AUFGABENSTELLUNG

ALLGEMEINE INFORMATIONEN

Die Aufgabenstellung ist als *Einzelarbeit* innerhalb von 2,5 Stunden zu lösen. Wie im realen Programmieralltag ist es erlaubt, das Internet zur Recherche zu verwenden. Bei Fragen zur Problemstellung, wende dich bitte an einen der KNAPP-Betreuer.

EINLEITUNG

KNAPP zählt zu den führenden Technologieunternehmen für Automatisierungslösungen und Software für Distribution und Produktion. Kunden auf der ganzen Welt aus unterschiedlichen Branchen wie zum Beispiel Gesundheitswesen, Onlinehandel, Lebensmittelhandel, Mode- und Textil-Branche bis hin zum Automotive-Bereich vertrauen auf die intelligenten Lösungen von KNAPP. Jede dieser Branchen hat dabei ihre Besonderheiten, denen KNAPP durch maßgeschneiderte Systemlösungen gerecht wird.

In der Mode- und Textil-Branche, speziell für Onlinehandel und Omnichannel Retail, haben sich Lösungen mit sogenannten Taschensortern etabliert. Der Vorteil dieser Lösung ist, dass mithilfe von Taschensorter eine Vielzahl an unterschiedlichen Waren (langsamdrehend, schnelldrehend) effizient und rasch mit einem System bearbeitet werden können. Und das unabhängig davon, ob die Waren für eine Filiale oder einen Endkunden bestimmt sind.

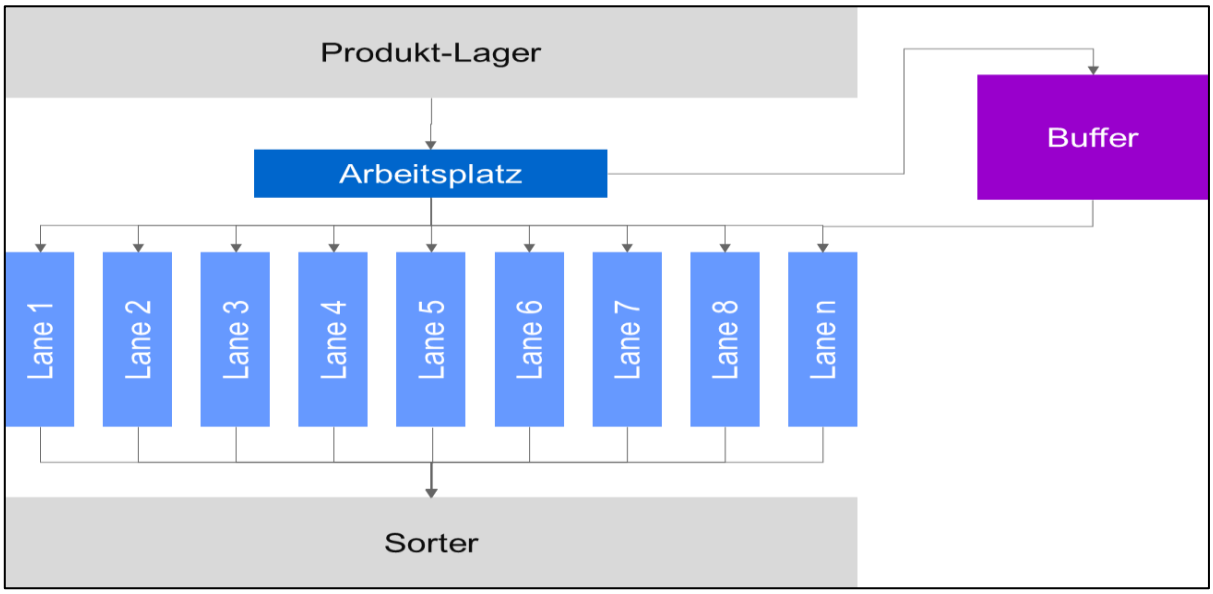


Abbildung 1 – Struktur des Lagers

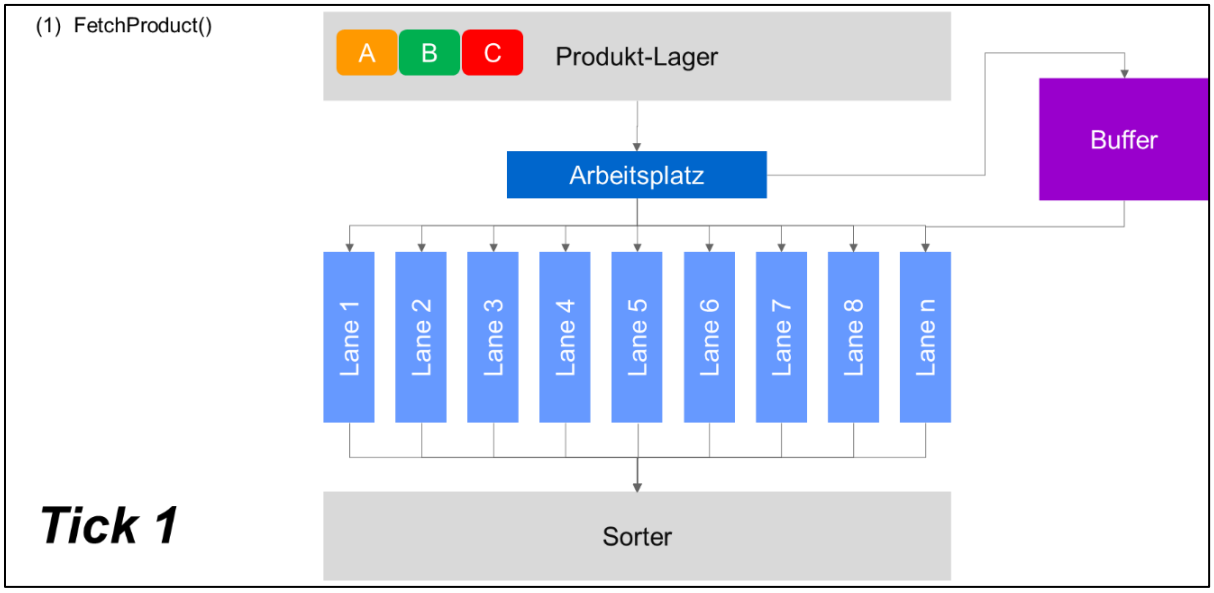
Die Bearbeitung der Waren erfolgt auf Einzelstückbasis, das bedeutet, dass in jede Tasche ein Stück eines Artikels übergeben wird. Die Taschen werden zum Vorpuffer – sogenannten Lanes – gebracht und von dort aus dann in den Taschensorter geschickt. Der Taschensorter sortiert die Taschen in eine vordefinierte Reihenfolge, danach werden die Waren aus den Taschen auftragsbezogen in Kartons verpackt und versendet.

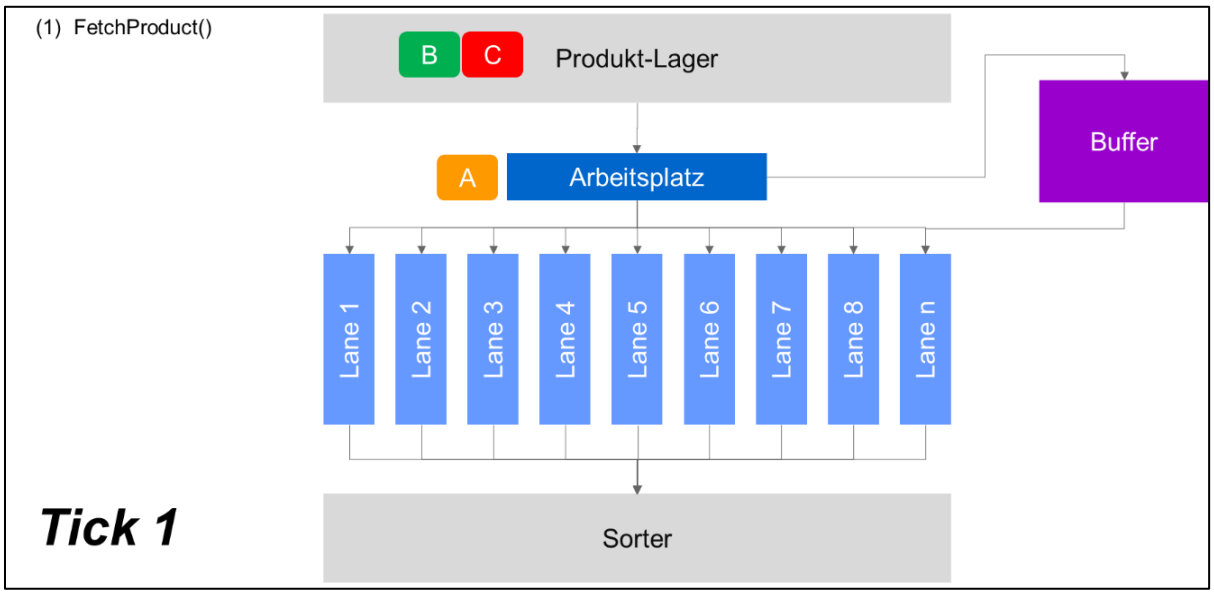
Abbildung 2 - Ablauf in einem Lager mit Taschensorter

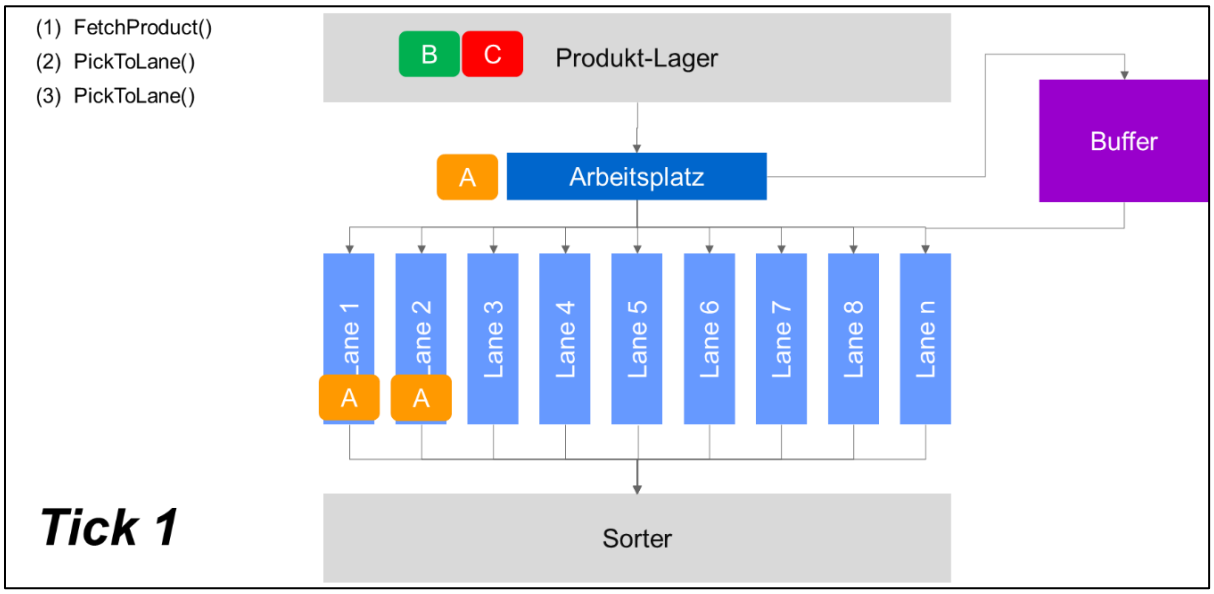
Da die Kartons durch Transportunternehmen vom Lager aus weiter transportiert werden, müssen diese zu bestimmten Zeiten im Versand zur Abholung bereitstehen.

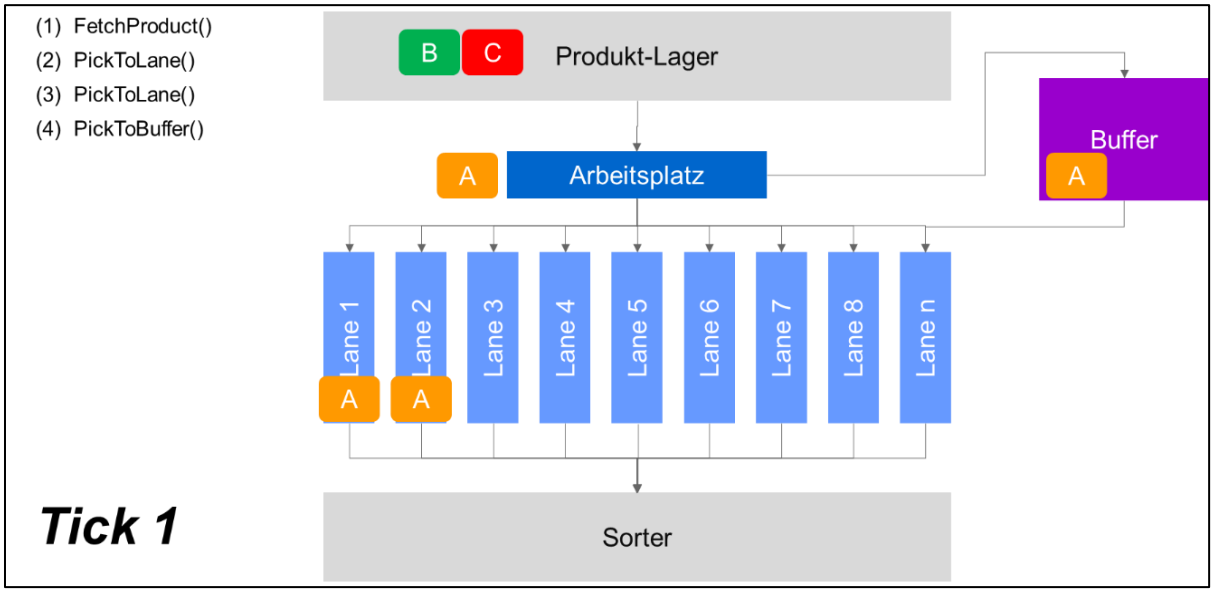
Deine Aufgabe ist es, Artikel zu holen und die Vorbereitung für die Sortierung zu steuern, sodass die einzelnen Aufträge in richtiger Reihenfolge im Versand sind und insgesamt möglichst wenig Arbeit anfällt.

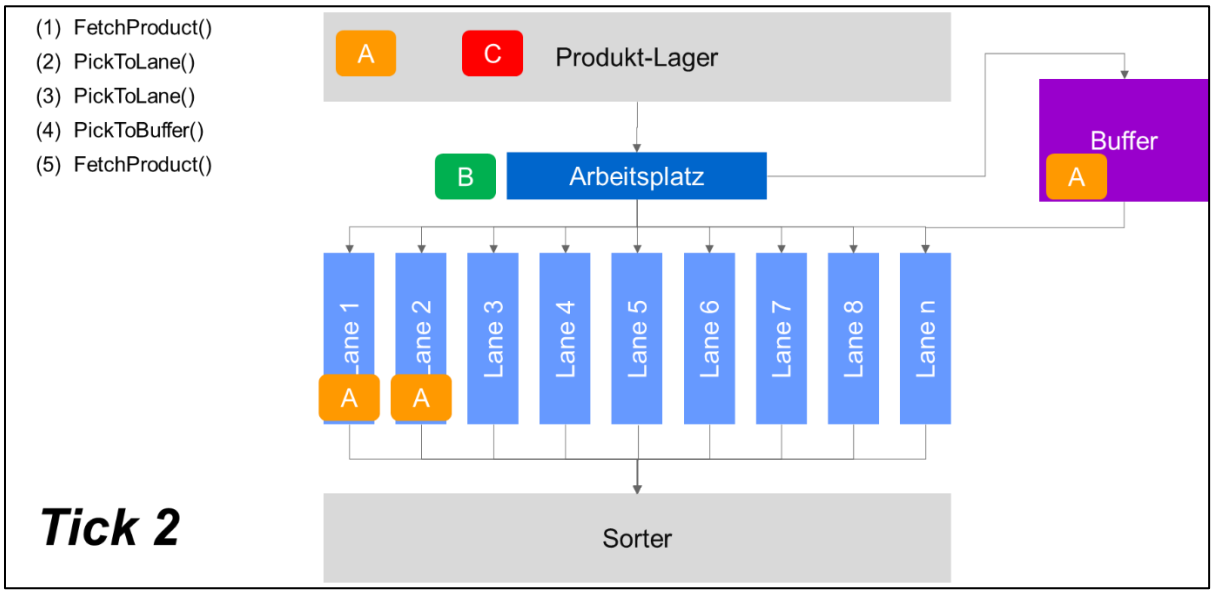
ABLAUF

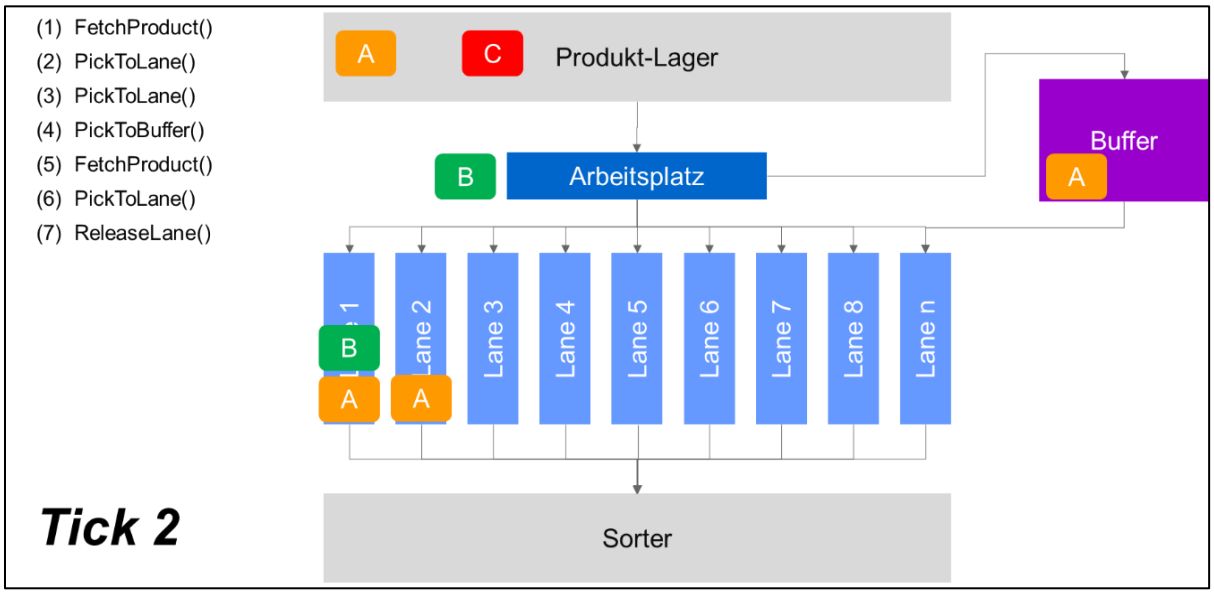

(1) FetchProduct()
A
B
C
Produkt-Lager
Buffer
Arbeitsplatz
Lane 1
Lane 2
Lane 3
Lane 4
Lane 5
Lane 6
Lane 7
Lane 8
Lane n
Tick 1
Sorter

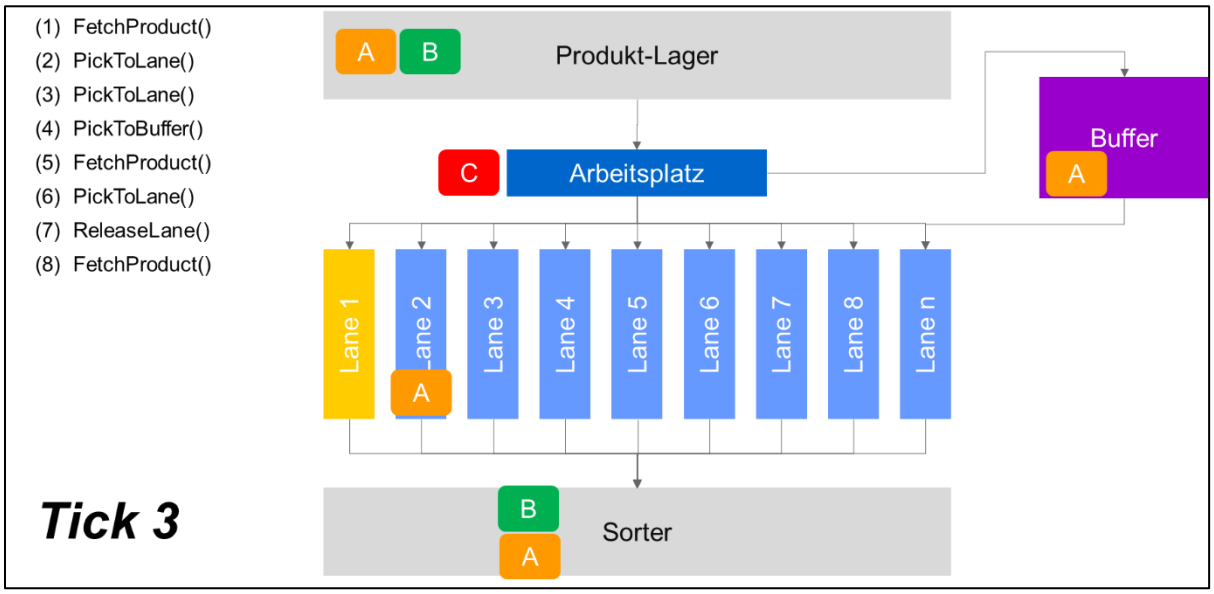

(1) FetchProduct()
B
C
Produkt-Lager
Buffer
A
Arbeitsplatz
Lane 1
Lane 2
Lane 3
Lane 4
Lane 5
Lane 6
Lane 7
Lane 8
Lane n
Tick 1
Sorter

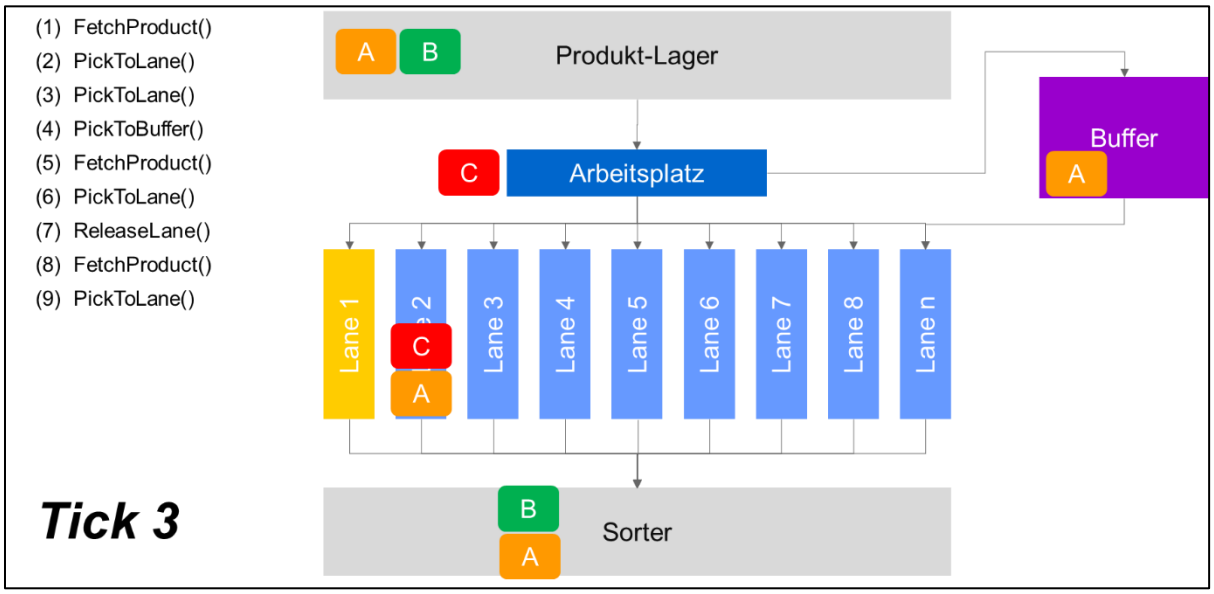

(1) FetchProduct()
(2) PickToLane()
(3) PickToLane()
B
C
Produkt-Lager
Buffer
A
Arbeitsplatz
Lane 1
Lane 2
Lane 3
Lane 4
Lane 5
Lane 6
Lane 7
Lane 8
Lane n
A
A
Tick 1
Sorter

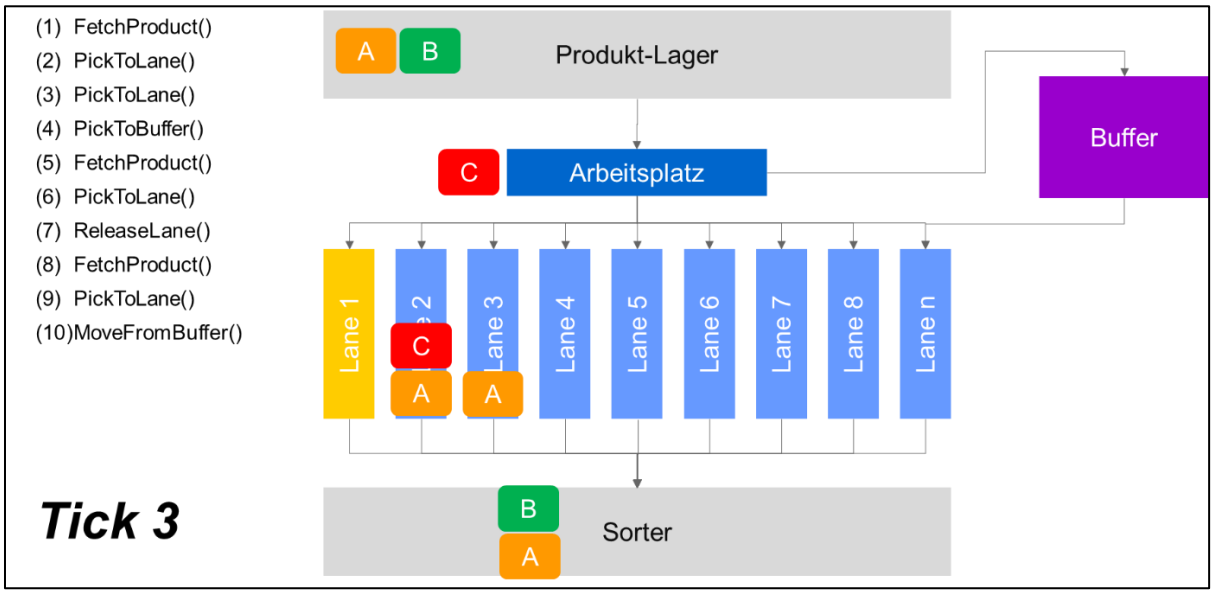

(1) FetchProduct()
(2) PickToLane()
(3) PickToLane()
(4) PickToBuffer()
B
C
Produkt-Lager
Buffer
A
A
Arbeitsplatz
Lane 1
Lane 2
Lane 3
Lane 4
Lane 5
Lane 6
Lane 7
Lane 8
Lane n
A
A
Tick 1
Sorter


(1) FetchProduct()
(2) PickToLane()
(3) PickToLane()
(4) PickToBuffer()
(5) FetchProduct()
A
C
Produkt-Lager
B
Arbeitsplatz
Buffer
A
Lane 1
Lane 2
Lane 3
Lane 4
Lane 5
Lane 6
Lane 7
Lane 8
Lane n
A
A
Tick 2
Sorter


(1) FetchProduct()
(2) PickToLane()
(3) PickToLane()
(4) PickToBuffer()
(5) FetchProduct()
(6) PickToLane()
(7) ReleaseLane()
A
C
Produkt-Lager
B
Arbeitsplatz
Buffer
A
B
A
A
Lane 3
Lane 4
Lane 5
Lane 6
Lane 7
Lane 8
Lane n
Tick 2
Sorter


(1) FetchProduct()
(2) PickToLane()
(3) PickToLane()
(4) PickToBuffer()
(5) FetchProduct()
(6) PickToLane()
(7) ReleaseLane()
(8) FetchProduct()
Tick 3
A
B
Produkt-Lager
C
Arbeitsplatz
Buffer
A
Lane 1
Lane 2
A
Lane 3
Lane 4
Lane 5
Lane 6
Lane 7
Lane 8
Lane n
B
A
Sorter


(1) FetchProduct()
(2) PickToLane()
(3) PickToLane()
(4) PickToBuffer()
(5) FetchProduct()
(6) PickToLane()
(7) ReleaseLane()
(8) FetchProduct()
(9) PickToLane()
Tick 3
A
B
Produkt-Lager
C
Arbeitsplatz
Buffer
A
Lane 1
C
A
Lane 3
Lane 4
Lane 5
Lane 6
Lane 7
Lane 8
Lane n
B
A
Sorter



Die Aufgabenstellung ist rundenbasiert, ein Aufruf von `Tick()` entspricht einer Runde.
Pro Runde ist folgendes möglich:

Wie oft?	Methode	Tätigkeit
0..1	`FetchProduct()`	Artikel holen
0..n	`PickToLane()`	Geholten Artikel auf Lanes aufteilen
0..n	`PickToBuffer()`	Geholten Artikel in den Puffer legen
0..n	`MoveFromBuffer()`	Beliebigen Artikel vom BPffer auf Lanes aufteilen
0..1	`ReleaseLane()`	Lane zum Sorter schicken

WICHTIGE METHODEN

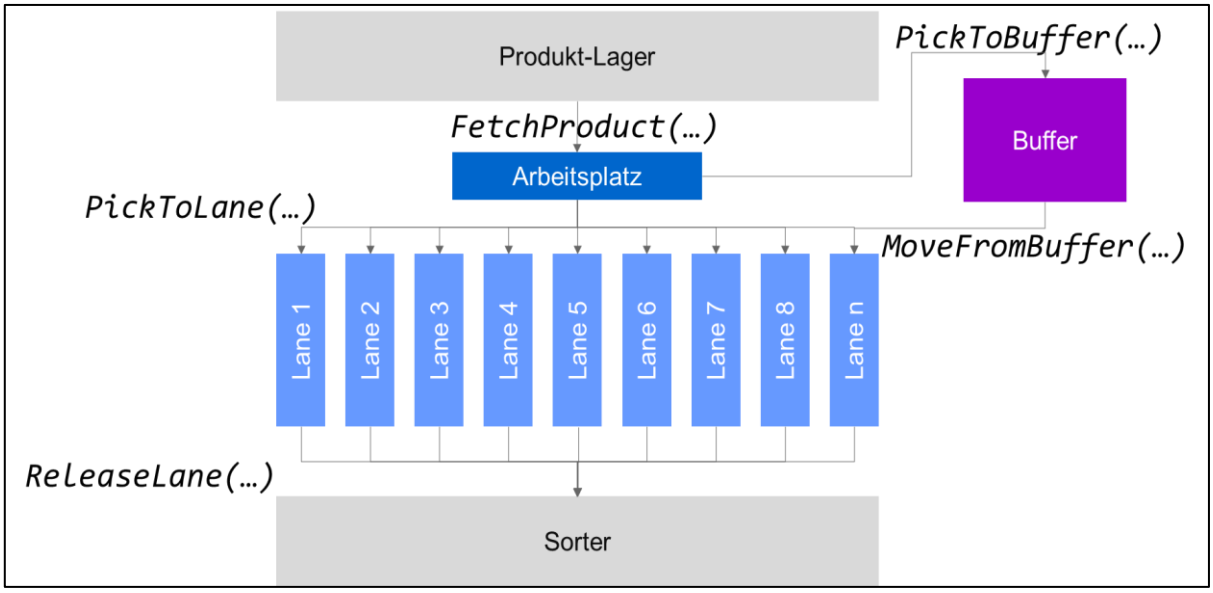


Abbildung 3 - Wichtige Methoden

ERLÄUTERUNGEN

Artikel

Artikel (*Product*) sind Waren, die gehandelt werden und voneinander unterscheidbar sind. Ein rotes Hemd, Größe 42 ist ein Artikel, es unterscheidet sich von einem roten Hemd Größe 43.

Ein Artikel hat einen Code (*ProductCode*), der es eindeutig identifiziert.

Auftrag

Ein Auftrag (*Order*) besteht aus mehreren Artikeln in einer bestimmten Stückzahl, die aus dem Lager an einen Kunden ausgeliefert werden.

Taschensorter

Der Taschensorter kann aus jeder beliebigen Reihenfolge an Artikeln die angeliefert wird bei konstanter Leistung jede beliebige Auslieferreihenfolge herstellen.

Abbildung 4 - Schema Taschensorter

Lane

Eine Lane ist ein Zwischenlager für Artikel, die zu einer Auftragszeile gehören, bevor diese sortiert werden.

- Ein Auftrag (Order) muss komplett auf einer Lane sein, die Reihenfolge der Zeilen ist egal, diese können auch mit anderen Aufträgen vermischt sein.
- Nur eine Lane kann abziehen (releasing): `warehouse.GetReleasingLane()`
- Anzahl an Lanes: `warehouse.getLanes().Count`
- Maximale Kapazität in Stück: `Lane::Size`

Buffer

Kann verschiedene Produkte ohne Bezug zu einem Auftrag speichern.

- Maximale Kapazität in Stück: *`Buffer::Size`*

ZEITVERHALTEN

- *`FetchProduct()`*
  - Wird sofort ausgeführt, Artikel steht nach dem Aufruf zur Verfügung
- *`PickToLane(), PickToBuffer(), MoveFromBuffer()`*
  - Werden sofort ausgeführt, die Artikel sind nach dem Aufruf am jeweiligen Ort.
- *`ReleaseLane()`*
  - Nach Aufruf beginnt der Abzug und damit die Sortierung der Lane.
- Sortierung
  - Die Dauer der Sortierung setzt sich aus zwei Teilen zusammen.
    - Fixe Dauer
      - *`input.Characteristics.ReleaseLaneFixedTicks`*
    - Dynamischer Anteil pro Stück
      - Ceil((ReleaseLaneItemsTick * Stück) / LaneSize)
  - Das Sortieren einer Lane zieht sich somit über mehrere Ticks. Die Lane ist nicht sofort nach *`ReleaseLane()`* wieder verfügbar.

ALLGEMEINES

- Es gibt eine maximale Anzahl an Ticks, nach dieser wird die Ausführung beendet.
  - *`input.Characteristics.MaxTickCount`*
- Produkte stehen von Anfang an zur Verfügung.
- Produkte stehen in unendlicher Menge zur Verfügung.
- Alle Aufträge sind von Anfang an bekannt.
- Ein Auftrag passt vollständig auf eine leere Lane.

MUSS-BEDINGUNGEN

- Ein Auftrag muss komplett auf einer Lane sein, bevor diese abgezogen werden kann
    - Die Reihenfolge ist nicht relevant, es können Auftragszeilen verschiedener Aufträge relevant sein.
    - Beim Abziehen (release) sortiert der KNAPP-Code automatisch die Aufträge einer Lane nach Abfahrtszeit.
- Nur eine Lane kann zeitgleich sortiert werden.
- Es kann nur auf eine Lane zugeteilt werden, die nicht abzieht.

BEWERTUNGSSCHEMA

- Es werden nur auf den Bewertungsserver hochgeladene Ergebnisse gewertet.
- Die Resultate werden am Server, mit den dort hinterlegten Algorithmen, errechnet.
- Die Punkte errechnen sich aus
    - den verbrauchten Ticks
    - der Anzahl an Aufrufen von `FetchProduct()`
    - Penalties
        - Für jeden Auftrag, der nach Ablauf der maximalen Ticks nicht fertig sortiert wurde.
        - Für jeden Auftrag, der sortiert wird nachdem ein anderer Auftrag mit späterer Abfahrtszeit sortiert wurde.
            - Innerhalb der Lane sortiert der KNAPP-Code die Zeiten.
    - dem Abgabezeitpunkt

Die Gewichtung zwischen Abgabezeitpunkt und Kosten ist dabei so gewählt, dass in der Regel eine bessere Lösung auch bei späterer Abgabe ein besseres Ergebnis bedeutet.

Bei mehreren Abgaben (Uploads) zählt immer die beste Abgabe zu dem Zeitpunkt, an dem diese getätigt wurde. Wenn Du nach dem Upload einer Lösung

weiterarbeitest und durch Deine Änderungen das Ergebnis schlechter wird, bleibt die zweite Abgabe unberücksichtigt.

ABGABEMODUS

Zur Beurteilung des Ergebnisses muss die vom KNAPP-Code automatisch erzeugte Datei `upload2019.zip` über die Abgabeseite hochgeladen werden. In diese Datei wird dein Source automatisch mitverpackt. Dieser Source muss das Ergebnis erzeugt haben. KNAPP behält sich hier Kontrollen vor.

Du kannst alle Code Teile modifizieren, die Ergebnisdatei (`warehouse-operations.csv`) wird von uns interpretiert und darf daher im Format nicht verändert werden.

UPLOAD WEBSITE

Unmittelbar nach dem Upload erhältst du ein detailliertes Feedback zu deiner Abgabe.

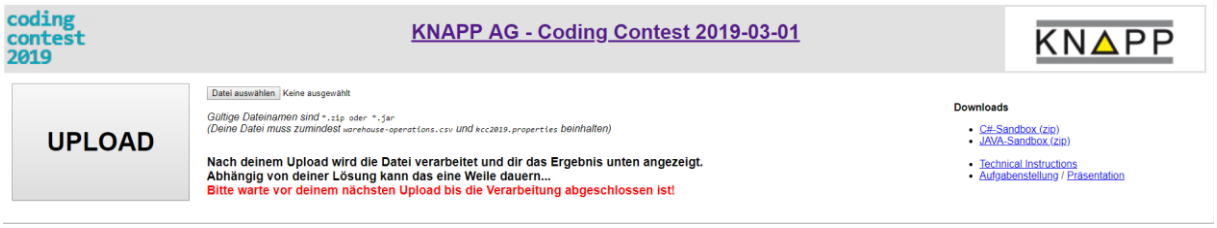


Abbildung 5: Upload-Server (Beispiel)

Im Feedback siehst du die Probleme, die noch in deiner Abgabe vorhanden sind, und deinen Score für diesen Upload. Je *niedriger* dieser Wert ist, desto besser ist das Ergebnis.

Results for *MARIO* (Sonstige) [c#] - file uploaded at 15:11 *(upload2019.zip-10.20.0.2*

(lines read: 50918)

Total Costs (only your operations costs)	[redacted]	
Parse checks	***Count***	
ParsingError	***0***	
InvalidArgument	***0***	
IllegalState	***0***	
Performance results	**Count**	**Costs**
Unfinished Orders	*0*	*0*
Route Sequence Errors	***0***	***0***
Statistics	**Count**	**Costs**
Start Tick	[redacted]	[redacted]
Fetch Product	[redacted]	[redacted]
Pick To Lane	[redacted]	
Pick To Buffer	***0***	
Move From Buffer	***0***	
Release Lane	[redacted]	

Abbildung 6 Abgabe - Details (Beispiel)

TIPPS

- Versuche mit deiner Software das Ergebnis zuerst mit einfachen Strategien zu verbessern und arbeite danach an weiteren Optimierungen.
    - Für eine Lösung ist der Puffer nicht zwingend notwendig.
- Schau dir den von uns vorgegeben Code an und prüfe, ob du Teile wiederverwenden oder erweitern kannst.
- Du kannst alle Teile des Source-Codes verändern. Die Abgabedatei muss jedoch dem vorgegebenen Format entsprechen.
- Lade öfters hoch - es wird nur die beste Abgabe bewertet.

CODE-DETAILS

- Name und Institution setzen
- Einsprungspunkt: *`solution::Solution::Tick()`*
- Die Klassen im Package *`core`* sind KNAPP Hilfsklassen und sind für die Lösung nicht von Bedeutung

KLASSENDIAGRAMM

Im Klassendiagramm ist die Struktur der wichtigsten Klassen dargestellt. In den Klassen selbst sind die wichtigsten Methoden und Attribute dargestellt. Das Diagramm folgt dem Java-Source. In C# sind einige Methoden, dem Stil der Sprache folgend, durch *Properties* ersetzt. Ebenso sind in C# Methodennamen immer großgeschrieben.

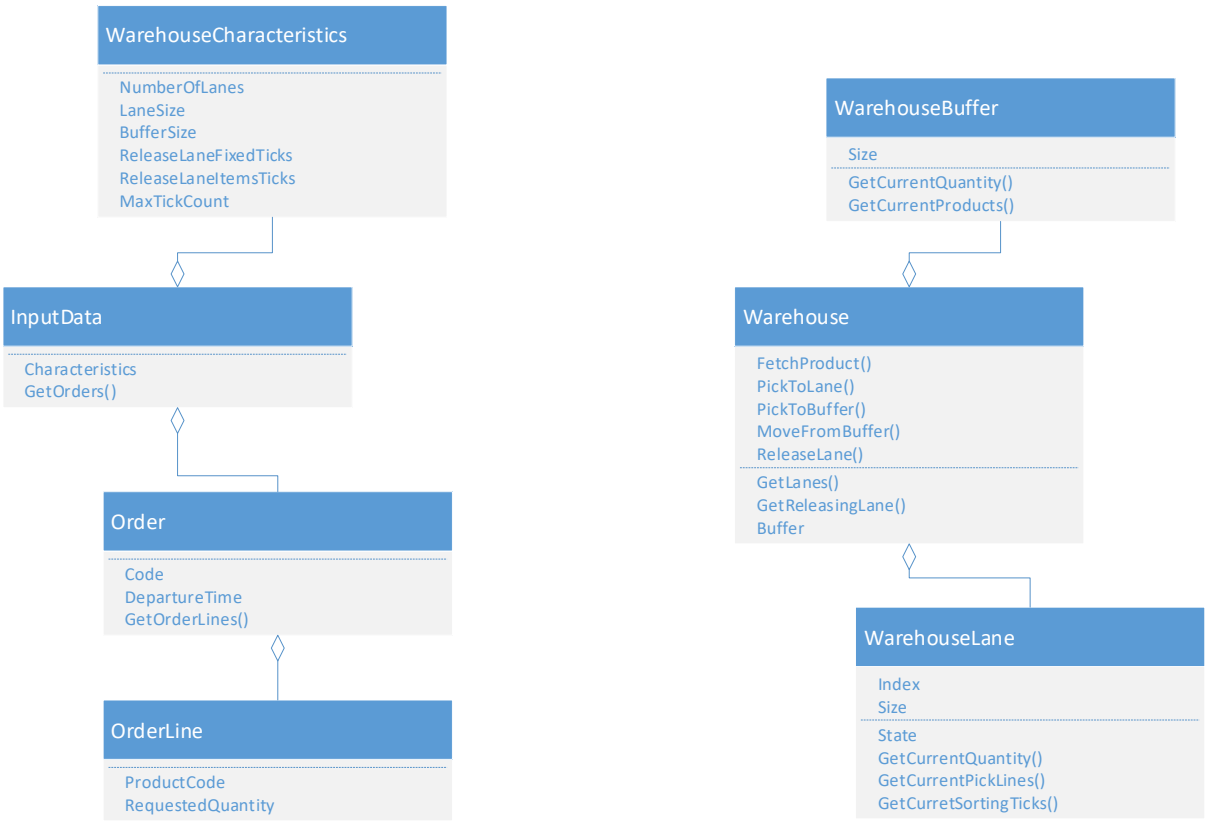


Abbildung 7 - Klassendiagramm