Assignment 2 report

# 

[Code snippets 1](#_Toc85378941)

[1. IndexOfByIterativeBinarySearch() 2](#_Toc85378942)

[2. Merge() 5](#_Toc85378943)

[3. Comparators: Product & Purchase 6](#_Toc85378944)

[4. FromLine() 7](#_Toc85378945)

[5. mergePurchasesFromFile() 8](#_Toc85378946)

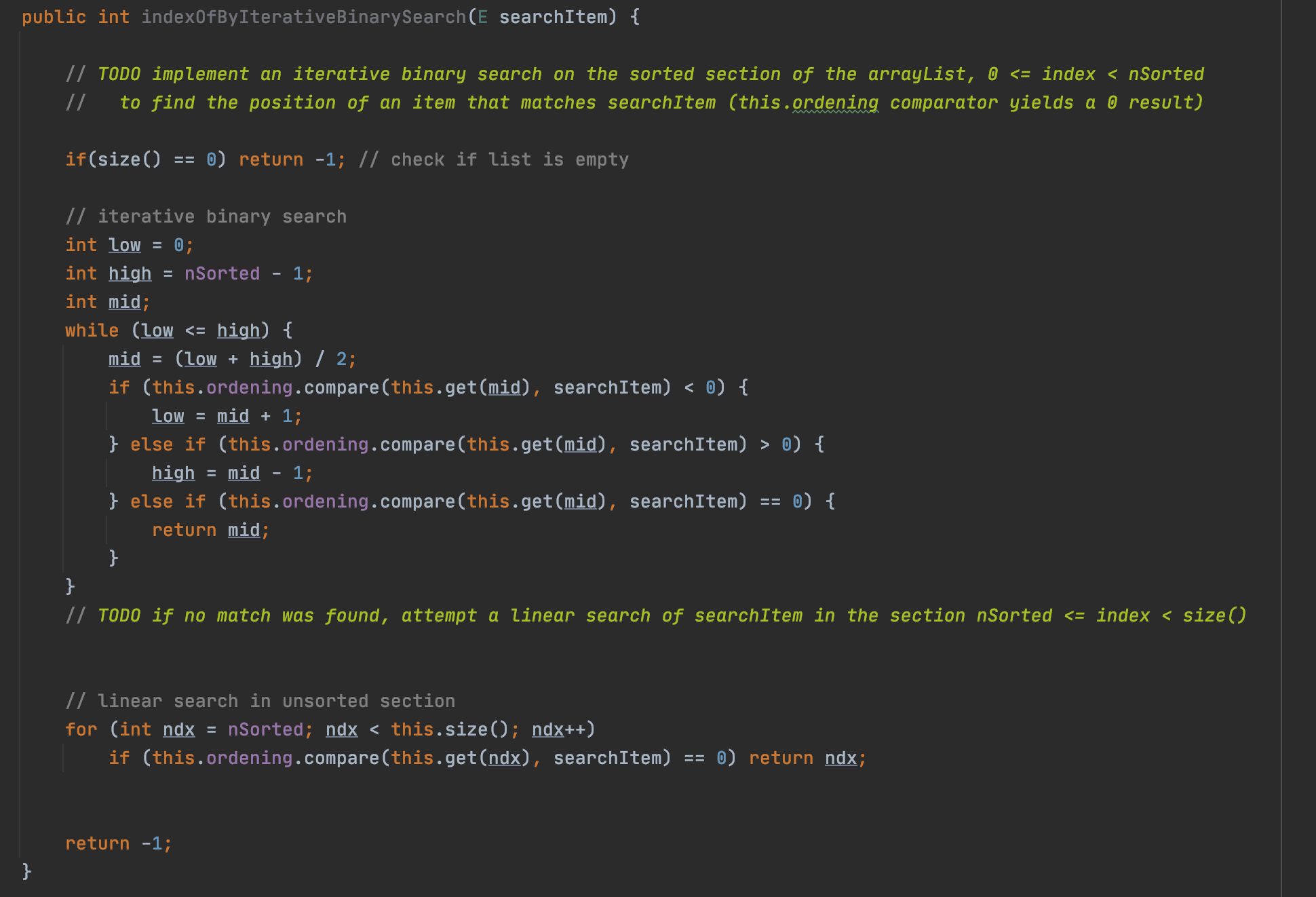
[6. Behoud van invariant 9](#_Toc85378947)

[7. mergePurchasesFromFileRecursively() 10](#_Toc85378948)

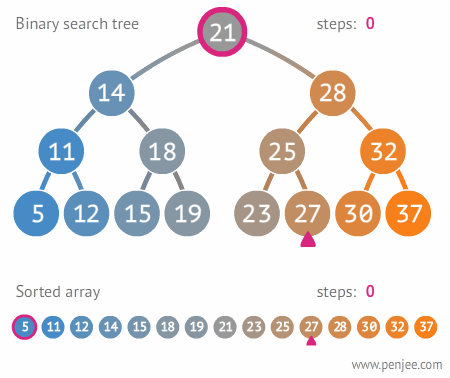
# Code snippets

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.

## 1. IndexOfByIterativeBinarySearch()



De binary search is een efficiënt algoritme dat als doel een gegeven element probeert te zoeken in een gesorteerde sequentie elementen. Om de binary search betrouwbaar te maken is een eis dat de sequentie gesorteerd is. Dit komt door vergelijkingen binnen de algoritme die de volgende handelingen zullen bepalen op basis van de groter dan of kleiner dan operatoren binnen de compare methode.



Wetende dat de correctheid van het resultaat van dit algoritme geheel gebaseerd is op sortering is het van belang dat de comparators die meegegeven worden kloppend zijn (*zie****: Comparators: Product & Purchase***). Het is daarbij handig om te weten dat een vergelijking met een compare methode 3 waardes kan teruggeven:

* -1: Als een element A kleiner is dan element B
* 0: Als element A gelijk is aan element B
* 1: Als element A groter is dan element B

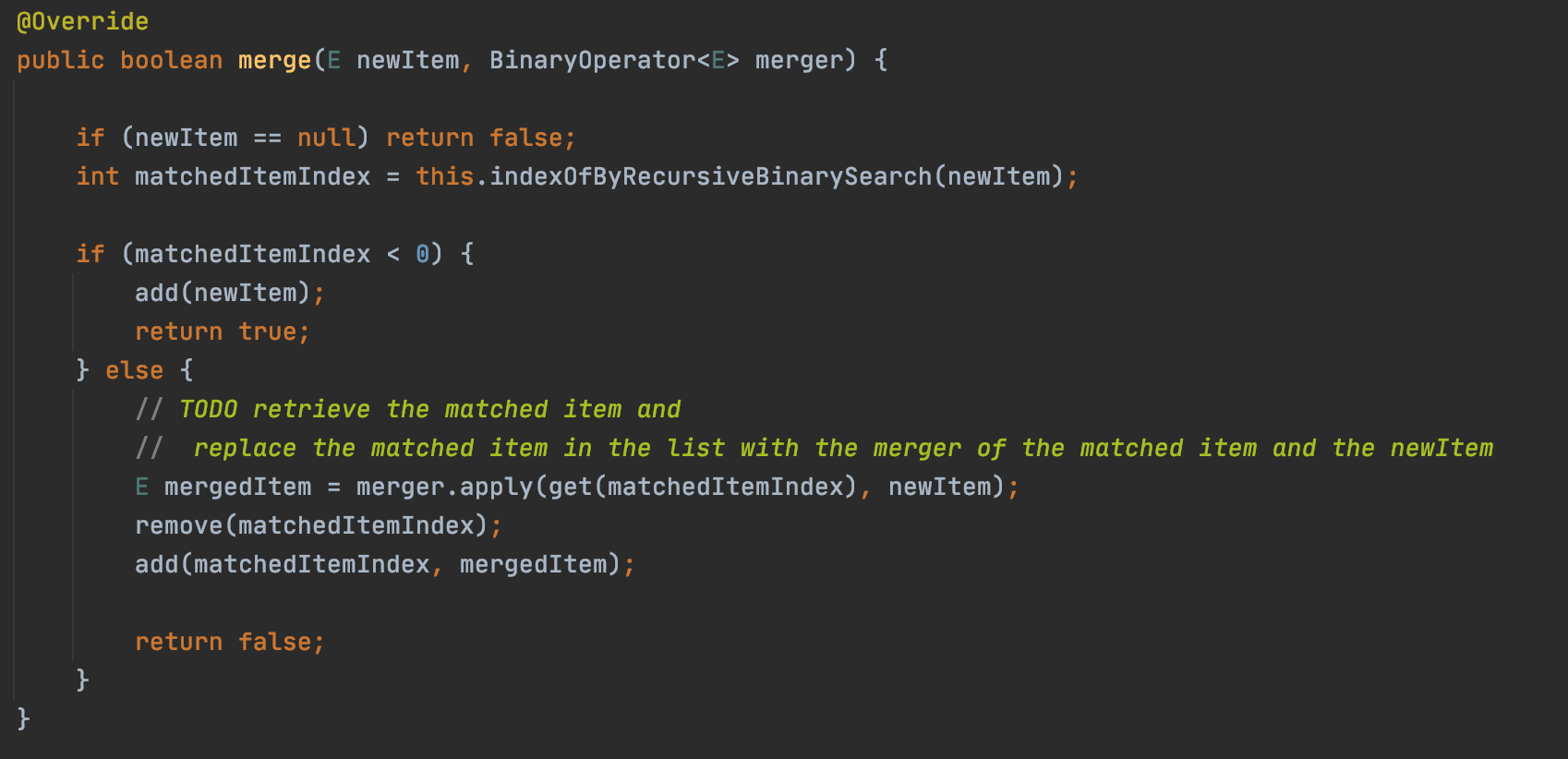
Dat wetende kunnen we beginnen met het beschrijven van de methode:

1. Om te beginnen met de binary search vragen we als argument een **searchItem**
   1. **De searchItem bevat een element van een generic type waarmee we zullen zoeken.**
2. Als eerst controleren we of de array leeg is waarop we een -1 waarde teruggeven.
   1. Er is dan immers geen reden om nog door te gaan.
3. We declareren vervolgens drie variabelen die de algoritme mogelijk zullen maken.
   1. **Low**: het begin punt van de search (meestal nul)
   2. **High**: het eind punt van de search (In dit geval het einde van de gesorteerde sequentie)
   3. **Mid**: een nog leegstaande waarde die gebruikt zal worden om steeds het nieuwe low of high punt te vinden.
4. We starten een while-loop die zal blijven itereren zolang de **low** kleiner is dan de **high**
5. We berekenen direct het middelste punt in de sequentie door de low en high op te tellen en die te delen door twee. Deze waarde plaatsen we in **mid.**
6. Om vervolgens te bepalen waar het element zich kan bevinden maken we gebruik van een aantal condities:
   1. Als de searchItem groter is dan het middelste element.
      1. *Dan wordt de* ***mid*** *waarde toegekend aan* ***low***
   2. Als de searchItem kleiner is dan het middelste element.
      1. *Dan wordt de* ***mid*** *waarde toegekend aan* ***high***
   3. Als de searchItem gelijk is aan het middelste element.
      1. *Dan hebben wij het element gevonden het geven we de waarde* ***mid*** *terug.*
7. Omdat we ons in een while-loop bevinden zullen we dit herhalen totdat het middelste element gelijk is aan searchItem.
8. Als we niet tot een equivalent komen tussen de searchItem en het middelste element. Dan kunnen wij ervan uitgaan dat het element zich niet bevindt in het gesorteerde deel.
   1. Als dit het geval blijkt te zijn dan zullen we een lineaire search doen in het niet gesorteerde deel van de sequentie. Hierop geven we de index bij een juiste vergelijking terug.
9. Als ook hier niks uitkomt dan geven we een -1 waarde terug omdat het element niet gevonden is in de sequentie elementen.

**Time complexity:**

T(n) = T(n/2) + c

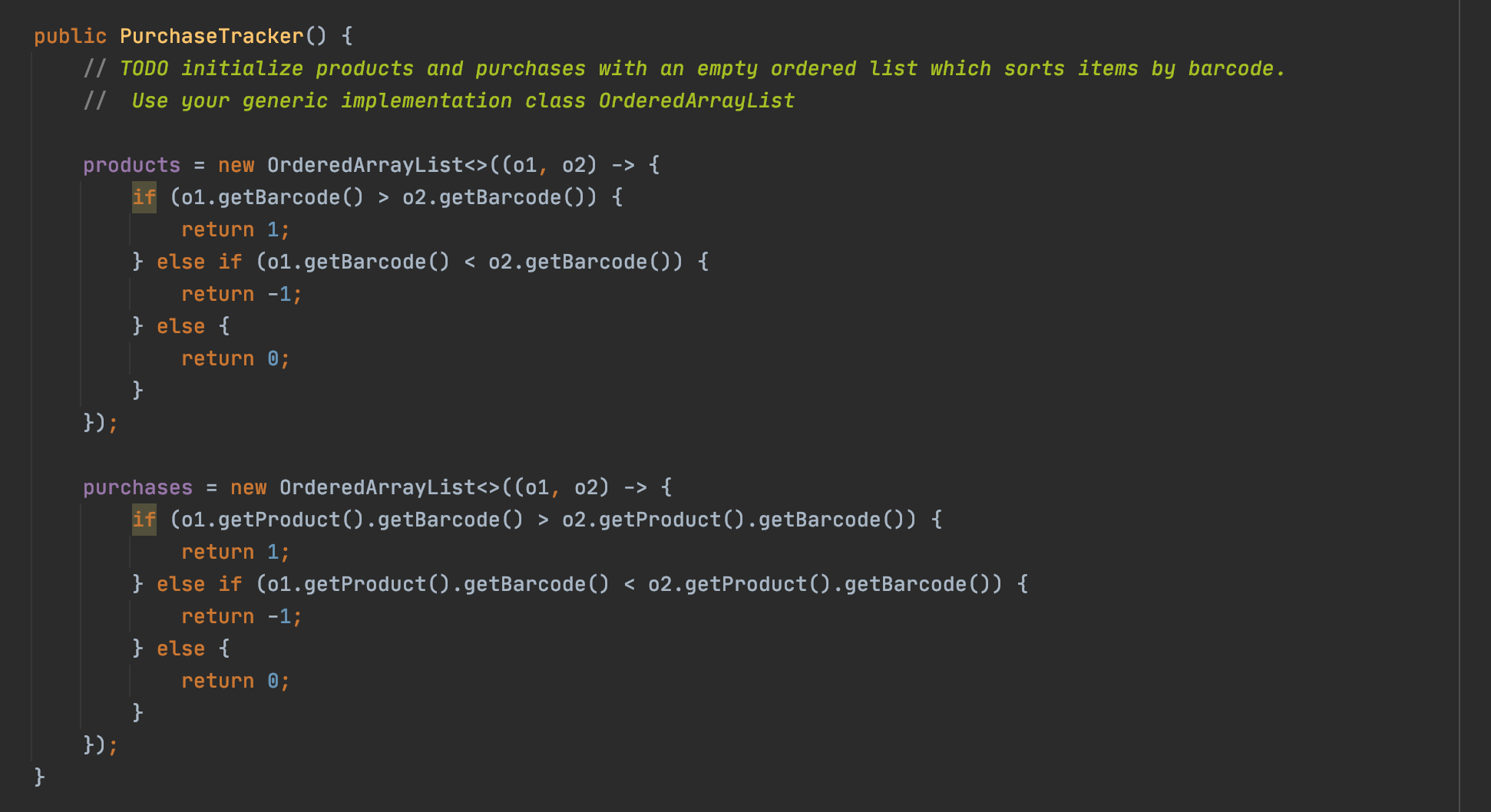
## 2. Merge()



De merge methode heeft als doel een element te verwerken in een sequentie elementen. De **merger** die gevraagd wordt voor deze methode bevat een functie die voor een bepaalde object type een samenvoeging kan waar maken. De **newItem** argument bevat het element dat in de array verwerkt moet worden.

1. De eerste controle is doormiddel van een if-statement. Hierin controleren we of het object niet null is.
2. We halen de index op van newItem.
   1. Als deze -1 is gaan we ervan uit dat het item niet in de array staat en voegen we deze toe aan de array
   2. Als er een positief getal uitkomt dan gaan we ervan uit dat het item zich bevindt in de array.
      1. We gebruiken de merger die meegegeven is om de twee objecten samen te voegen
      2. Het oude object verwijderen we uit de array
      3. Het nieuwe samengevoegde object wordt geplaatst op de positie van het oude object.

## 3. Comparators: Product & Purchase



Voor de PurchaseTracker wordt gevraagd 2 OrderedArrayLists te instantiëren voor *products* en *purchases*. Bij het instantiëren is het van belang een comparator mee te geven omdat we dan gebruik kunnen maken van de binary search en merge methodes. Er zijn meerdere manieren omdat te doen:

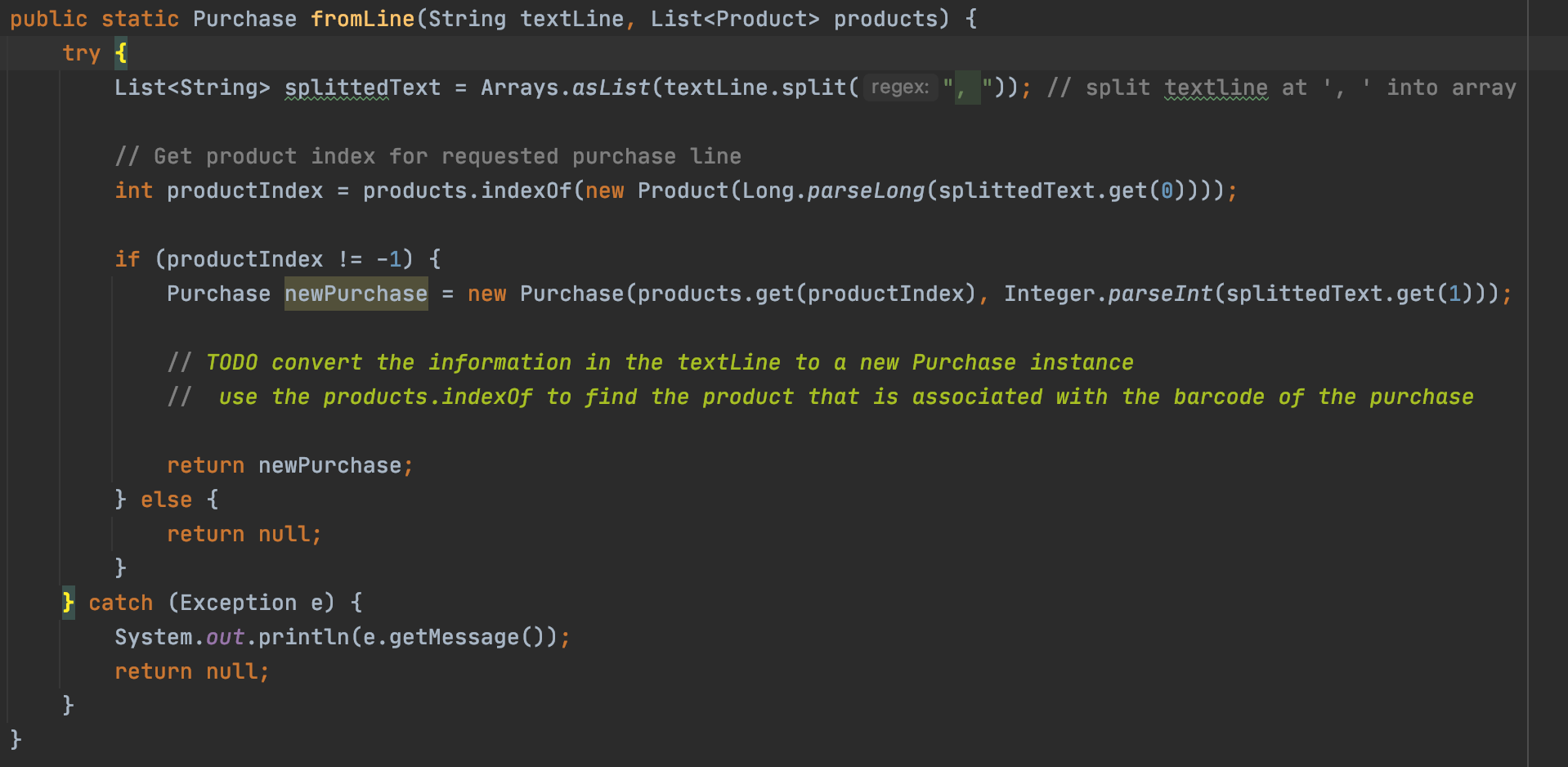
* We implementeren in de class Product & Purchase de interface Comparable
* We geven een in de instantie een comparator mee.

Omdat we niets in de signatuur mogen aanpassen van classes en methodes kiezen we ervoor om in de instantie onze comparator mee te geven. Zoals beschreven bij de iterativeBinarySearch geeft een comparator 3 waardes *(zie:* ***IndexOfByIterativeBinarySearch****).* Deze geven aan of een element groter, kleiner of gelijk is aan een ander element. Het verschil ligt in het type van het object:

* Een product zal zijn vergelijking doen op basis van de barcode.
* Een purchase zal zijn vergelijking doen op basis van de barcode van het product dat in de purchase staat.

Aangezien de merge methode generic is en dat de plaats van de barcode op verschilt zullen we de logica ervan in de instantie moeten beschrijven.

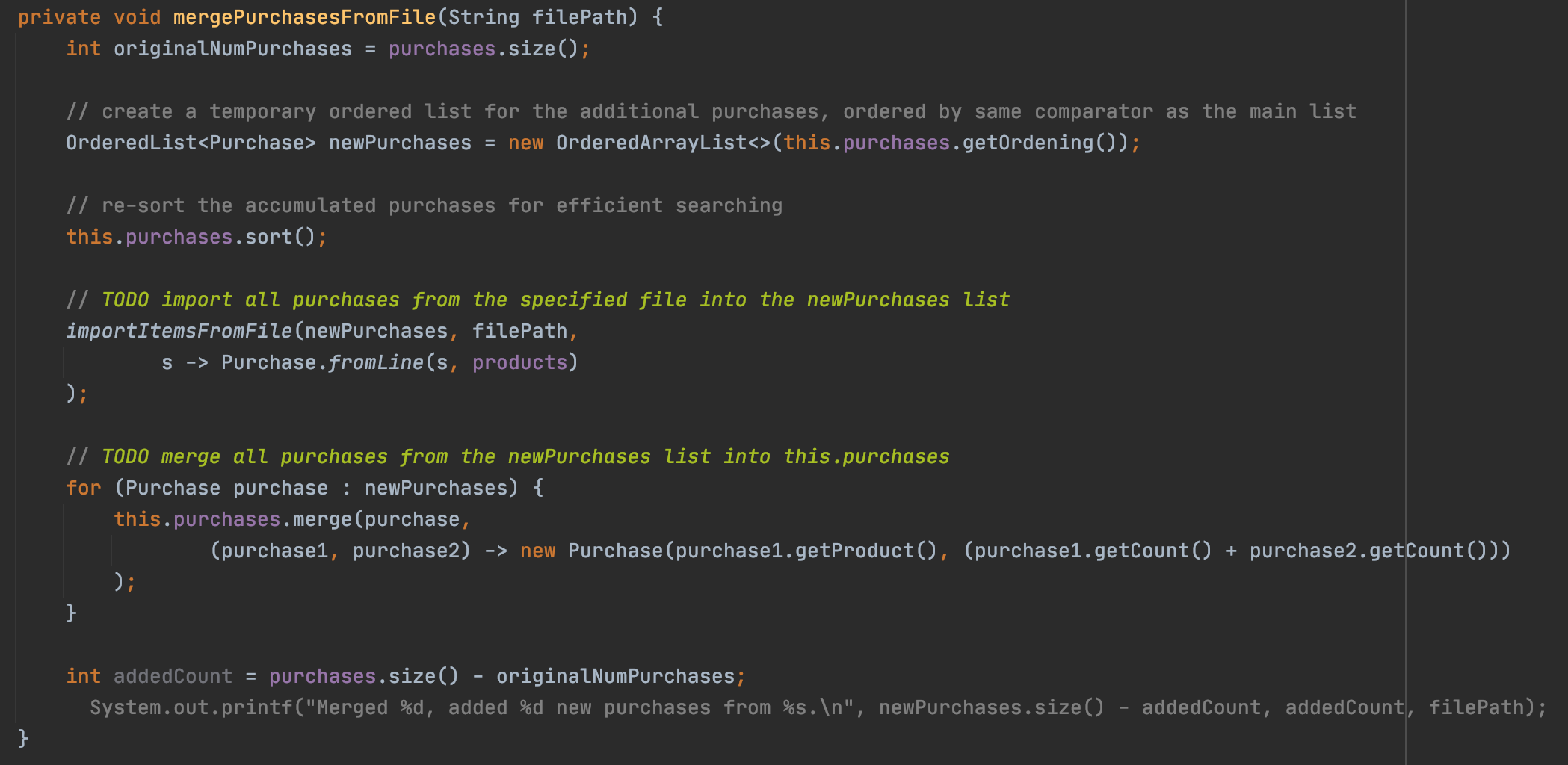
## 4. FromLine()



De FromLine methode heeft als functie het omzetten van een gegeven string naar een bepaald object. Voor dit voorbeeld gebruiken we de Purchase class maar deze methode komt ook voor in de Product class maar dan met een nog net andere implementatie. Het is handig om te weten dat de strings die worden gegeven een bepaald formaat hebben. De waardes worden doormiddel van een komma gescheiden (####,####,####). De eerste waarde bevat een barcode waarmee we het product kunnen vinden en de tweede waarde bevat een getal dat het aantal verkochte producten aangeeft.

1. Als eerst maken we gebruik van de split() functie om de string te scheiden bij de kommas en om te zetten naar een Array. De array zetten we weer om naar een List.
2. We zoeken vervolgens doormiddel van een indexOf naar de index van het product in de meegegeven producten lijst.
   1. Door een dubbele constructor in de Product class hoeven wij alleen de barcode mee te geven.
3. Als de index -1 is gaan we ervan uit dat het product niet bestaat en geven we null terug.
   1. Als de index wel klopt dan maken we een nieuwe Purchase object en geven we die terug.

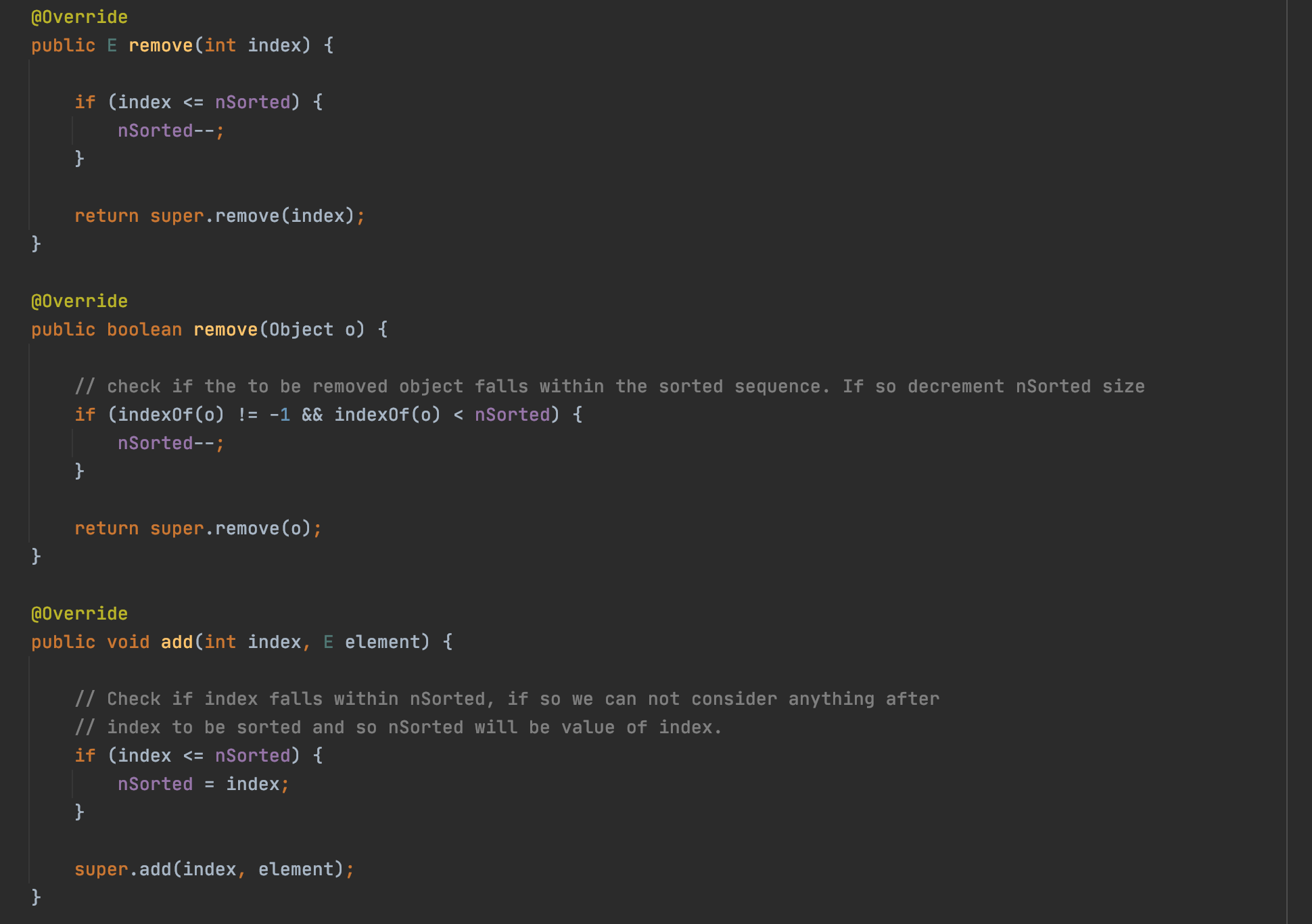
## 5. mergePurchasesFromFile()



De mergePurchasesFromFile heeft als doel het ophalen van producten uit een bestand en die verwerken in de purchases array doormiddel van merge. Hier zien we een aantal onderdelen samenkomen zoals de comparator, fromLine en de merge() methode. We zien ook dat er een sort() functie op de array wordt losgelaten. Dit zal bijdragen aan de snelheid van het zoekalgoritme.

1. We halen de lengte op van purchases array. Dit moet het aantal bestellingen voorstellen.
2. We maken een OrderedArrayList waar in we de nieuw geimporteerde purchases in kunnen plaatsen.
3. Er wordt een sort() methode uitgevoerd om zeker te zijn dat het zoekalgoritme zijn taak betrouwbaar zal doen.
4. We voeren importItemsFromFile uit en geven de newPurchases array, bestandpad en bijhorende fromLine methode mee.
5. We itereren vervolgens door alle purchases in newPurchases en voeren de merge functie uit op de elementen *(zie:* ***Merge()****).*
6. We berekenen de addedCount door de oude lengte van de array af te trekken van de nieuwe lengte van purchases array.

## 6. Behoud van representatie invarianten



Hierboven staan de verschillende methodes in OrderedArrayList waar we het representatie invarianten op verschillende wijze behouden. We zullen ze alle drie stapsgewijs beschrijven en de implicaties daarvan erbij zetten:

***remove(int index)***

1. Als er een element wordt verwijderd doormiddel van een index zal deze methode worden uitgevoerd.
2. We controleren of de index lager is dan nSorted en dus een gesorteerd element verwijderd
   1. Als dit het geval blijkt dan zal de nSorted aantal met 1 verminderd moeten worden.
3. Op deze manier zal de nSorted aantal weer kloppend zijn voor de indexOf methodes

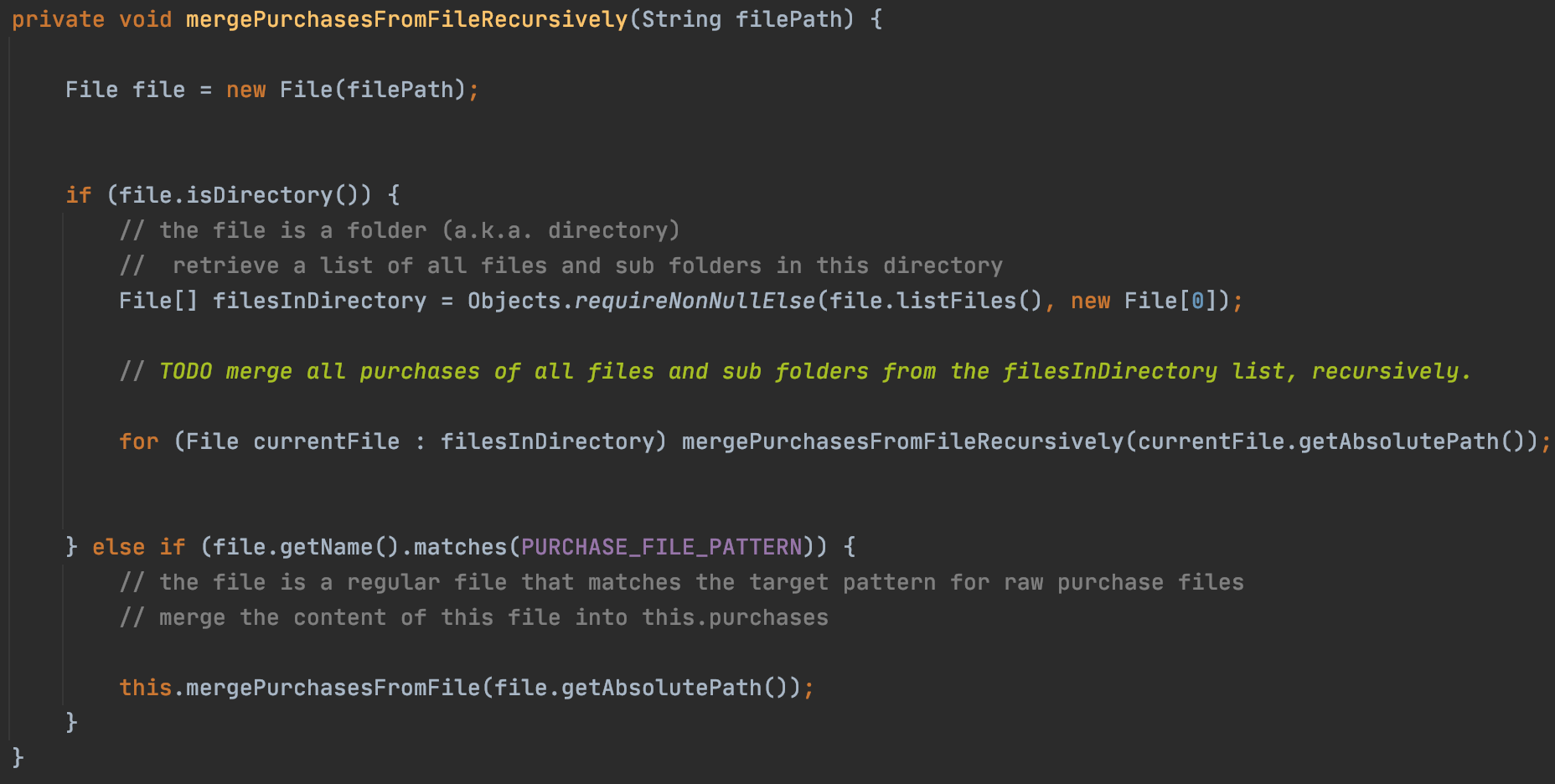
***remove(int index, E element)***

1. Als er een element wordt verwijderd doormiddel van een object zal deze methode worden uitgevoerd.
2. We controleren of het element dat wordt verwijderd wel in de array staat.
   1. Als dit het geval blijkt te zijn dan zal de nSorted met 1 verminderd moeten worden
   2. Als dit niet het geval is dan is er geen reden om de nSorted nog aan te passen omdat het verwijderde element niet bestaat of omdat het in de niet gesorteerde deel van de array staat.

***add(int index, E element)***

1. Als er een element wordt toegevoegd doormiddel van een object en index zal deze methode worden uitgevoerd.
2. We controleren of de index waar het element toegevoegd zal moeten worden lager ligt dan de nSorted.
   1. Als dit het geval is gaan we er nu van uit dat de array tot de gegeven index is gesorteerd. De waarde van nSorted zal nu index worden.

## 7. mergePurchasesFromFileRecursively()



De mergePurchasesFromFileRecursively methode heeft als doel om door een gegeven bestandsstructuur te itereren en te zoeken naar tekstbestanden waarin het relevant data staat. Een bestand kan een directory of tekstbestand zijn. Op basis hiervan zal de methode bepalen of het mogelijk is een File te verwerken of dat we nog door kunnen itereren in de bestandsstructuur.

1. We maken een File object met daarin de verwijzing naar het bestand dat we willen controleren.
2. We controleren in een if-statement of het bestand een folder of een werkbaar bestand is.
   1. Als we met een folder te maken hebben dan zullen we alle onderliggende bestanden omzetten naar een array.
   2. Voor alle bestanden gaan we een recursie starten met de huidige methode.
3. Als we te maken hebben met een werkbaar tekstbestand dan kunnen we **mergePurchasesFromFile (**zie: **mergePurchasesFromFile)** gebruiken om de data in deze bestanden te verwerken.

De recursie maakt het mogelijk alle werkbare bestanden te vinden in bestandstructuren die in diepte variëren.