SPO 4: Containerterminal

De haven van Rotterdam werkt als een goed geoliede machine. Vrachtwagens rijden iedere dag af en aan om containers te leveren, die vervolgens op schepen worden geladen om naar hun bestemming te reizen. Dit proces is heel strak ingericht; de aankomsttijd van een schip, de aankomsttijd van de container, douanecontrole; als er ergens vertraging optreedt kan dat enorme gevolgen hebben. Zelfs Rijkswaterstaat wordt ingezet om vrachtwagens veilig op de snelweg te parkeren in het ergste geval!

Containers worden naar een zogenaamde *terminal* gereden (Nee, niet die terminal). Daar worden ze op stapels gezet totdat het schip waar ze voor bestemd zijn, is aangekomen.  
Wanneer een schip aankomt, worden de containers binnen de terminal verzameld, op vrachtwagens gezet en naar de kade gereden om in het schip geladen te worden, zoals op deze foto:

  
*Bron: Havenbedrijf Rotterdam*

Aangezien dingen alléén maar efficiënter kunnen met IT, ben jij erbij gehaald om deze situatie in stappen te verbeteren en vloeiender te maken. Je bent gevraagd om de route van de container binnen de terminal te simuleren.

De keten die een container aflegt, kan grofweg als volgt geschetst worden:

*Producent > Vrachtwagen > Kade > Schip > Bestemming*

Dus, het deel binnen de terminal is alleen:

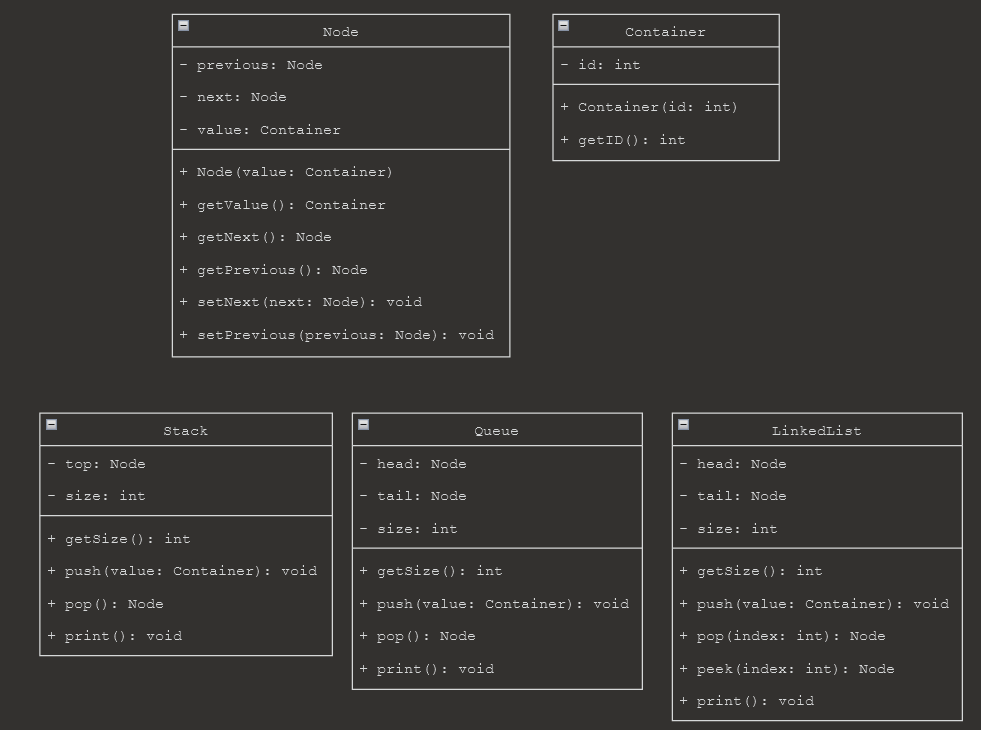
*Vrachtwagen > Kade > Schip*

Met behulp van alle theorie die je tot nu toe gehad hebt over Objecten en Objectgeoriënteerd programmeren, en over Datastructuren, is zo’n simulatie prima te schrijven; zeker als je je code van de afgelopen lessen hergebruikt 😉

De opdracht

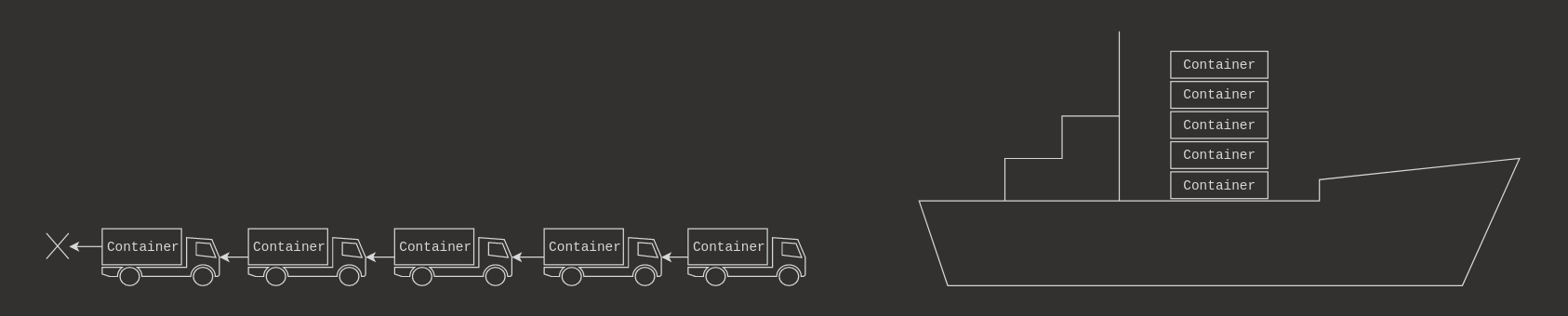
Begin allereerst met je LinkedList, Queue en Stack. Als je de afgelopen lessen hebt meegedaan en het huiswerk gedaan hebt, zou je dit al (grotendeels) moeten hebben.

Om deze opdracht te kunnen doen heb je de volgende classes nodig (en Main/App natuurlijk):  
(Let op: Ook als je de code al had; check even of het overeen komt.)



De oplettende lezer zal merken dat de value in elke Node nu geen String meer is, maar een Container. Pas je code hierop aan op álle plekken!

In dit scenario draait alles om de Container. De Containers moeten in een datastructuur worden gezet, waarna ze 1 voor 1 uit die datastructuur in een andere datastructuur worden gezet. De eerste stelt de kade voor, en de tweede het schip. Kies zelf de correcte datastructuur voor zowel de kade als het schip. Hieronder een schematische weergave van het scenario:



… waarbij het schip leeg is als de kade vol is, en vice versa.

Stap voor stap, binnen je main():

1. Kies logische datastructuren voor de kade en het schip.
2. Vul de kade-datastructuur met 5 Containers, en geef elke Container een uniek ID. Mag random.
3. Pop vervolgens één voor één de Containers van de kade, en push ze op de schip-datastructuur.   
   🡪 Voorkom NullPointerException en voorkom [Dangling Pointers](https://www.wikiwand.com/en/Dangling_pointer)!   
    Hier zal zal het meeste debugging-werk in zitten…
4. Wanneer het schip is volgeladen: print alle Container-ID’s onder elkaar.   
   Check of de volgorde klopt!

Wanneer je klaar bent, kan je verder met deel 2.