Zeef van Eratosthenes

[**Waarschuwing!** Teveel *wiskunde* kan schade veroorzaken aan jouw interesse voor de *informatica*.]

De *Zeef van Eratosthenes* (ca. 240 v. Chr.) is een methode waarmee je alle priemgetallen kan bepalen tot aan een zekere bovengrens. Om bijvoorbeeld alle priemgetallen te bepalen tot aan 1000, ga je als volgt te werk:

- Schrijf alle getallen tussen 2 en 1000 op een blad papier. Schrap alle veelvouden van 2. Het eerste getal dat nu nog niet geschrapt is, is 3. Schrap daarom alle veelvouden van 3. Het eerste getal dat nu nog niet geschrapt is, is 5. Schrap nu alle veelvouden van 5, . . .

- We hebben nu reeds 3 priemgetallen gevonden (2, 3 en 5). Door deze methode voldoende lang vol te houden, vinden we uiteindelijk alle priemgetallen tussen 0 en 1000.

Ontwerp een klasse *Zeef* waarmee je bovenstaande techniek kunt toepassen om alle priemgetallen te bepalen tussen 0 en een gegeven bovengrens. Deze klasse moet voldoen aan de volgende specificaties:

- De bovengrens wordt meegegeven bij de constructie van het *Zeef*-object.
- De klasse heeft een methode *bereken()* die de berekeningen uitvoert.
- Nadat de berekeningen zijn uitgevoerd, kan je met behulp van de methode *isPriem(getal)* kijken of het opgegeven getal een priemgetal is (je mag veronderstellen dat dit getal kleiner is dan de vroeger opgegeven bovengrens).
- Er is ook een methode *print* die alle berekende priemgetallen afdrukt, netjes naast elkaar, 10 getallen per lijn, en een methode *aantalPriem* die bepaalt hoeveel priemgetallen er zijn in dit bereik.

Het ligt voor de hand om het blad papier voor te stellen als een lijst of een tabel (array). Een element schrappen (weghalen) uit een lijst of tabel is echter een dure operatie, zeker wanneer een tabel honderdduizenden elementen bevat. Je zal dus een andere techniek moeten gebruiken. *Bezint voor ge begint!*

Verdere tips voor een goede oplossing:

- Om de opeenvolgende veelvouden van een getal te bepalen, hoef je niet te vermenigvuldigen.
- Wat is het laatste getal waarvoor je de veelvouden nog dient te schrappen?

StockManager

[Dit zijn oefeningen **4.56-4.60** uit het boek.]

Open het project *products* en werk de klasse *StockManager* in de volgende oefeningen verder uit. *StockManager* slaat objecten van het type *Product* op in een *ArrayList*. De methode *addProduct* ervan voegt al een product toe aan de collectie, maar de methodes *delivery*, *findProduct*, *printProductDetails* en *numberInStock* moeten nog verder worden uitgewerkt.

Elk verkocht product wordt voorgesteld door een instantie van de klasse *Product*, die een productnummer, de naam van het product en het aantal van dat product in voorraad bevat. De klasse *Product* definieert de methode *increaseQuantity* om toenames van de voorraad van het product te registreren. De methode *sellOne* registreert dat er van het product een exemplaar verkocht is, door het veld *quantity* 1 te verlagen. We hebben de klasse *Product* al voor je gemaakt, en het is niet nodig om die verder aan te passen.

Schrijf eerst de methode *printProductDetails* om ervoor te zorgen dat je de collectie producten kunt bekijken. Druk informatie over elk *Product*-object af door de methode *toString* ervan aan te roepen.

Schrijf de methode *findProduct*. Deze methode moet in de collectie producten zoeken naar een product waarvan het veld id overeenkomt met het ID-argument van deze methode. Als de methode in de collectie een overeenkomstig product vindt, moet de methode dit product retourneren. Als er geen overeenkomstig product te vinden is, moet de methode null retourneren.

De methode *findProduct* verschilt van de methode *printProductDetails*, omdat deze niet noodzakelijkerwijs elk product in de collectie hoeft te onderzoeken voordat het product gevonden wordt. Als bijvoorbeeld het eerste product in de collectie het gezochte ID heeft, kan de methode eindigen en direct het eerste product object retourneren. Aan de andere kant is het ook mogelijk dat het gezochte ID helemaal niet voorkomt in de collectie. In dat geval zal de methode de hele collectie moeten onderzoeken, zonder echter een product te kunnen retourneren. In dit geval moet de methode de waarde null retourneren.

Bij het zoeken naar een overeenkomstig product, zul je de methode *getID* voor een *Product*-object moeten aanroepen.

Schrijf de methode *numberInStock*. Deze methode moet een product met een overeenkomend ID in de collectie zoeken en de huidige voorraad van dat product retourneren. Als er geen product met een overeenkomend ID aanwezig is, moet

de methode 0 retourneren. Deze methode is relatief eenvoudig te construeren zodra je de methode *findProduct* hebt uitgewerkt. De methode *numberInStock* kan bijvoorbeeld de methode *findProduct* aanroepen om het product te zoeken en vervolgens de methode *getQuantity* het resultaat laten retourneren. Vergeet echter niet om ook in een oplossing te voorzien voor het geval er geen product aanwezig is dat aan het criterium voldoet.

Schrijf de methode *delivery* op een vergelijkbare manier als de methode *numberInStock*. De methode *delivery* moet in de lijst producten het product met een bepaald ID vinden en vervolgens de methode *increaseQuantity* aanroepen.

Uitdaging Schrijf een methode in de klasse *StockManager* om informatie over alle producten af te drukken waarvan het voorraadniveau onder een bepaalde waarde ligt (die als parameter aan de methode wordt doorgegeven).

Pas de methode *addProduct* zodanig aan dat het niet mogelijk is om een nieuw product aan de lijst producten toe te voegen als het nieuwe product hetzelfde ID heeft als een product dat al in de lijst aanwezig is.

Voeg aan de klasse *StockManager* een methode toe die een product zoekt op basis van de naam in plaats van op basis van het ID.

```
public Product findProduct(String name)
```