Week 1+2: Het Circuit

Een Circuit is een abstracte datastructuur die gebaseerd is op het principe van een kring. Het is een collectie van objecten die in een rij staan waarbij het einde van de rij aansluit op het begin van de rij. Je zit steeds op een bepaalde positie in het Circuit en kunt de volgende operaties uitvoeren:

- `size()`: kijk uit hoeveel objecten het Circuit bestaat;
- `peek()`: kijk welk object er op de huidige positie staat;
- `next()`: zet de positie op het volgende object en kijk welk object dat is;
- `previous()`: zet de positie op het vorige object en kijk welk object dat is.

Je kunt dus door de kring heen lopen. Maar hoe weet je wanneer je alle objecten gehad hebt? Door middel van een marker met de volgende bijbehorende operaties:

- `mark()`: zet de marker op de huidige positie;
- `isMarked()`: kijk of de marker op de huidige positie staat.

Maar hoe bouw je een Circuit op? Via twee operaties die een object toevoegen:

- `insertAfter(E e)`: voeg een object toe na de huidige positie;
- `insertBefore(E e)`: voeg een object toe voor de huidige positie.

Na deze twee operaties staat de huidige positie op het nieuw ingevoegde element.
Er is een project beschikbaar op Blackboard waarin al een aantal dingen voorgekookt zijn.

Opdracht 1

Maak een interface waarin de bovenstaande acht operaties zijn gedefinieerd op een Circuit van generieke objecten. Laat dit tijdens het practicum door de practicumdocent controleren om er zeker van te zijn dat je op de goede weg zit.
In week 2 komt de standaarduitwerking hiervan beschikbaar zodat je daarmee verder kunt werken.

Opdracht 2

Maak een implementatie van het interface Circuit getiteld ListCircuit. Deze implementatie moet intern gebruik maken van een ArrayList om de objecten op te slaan. Er zijn daarbij twee dingen waar je op moet letten:

- Als de marker al geplaatst is en je voegt daarna een object toe, moet de marker naar dezelfde plaats in het Circuit blijven wijzen.
- Hetzelfde object kan meerdere keren in het Circuit voorkomen. De marker kan dus niet naar een object wijzen!

Maak een klasse CircuitException om met uitzonderlijke situaties om te gaan, bv. een operatie op een leeg Circuit.
Er is een unit-test beschikbaar voor de klasse ListCircuit.

Opdracht 3

Maak nu een interface BoundedCircuit dat een uitbreiding vormt van het interface Circuit en modelleert dat het Circuit maar een beperkt aantal objecten kan bevatten (zoals de BoundedStack uit de les). Het heeft twee extra methoden:

- `boolean isFull()`: geeft true wanneer het BoundedCircuit vol is.
- `int getMaxSize()`: geeft de maximale grootte van het BoundedCircuit.

Opdracht 4

Maak een implementatie van het interface BoundedCircuit getiteld ArrayBoundedCircuit. Deze implementatie moet intern gebruik maken van een array om de objecten op te slaan. Er moeten twee constructoren zijn: een default-constructor die een array van 256 elementen maakt, en een constructor waarin de grootte van de array als parameter wordt meegegeven. Als de meegegeven grootte ongeldig is, wordt alsnog de default-grootte (256) gebruikt.
Er is een unit-test beschikbaar voor de klasse ArrayBoundedCircuit.

Opdracht 5 (als je meer dan een 7 wilt)

Voeg een methode moveLeft() toe aan het interface Circuit. Deze methode dient het object op de huidige plaats één plaatsje naar links te verschuiven. De huidige plaats verschuift mee. Voorbeeld (de rechte haakjes geven de huidige positie aan):

12 23 45 [56] 67 78 89

Na het uitvoeren van moveLeft() ziet het Circuit er zo uit:

12 23 [56] 45 67 78 89

Implementeer deze methode zowel in ListCircuit als in ArrayBoundedCircuit.

Opdracht 6 (als je meer dan een 8 wilt)

Implementeer de methode `GnomeSort.gnomeSort()`. Deze methode sorteert een Circuit<Integer> van laag naar hoog op de wijze der tuinkabouters en zet de huidige positie én de marker op het laagste getal in het circuit. Voor uitleg van het algoritme zie http://en.wikipedia.org/wiki/Gnome_sort. Merk op dat je het "verwisselen" in een circuit al gemaakt hebt; zie de vorige opgave. Dat is geen toeval!
Er is een unit-test beschikbaar voor de klasse `GnomeSort`.

Inleveren

De uitwerking moet aan het begin van het practicum van week 3 ingeleverd zijn via de Dropbox van de Blackboard-cursus. Je moet het complete NetBeans-project gezipt inleveren. De practicumdocent zal de uitwerking kritisch nakijken en een cijfer geven tussen 1 en 10. Als je niet of niet op tijd inlevert, krijg je voor straf een **gewijzigde** (extra uitgebreide en/of moeilijke) opgave. Hetzelfde kan gebeuren als je iets inlevert waar naar het oordeel van de docent niet serieus aan gewerkt is.
Als je door bijzondere omstandigheden niet op tijd kunt inleveren, dien je dat **zo vroeg mogelijk** aan de docent te melden; dus niet pas als het inlevermoment daar is.

Succes!