Lagerliste 1.0

Eine Lagerliste verwaltet Lagerbewegungen eines Artikels. Dabei werden folgende Operationen unterstützt:

- **store**: Damit wird in der Lagerliste das Einlagern einer bestimmten Menge bewertet mit einem bestimmten Preis pro Mengeneinheit vorgenommen. Der Vorgang ist mit einem Datum gekennzeichnet.
  Beispiel: 7.1.2024, 100, € 2.50 -> am 7.1.2024 werden 100 Einheiten zu je € 2,50 eingelagert
- **remove**: Store: Damit wird die Entnahme einer bestimmten Menge zu einem gegebenen Datum verzeichnet. Die Entnahme muss bewertet werden.
  Beispiel: 9.1.2024, 10-> am 9.1.2024 werden 10 Einheiten entnommen. Der Preis je Einheit muss noch ermittelt werden.
- **getStockStatus**: Damit wird der aktuelle Lagerstand als String zurückgegeben.
  Beispiel für einen Rückgabewert:

07.01.2024	90 EH je	€ 2,50	€ 225,00
12.01.2024	100 EH je	€ 2,70	€ 270,00

  Hier sind noch 2 Warenbestände vom 07.01. bzw. 12.01. vorhanden.
- **getStockIngoings**: Damit werden alle registrierten Wareneingänge als String zurückgegeben.
  Beispiel für einen Rückgabewert:

07.01.2024	100 EH je	€ 2,50	€ 250,00
12.01.2024	100 EH je	€ 2,70	€ 270,00

  Es gab hier 2 Wareneingänge am 07.01. bzw. 12.01. mit jeweils 100 EH.
- **getStockOutgoings**: Damit werden alle registrierten Warenausgänge als String zurückgegeben.
  Beispiel für einen Rückgabewert:

09.01.2024	10 EH je	€ 2,50	€ 25,00
15.01.2024	25 EH je	€ 2,50	€ 62,50
16.01.2024	20 EH je	€ 2,50	€ 50,00

  Es gab hier 3 Warenausgänge mit jeweils 10, 25, bzw. 20 EH.

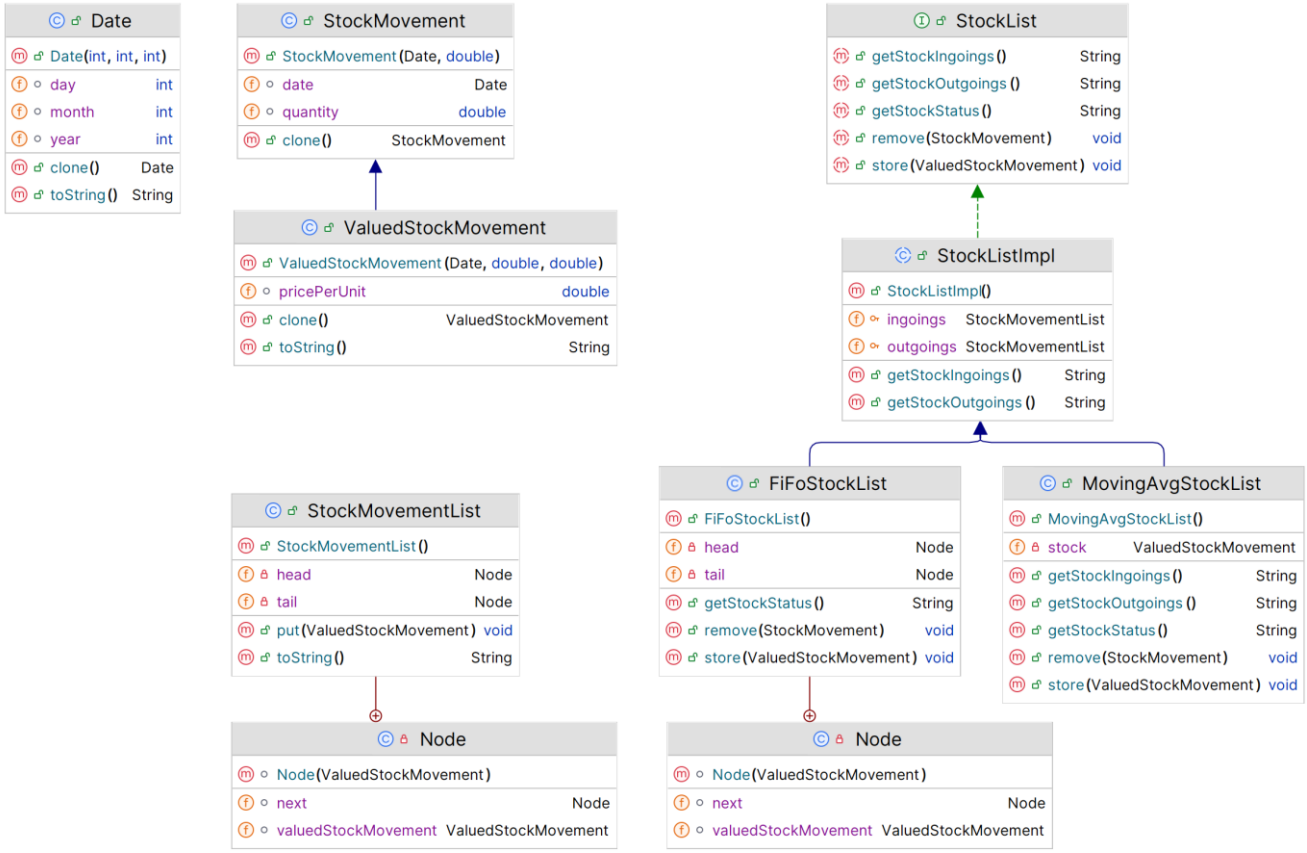
Bezüglich der Warenentnahme und der Bewertung gibt es aktuell 2 verschiedene Implementierung:

- Die FIFO-Warenliste folgt dem First-In-First-Out-Prinzip. Jede Warenentnahme wird also zunächst vom ältesten Warenbestand bedient, bis dieser aufgebraucht ist. Dann wird zum zweitältesten Warenbestand gegriffen usw. Demensprechend wird auch die Warenentnahme bewertet.
  Beispiel:

Aktueller Warenbestand	09.01.2024	100 EH	je	€ 2,50	€	250,00
	12.01.2024	100 EH	je	€ 2,70	€	270,00
Beabsichtigte Entnahme	14.01.2024	120 EH				
Bewertung		100 EH	je	€ 2,50	€	250,00
		20 EH	Je	€ 2,70	€	54,00
Warenbestand nach Entnahme	12.01.2024	80 EH	je	€ 2,70	€	216,00

- Beim Gleitenden Durchschnittspreisverfahren wird nach jedem Eingang ein Durchschnittspreis für die Gesamtmenge des Lagers berechnet. Jeder Warenausgang wird nun mit diesem Durchschnittspreis bewertet, bis wieder ein neuer Lagerzugang stattfindet, welcher zu einem neuen Durchschnittspreis führen kann, usw.

Klassendiagramm:



Aufgabenstellung:

- Implementieren Sie die im Klassendiagramm angeführten Klassen und Interfaces. Beschränken Sie sich im ersten Schritt nur auf das FiFo-Verfahren. Erstellen Sie dazu 3 Packages mit folgenden Klassen:
  - **date**: Date
  - **stock**: StockMovement, ValuedStockMovement, StockMovementList, StockList, StocklistImpl, FiFoStockList, MovingAvgStockList
  - **stockApp**: Main
- Sie können grundsätzlich davon ausgehen, dass alle Lagerein- und Ausgänge in chronologischer Reihenfolge verarbeitet werden.
- Implementieren Sie in der Main-Klasse ein Testscript, mit dem Sie einfache Lagervorgänge prüfen können.

Lösungshinweis:

- Implementieren Sie bei FIFO-Verfahren eine verkettete Liste, welche jeden Wareneingang als Knoten in die Liste einfügt. Beachten Sie dabei:
  - Eine Warenentnahme führt zunächst einmal dazu, dass lediglich die Menge (= Lagerbestand) des Knoten verändert wird. Erst wenn die Lagermenge eines Knoten 0 beträgt, muss der Knoten aus der Liste entfernt werden.
  - Beachten Sie, dass bei einer Warenentnahme so auch mehrere Knoten betroffen sein können, nämlich dann, wenn die angefordert Menge größer als der aktuelle Bestand des ältesten Knotens ist.
- Damit Sie alle registrierten Warenein- bzw. Warenausgänge abfragen können, benötigen Sie hier zwei weitere Listen zur Verzeichnung dieser Vorgänge, die in der Klasse *StockListImpl* implementiert werden, da die Protokollierung aller Warenein- und Ausgänge vom Bewertungsverfahren unabhängig ist.
- *Date*-Objekte und *StockMovement*- bzw. *ValuedStockMovement*-Objekte sollen clonebar (Interface *Clonable* implementieren!) sein, damit dafür einfache Objekt-Kopien erzeugt werden können.