Documentation du modèle
Système de Gestion d'Entrepôt (Warehouse Manager)

Chaymae MAKHOKH Anas BZIOUI

3 décembre 2025

Table des matières

Le modèle implémente les classes suivantes, telles que définies dans le diagramme UML :
— `ProduitAvecCaracteristiques`
— `ProduitAvecCycleDeVie`
— `Product`
— `ContrainteCompatibilite`
— `ReglesCompatibilite`
— `ElementsPalette`
— `Palette`
— `Conteneur`
— `Entrepot`
— les énumérations `TypeProduit`, `TypeConteneur`, `EtatProduit`

Toutes les classes du modèle sont regroupées dans le dossier `src/domain` du projet Qt.

1 Vue d'ensemble du modèle

Le modèle représente la structure suivante :
— Un **Entrepôt** (`Entrepot`) contient plusieurs **Conteneurs** (`Conteneur`) et plusieurs **Palettes** (`Palette`).
— Chaque **Conteneur** contient une liste de **Produits** (`Product`).
— Chaque **Palette** contient un ensemble d'**Éléments de palette** (`ElementsPalette`) qui eux-mêmes référencent des produits.
— Les **Produits** sont décrits par deux composantes : `ProduitAvecCaracteristiques` et `ProduitAvecCycleDeVie`.
— Les compatibilités entre produits sont définies par des `ContrainteCompatibilite`, regroupées dans `ReglesCompatibilite`.

2 Énumérations

2.1 `TypeProduit`

Rôle Enumération représentant le type d'un produit. Elle correspond au bloc *«enum» TypeProduit* du diagramme.

Valeurs
— `Alimentaire`
— `Electronique`
— `Medicament`
— `Autre` (valeur par défaut pour les cas non précisés)

Utilisation
— Attribut `type` de la classe `Product`.
— Attributs `typeA` et `typeB` de la classe `ContrainteCompatibilite`.
— Utilisé dans les méthodes de compatibilité de `ReglesCompatibilite`.

2.2 `TypeConteneur`

Rôle Enumération représentant le type d'un conteneur (normal, froid, fragile, etc.). Elle correspond au bloc *«enum» TypeConteneur* dans le diagramme.

Valeurs
- `Normal`
- `Froid`
- `Fragile`
- `Autre`

Utilisation Attribut `type` dans la classe `Conteneur`.

2.3 `EtatProduit`

Rôle Enumération représentant l'état d'un produit. Elle correspond au bloc *«enum» EtatProduit* du diagramme.

Valeurs
- `Stocke`
- `Expedie`

Utilisation
- Attribut `etat` dans `ProduitAvecCaracteristiques`.
- Attribut `etat` dans `ProduitAvecCycleDeVie`.

3 Classes de données simples

3.1 `ProduitAvecCaracteristiques`

Rôle Classe de données représentant les caractéristiques physiques et les conditions de conservation d'un produit. Elle correspond à la classe *ProduitAvecCaracteristiques* reliée à `Product` par une composition (losange plein) dans le diagramme UML.

Attributs
- `double m_poids` : poids du produit.
- `double m_volume` : volume occupé par le produit.
- `QString m_conditionsConservation` : texte décrivant les conditions de conservation (ex. « -18°C »).
- `EtatProduit m_etat` : état du produit (stocké/expédié).

Méthodes principales La classe est implémentée uniquement dans un fichier d'en-tête (`produit_caracteris`
avec des accesseurs inline :
- Getters et setters pour chaque attribut : `poids()`, `setPoids()`, `volume()`, `setVolume()`, `conditionsConservation()`, `setConditionsConservation()`, `etat()`, `setEtat()`.

Lien avec le diagramme
- Tous les attributs de la boîte UML *ProduitAvecCaracteristiques* sont présents.
- La composition 1–1 avec `Product` est réalisée par deux attributs privés dans `Product` : `ProduitAvecCaracteristiques m_caracteristiques;` et des méthodes d'accès `caracteristiques()`.

3.2 `ProduitAvecCycleDeVie`

Rôle Classe de données représentant le cycle de vie d'un produit (dates et état). Elle correspond à la classe *ProduitAvecCycleDeVie* reliée à `Product` par une composition dans le diagramme.

Attributs
- `QDate m_dateEntreeStock`
- `QDate m_datePeremption`
- `EtatProduit m_etat`

Méthodes Getters / setters inline dans `produit_cycledevie.h` : `dateEntreeStock()`, `setDateEntreeStock(`
`datePeremption()`, `setDatePeremption()`, `etat()`, `setEtat()`.

Lien UML
- Reprise exacte des attributs du diagramme.
- Composition 1–1 avec `Product` via l'attribut `ProduitAvecCycleDeVie m_cycleDeVie;` dans `Product`.

3.3 `ElementsPalette`

Rôle Élément de la classe `Palette` correspondant à la boîte *ElementsPalette* du diagramme (quantité + liste de produits).

Attributs
- `int m_quantite` : quantité de produits.
- `QList<Product*> m_produits` : liste de pointeurs vers les produits concernés.

Méthodes Getters / setters simples pour la quantité et la liste de produits, définis dans `elements_palette.h`.

Lien UML L'association 1–* entre `Palette` et `ElementsPalette` est implémentée par l'attribut :
- `QList<ElementsPalette> m_elements;` dans `Palette`.

4 Classes métier principales

4.1 `Product`

Rôle Classe centrale représentant un produit manipulé dans l'entrepôt. Elle hérite de `QObject` pour pouvoir émettre le signal `productChanged()` et être reliée à l'interface Qt par signaux/slots. Elle correspond à la classe *Produit* du diagramme UML.

Attributs privés
- `QString m_idProduit` (UML : `IdProduit : String`)
- `QString m_nom` (UML : `Nom : String`)
- `TypeProduit m_type` (UML : `type : TypeProduit`)
- `double m_capaciteMax` (UML : `capaciteMax : Double`)
- `ProduitAvecCaracteristiques m_caracteristiques`
- `ProduitAvecCycleDeVie m_cycleDeVie`

Méthodes publiques
- Accesseurs pour les attributs d'identité et de type : `idProduit()`, `setIdProduit()`, `nom()`, `setNom()`, `type()`, `setType()`, `capaciteMax()`, `setCapaciteMax()`.
- Accès aux sous-objets : `caracteristiques()`, `cycleDeVie()`.
- Raccourcis vers les attributs des sous-objets : `poids()`, `volume()`, `dateEntreeStock()`, `datePeremption()`, `etat()`.

— Méthodes métier :
  — `bool estPerime(const QDate& aujourdHui) const;`
  — `int joursAvantPeremption(const QDate& aujourdHui) const;`

Signal
— `void productChanged();` : émis lorsque l'une des propriétés du produit est modifiée.

Lien UML
— Tous les attributs et types du bloc *Produit* sont repris.
— Les compositions vers `ProduitAvecCaracteristiques` et `ProduitAvecCycleDeVie` sont implémentées par des membres internes.
— L'association 0..* vers `Conteneur` et `Palette` n'est pas codée directement dans `Product` mais via les collections dans `Conteneur` et `Palette` (relation unidirectionnelle côté conteneur/palette).

4.2 `ContrainteCompatibilite`

Rôle Représente une règle élémentaire de compatibilité entre deux types de produit. Elle correspond à la classe *ContrainteCompatibilite* du diagramme UML.

Attributs
— `TypeProduit m_typeA` (UML : `typeA : TypeProduit`)
— `TypeProduit m_typeB` (UML : `typeB : TypeProduit`)
— `bool m_compatible` (UML : `compatible : bool`)

Méthodes
— Getters / setters pour les trois attributs.
— `bool concerne(TypeProduit a, TypeProduit b) const;` : teste si la contrainte s'applique au couple de types donné (ordre indifférent).

Lien UML La classe reprend exactement les attributs de la boîte UML correspondante et est utilisée par `ReglesCompatibilite` avec une multiplicité 1..*.

4.3 `ReglesCompatibilite`

Rôle Contient l'ensemble des règles de compatibilité entre types de produits. Elle correspond à *ReglesCompatibilite* dans le diagramme, en association 1..* avec `ContrainteCompatibilite`.

Attributs
— `QList<ContrainteCompatibilite> m_contraintes;`

Méthodes
— `void ajouterContrainte(const ContrainteCompatibilite& c);` : ajoute une règle.
— `bool areCompatible(TypeProduit a, TypeProduit b) const;` : renvoie `true` si les deux types sont compatibles selon les contraintes enregistrées.
— `const QList<ContrainteCompatibilite>& contraintes() const;` : accès en lecture à la liste des contraintes.

Signal
— `void rulesChanged();` : émis lorsque les règles sont modifiées.

Lien UML

- La relation 1..* vers `ContrainteCompatibilite` est implémentée par la liste `m_contraintes`.
- `ReglesCompatibilite` est utilisée par `Entrepot` et `Palette` pour vérifier la compatibilité des produits lors de la construction des palettes (algorithmes FIFO/FEFO).

4.4 `Palette`

Rôle Représente une palette d'expédition, contenant un ensemble d'éléments (`ElementsPalette`) et donc de produits. Elle correspond à la classe *Palette* du diagramme UML.

Attributs

- `QString m_idPalette` (UML : `IdPalette : String`)
- `QString m_destination` (UML : `Destination : String`)
- `QDate m_dateEnvoiPrevue` (UML : `dateEnvoiPrevue : QDate`)
- `double m_capaciteMax` (UML : `capaciteMax : Double`)
- `QList<ElementsPalette> m_elements`
- Les contraintes de compatibilité sont gérées via `ReglesCompatibilite`, passée en paramètre aux méthodes d'ajout.

Méthodes

- Accesseurs classiques pour les attributs d'identité et de date.
- `const QList<ElementsPalette>& elements() const;` et `QList<ElementsPalette>& elementsRef();` pour accéder aux éléments.
- `double poidsTotal() const;` : calcule le poids total de la palette en sommant les produits contenus dans `m_elements`.
- `bool peutAjouter(Product *p, const ReglesCompatibilite *regles) const;` : vérifie si un produit peut être ajouté en respectant la capacité max et les règles de compatibilité.
- `bool ajouterProduit(Product *p, const ReglesCompatibilite *regles);` : ajoute le produit à la palette (en augmentant la quantité si nécessaire).

Signal

- `void paletteChanged();` : émis lorsqu'on modifie le contenu ou les propriétés de la palette.

Lien UML

- Les attributs correspondent à ceux de la boîte *Palette*.
- La relation 1..* vers `ElementsPalette` est implémentée par `QList<ElementsPalette> m_elements;` avec une multiplicité 0..*.
- La relation vers `ContrainteCompatibilite` est représentée par l'utilisation de `ReglesCompatibilite` lors des ajouts de produits.

4.5 `Conteneur`

Rôle Représente un conteneur physique dans l'entrepôt. Il correspond à la classe *Conteneur* du diagramme UML.

Attributs

- `QString m_idConteneur` (UML : `IdConteneur : String`)
- `TypeConteneur m_type` (UML : `type : TypeConteneur`)
- `double m_capaciteMax` (UML : `capaciteMax : Double`)
- `QList<Product*> m_produits` (UML : `Produits : List<Produit>`)

Méthodes

- Accesseurs pour `idConteneur`, `type`, `capaciteMax`, `produits()`.
- `double poidsTotal() const;` : somme des poids des produits.
- `bool peutAjouter(Product *p) const;` : vérifie la capacité max.
- `bool ajouterProduit(Product *p);` : ajoute un produit si possible.
- `void retirerProduit(Product *p);` : retire un produit.

Signaux

- `void conteneurChanged();`
- `void produitAjoute(Product *p);`
- `void produitRetire(Product *p);`

Lien UML La composition entre `Entrepot` et `Conteneur` (1..* conteneurs par entrepôt) est implémentée par l'attribut `QList<Conteneur*> m_conteneurs;` dans `Entrepot`. La relation 0..* vers `Product` est implémentée par `m_produits`.

4.6 `Entrepot`

Rôle Racine du modèle hiérarchique : représente l'entrepôt global. Il correspond à la classe *Entrepot* du diagramme UML et constitue le niveau 1 de l'arborescence.

Attributs

- `QString m_idEntrepot` (UML : `IdEntrepot : String`)
- `QString m_nom` (UML : `Nom : String`)
- `QString m_adresse` (UML : `Adresse : String`)
- `double m_surface` (UML : `Surface : Double`)
- `QList<Conteneur*> m_conteneurs` (UML : `allConteneurs`)
- `QList<Palette*> m_palettes` (UML : `allPalette`)
- `ReglesCompatibilite m_regles` (ensemble des contraintes).

Méthodes

- Accesseurs pour l'identité et l'adresse de l'entrepôt.
- Accès aux listes : `const QList<Conteneur*>& conteneurs() const;`, `const QList<Palette*>& palettes() const;`.
- `QList<Product*> tousLesProduits() const;` : construit la liste de tous les produits à partir des conteneurs.
- Gestion des conteneurs : `Conteneur* creerConteneur();`, `void supprimerConteneur(Conteneur *c);`.
- Gestion des produits : `Product* creerProduitDans(Conteneur *c);`, `void supprimerProduitDe(Cont *c, Product *p);`.
- Gestion des palettes : `Palette* creerPalette();`, `void supprimerPalette(Palette *p);`.
- Accès aux règles de compatibilité : `ReglesCompatibilite* regles();`.
- Algorithmes de construction automatique des palettes :
  - `void genererPalettesFIFO(double capacitePalette);`
    tri des produits par date d'entrée (*First In First Out*).
  - `void genererPalettesFEFO(double capacitePalette);`
    tri des produits par date de péremption (*First Expire First Out*).

Signal

— `void entrepotChanged();` : émis lorsqu'on modifie la structure (ajout/suppression de conteneurs, produits, palettes).

Lien UML

— Les attributs correspondent au bloc *Entrepot* du diagramme.

— Les associations 0..* vers `Conteneur`, `Product` et `Palette` sont implémentées via les listes `m_conteneurs`, `m_palettes` et la méthode `tousLesProduits()`.

— La relation avec `ReglesCompatibilite` est implémentée par un membre interne `m_regles`.