Assignment 1 report

# 

[**Inleiding**](#_2e2626ob7o3j) **1**

[**Code snippets**](#_8p1lgyjr7w1u) **2**

[getLastWagonAttached (Wagon)](#_d4ut98k6o1yb) 2

[attachTail (Wagon)](#_goco6h5kxbol) 2

[reverseSequence (Wagon)](#_wb45pbmdoi1r) 2

[getTotalMaxWeight (Train)](#_acd838xcf2d) 2

[findWagonAtPosition (Train)](#_vyqjexje8ri7) 2

[trainInvariant (Train)](#_liwb2qeb4nop) 2

[splitAtPosition (Train)](#_ug60hx2aljlk) 2

# Code snippets

We zullen 7 stukken code beschrijven en wanneer nodig zullen we ook een verbeterde implementatie neerzetten en de voordelen hiervan in kaart te brengen. Op deze manier kunnen we de code snippets vanuit een complexity & smartness perspectief beschrijven.

## getLastWagonAttached (Wagon)

## 

De methode *getLastWagonAttached* binnen de class Wagon is een handige manier om van een sequentie de laatste wagen op te halen. De stappen zijn simpel en ze vertonen duidelijk de karakteristieken van een Doubly Linked list.

1. We declareren een tijdelijke pointer variabel genaamd tempWagon.
2. We plaatsen in het variabel een referentie naar het huidige object door middel van **this.**
3. We starten een while-loop die controleert of tempWagon een volgende wagon heeft.
   1. Zolang dit het geval is vervangen we tempWagon met de volgende wagon totdat er geen volgende is.
4. Als de while-loop klaar is zal de laatste wagon gelijk zijn aan tempWagon en wordt het variabel teruggegeven.

De gebruikte while-loop is vervangbaar met een functioneel soortgelijke for-loop. Doordat wij de while-loop gebruiken vermijden we dat er een nieuwe variabel geinstantieerd hoeft te worden.

## attachTail (Wagon)

## 

De methode *attachTail* heeft als doel dat we een losstaande wagon kunnen verbinden aan de achterkant. Dit vereist een aantal condities die hieronder beschreven zullen worden.

1. Om te beginnen controleren we of de huidige wagon al verbonden is aan de achterkant en of de tail wagon al verbonden is aan een voorkant.
   1. Zo ja zal er een StringBuilder aangemaakt worden waarin we de sequentie van de voorkant erin plaatsen door middel van een for-loop.
   2. Als de for-loop klaar is geven we een *Exception* terug met als bericht de gemaakte sequentie string.
2. Zo niet zal de volgende wagon ingesteld worden als de gevraagde tail.
3. Om een representatieve invariant aan te houden stellen we de vorige wagon van tail in als de huidige wagon.

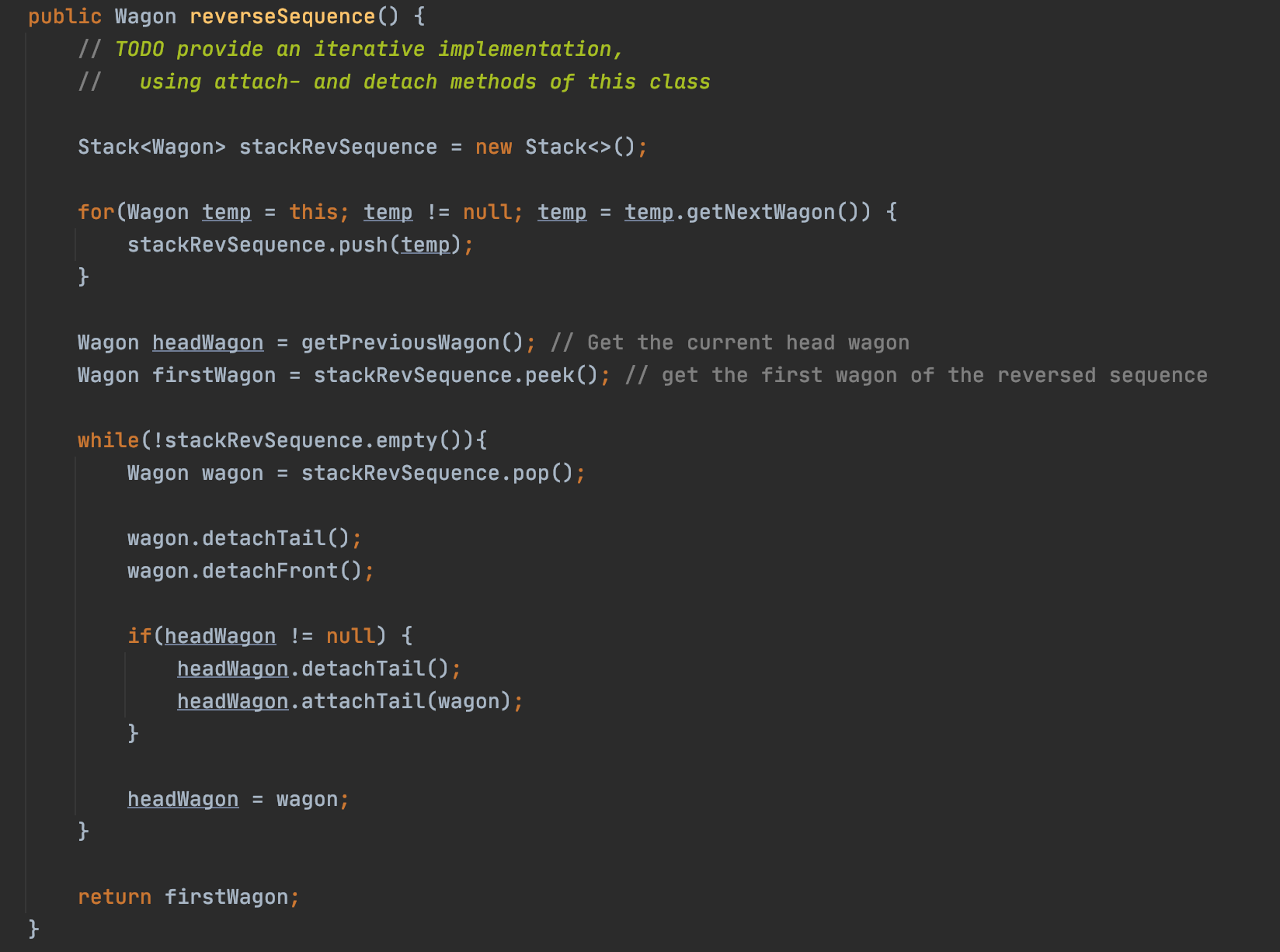
## reverseSequence (Wagon)

## 

De *reverseSequence* methode heeft als doel om een sequentie wagons om te keren. Dit gebeurt vanaf de wagon waar deze methode is aangeroepen. Op eerste ogenblik is het duidelijk dat dit niet de meest efficiënte manier is om dit te doen. Eerst zullen we uitleggen wat er gebeurt in ons originele versie waarna we gaan kijken wat er verbeterd is.

1. In de methode starten we met het ophalen van de belangrijke variabelen
   1. De lengte van de tail waar ook de head ervan wordt bijgeteld
   2. Een arrayList waarin we alle wagons in kunnen plaatsen.
   3. De head wagon waaraan we vervolgens alles kunnen aansluiten
   4. De last wagon als startpunt.
2. In een for-loop beginnen we wagons in de array te plaatsen om verder op in methode weer gebruik van te maken.
3. Daarna halen we de wagons uit elkaar te halen door de verbinding met de voorkant en achterkant te verwijderen.
4. In een volgende for-loop zetten we deze wagons weer omgekeerd aan elkaar door gebruik te maken van de plaatsing volgorde.
5. We geven de eerste element van *reversedSequence* terug omdat dit gelijk is aan de laatste wagon in de vorige tail.

Er zijn verbeterpunten te observeren als we wat langer kijken naar de implementatie en daarom hebben we een verbeterde versie gemaakt. Dit maakt gebruik van de werkingen binnen de Stack API om een omgekeerde sequentie te krijgen.



We hebben hier een aantal veranderingen gemaakt die volgens ons een bijdrage levert aan de complexiteit en leesbaarheid van de code:

* Als we kijken naar deze implementatie zien we dat het aantal variabelen met drie is verminderd.
* De loops zijn aanzienlijk duidelijker en we maken beter gebruik van de karakteristieken van Stack.
* Omdat Stack LIFO hanteert hoeven we niet vanaf het einde tot de huidige wagon te itereren.

## getTotalMaxWeight (Train)

De methode *getTotalMaxWeight* heeft als bedoeling om het totale maximum gewicht op te tellen van alle wagons van het type FreightWagon:

1. We initialiseren een variabel genaamd totalMaxWeight.
2. Om te controleren of we te maken hebben met een FreightWagon gebruiken we *instanceof* op de firstwagon.
3. We itereren door de sequentie wagons
4. Bij elke iteratie voegen we het maximale gewicht van de wagon

Om te controleren of we te maken hebben met een FreightWagon gebruiken we *instanceof* op de firstwagon en als dit het geval is beginnen we de telling door middel van een loop zoals beschreven in vorige snippets. Vervolgens casten we de wagon naar de subklasse FreightWagon om zijn methodes te kunnen gebruiken. Als we te maken hebben met een PassengerTrain zal de telling op nul blijven.

Dit lijkt prima te werken maar we hebben een poging gewaagd om deze methode nog simpeler te maken:



We hebben hier een aantal veranderingen gemaakt die volgens ons een bijdrage levert aan de complexiteit en leesbaarheid van de code:

* Condities plaatsen we bovenaan zodat er niet onnodig geheugen vrijgemaakt wordt op het moment dat er **false** wordt teruggegeven.
* Door de condities bovenaan te zetten en te controleren op negatie hoeven we code niet meer te nesten in een if-statement.
* We maken gebruik van een for-loop omdat de scope van variabel temp onnodig breed was.

## findWagonAtPosition (Train)

## 

De methode *findWagonAtPosition* heeft als doel een wagon te kunnen vinden met als parameter de numerieke positie binnen de sequentie:

1. In de methode starten we met het ophalen van de belangrijke variabelen
   1. Een counter die begint bij 1 (sequentie is niet zero-based).
   2. Een tijdelijke pointer voor wagons binnen de sequentie
2. Vervolgens itereren we door de sequentie met als conditie dat we doorgaan zolang de counter niet gelijk is aan position.
   1. Op deze manier zal tempWagon na afloop van de loop gelijk zijn aan de gevraagde wagon bij position.
3. We geven tempWagon als waarde terug.

Deze methode konden we niet verder versimpelen.

## canAttach (Train)

## 

De methode *canAttach* heeft als doel een wagon te kunnen valideren voor een verbinding met een sequentie. Deze methode zal veelal gebruikt worden om de doubly linked structuur in stand te houden:

1. Als eerst beginnen controleren we of er wagons zijn.
   1. Zo niet, kan er geen reden zijn om verder te valideren op een verbinding
2. We halen vervolgens belangrijke variabelen op:
   1. currentAmount: de sequentie lengte van de huidige trein.
   2. toBeAdded: de sequentie lengte van de nieuwe wagon.
   3. maxWagons: het maximale wagons dat de huidige trein kan hebben
3. De eerste controle gebeurt op capaciteit door de bovenstaande variabelen door te berekenen kunnen we achterhalen of er teveel wagons zijn verbonden.
4. De volgende controle kijkt naar de type wagon en bepaald of het niet dezelfde soort om vervolgens false terug te geven als waarde.
5. Tot slot kijken we of er al een identieke wagon binnen de sequentie te vinden is. Dit doen we met behulp van een hulp methode genaamd *arrayTrain()* waarmee we array operaties kunnen toepassen op de sequentie.
6. Blijkt dit allemaal niet het geval te zijn kunnen we een true teruggeven als waarde en is het mogelijk om een bepaald wagon aan te sluiten op de andere.

## 

## splitAtPosition (Train)

## 

De methode *splitAtPosition* heeft als doel dat de wagon bij een gegeven positie uit de sequentie wordt gehaald en bij een nieuwe trein wordt geplaatst aan de achterkant.

1. Voordat we iets doen controleren we of er wel wagons zijn met hasWagons()
2. We halen vervolgens twee belangrijke waarden op:
   1. splitWagon: De wagon die verwijderd wordt uit de sequentie
   2. prevHead: de vorige wagon van splitWagon
3. Eerst controleren we of we de splitWagon kunnen toevoegen doormiddel van canAttach()
4. We controleren of de wagon een vorige wagon heeft om de verbinding los te maken
   1. Zo niet gaat we hier vanuit dat het een firstwagon is en stellen we dit in als **null**
5. Vervolgens gaan we bij de trein kijken waar de wagon wordt verplaatst en daar beginnen we met een controle of er wel wagons zijn.
   1. Dit bepaald of we de nieuwe head wagon als firstwagon moeten instellen of we de wagon achteraan moeten plaatsen.
6. Als de operatie is gelukt geven we de waarde **true** terug.