saisies utilisateur

6. LIVRABLES

6.1. Code source

- Code Java complet et commenté
- Structure de packages respectée

- Gestion d'erreurs complète

6.2. Documentation

- README.md avec instructions d'installation et d'utilisation
- Cahier des charges (ce document)
- Diagrammes UML (classes et séquences)

6.3. Fichiers de configuration

- pom.xml pour Maven
- Structure de dossiers conforme aux standards Java

---

7. TESTS ET VALIDATION

7.1. Scénarios de test recommandés

- Ajout/modification/suppression de produits
- Ajout/modification/suppression d'entrepôts
- Enregistrement d'entrées et sorties
- Transferts entre entrepôts
- Consultation des stocks
- Génération de statistiques
- Gestion des erreurs (produit/entrepôt inexistant, stock insuffisant)

7.2. Validation

- Vérification de l'unicité des codes
- Vérification des contraintes de données

- Vérification de la cohérence des stocks

- Vérification de la persistance des données

---

8. AMÉLIORATIONS FUTURES (OPTIONNEL)

- Interface graphique (JavaFX/Swing)

- Base de données au lieu de fichiers JSON

- Authentification et gestion des utilisateurs

- Export des rapports en PDF/Excel

- Notifications pour produits en rupture

- Historique des modifications (audit trail)

- API REST pour intégration avec d'autres systèmes