Java Advanced - Week 6: Collections

Opdracht 1

Maak een klasse ***Card*** die een speelkaart zal voorstellen. De klasse moet member variabelen *color* en *value* hebben. Beide variabelen moeten van een *enum* -type zijn:

- enum Color: Club, Diamonds, Heart, Spade
- enum Value: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A

Deze enums krijg je al cadeau. Gebruik zo veel mogelijk functionaliteit uit deze enums.

Maak een klasse Deck die in de constructor een volledig kaartspel aanmaakt en in een Deque steekt. Zorg er voor dat de kaarten in de Deque geschud zijn. (Tip: Bekijk de klasse Collections, misschien vind je wel een manier om de kaarten gemakkelijk te schudden).

De *Deck* klasse moet ook een functie ***getDeckSize()*** hebben, die op elk moment het resterende aantal kaarten in het deck terug geeft.

Maak daarna een functie ***dealCard()*** die de bovenste kaart van de stapel zal verwijderen en teruggeven.

We willen nu een klasse ***Hand*** waaraan we kaarten kunnen toevoegen bij het delen. Bovendien moeten de kaarten in een *Hand* steeds gesorteerd zijn. Zoek zelf uit welke Collection je hiervoor best kan gebruiken.
Het sorteeralgoritme moet als volgt werken: eerst moeten alle harten (heart) zitten, daarna ruiten (diamond) , klaveren (clubs) en tenslotte schoppen (spade). De kaarten van dezelfde 'kleur' (= harten, klaveren, ruiten of schoppen) zijn gesorteerd volgens waarde: 2, 3, …, J, Q, K, A.

Maak een functie ***addCard()*** die een kaart als parameter krijgt en deze in de gekozen Collection toevoegt. Denk er aan dat de kaarten in je hand (en dus in de Collection) steeds gesorteerd moeten blijven. Als je de juiste Collection hebt gekozen, gebeurt dit automatisch als je eenmalig de Comparator functie voor de elementen in de Collection hebt gedefinieerd.

Maak een functie ***showHand()*** die de kaarten in de hand – in gesorteerde volgorde – als een String terug geeft. Een kaart print je als volgt: Klaveren 6 wordt "CLUB_6", Ruiten boer (jack) wordt "DIAMOND_J", Schoppen koningin (queen) wordt "SPADE_Q", Harten koning wordt "HEART_K", etc. De kaarten worden in deze functie gewoon achter elkaar geplakt in een String. Voorgaande vier kaarten zouden dus getoond moeten worden als "HEART_K DIAMOND_J CLUB_6 SPADE_Q" . Maak gebruik van streams waar dat handig kan zijn.

Maak een functie ***hasColor()*** die een *Color* als parameter mee krijgt. Deze functie geeft een boolean waarde terug; *true* als er ten minste 1 kaart met de meegegeven kleur in de hand zit, anders *false*. Maak gebruik van een passende stream functie om deze functie op een compacte manier te schrijven.

Voorzie tenslotte een ***showDeck()*** functie in de *Deck* klasse, die de hele inhoud van het deck toont, op dezelfde manier als een Hand geprint kan worden.

Test je resultaat met de CardTest.

Opdracht 2

Je krijgt de klasse ***Robot*** die een member variabele *name* heeft, die ook als waarde meegegeven wordt aan de constructor. Verder is er ook een inner class ***Command*** die een *action* en een *value* meekrijgt in de constructor. *Action* is hierbij een *enum* die gedefinieerd is in een aparte klasse.

Hier begint jouw werk: de robot moet een lijst van commando's bijhouden. Hierbij moet het commando dat al het langst in de lijst zit, als eerstvolgende uitgevoerd worden. Bepaal zelf welke Collection je hiervoor het best kan gebruiken. Wanneer een commando wordt uitgevoerd, wordt het nadien ook verwijderd uit de lijst.
Maak een functie ***executeNext()*** die dit doet: het 'oudste' commando wordt uit de lijst verwijderd en uitgevoerd (dit doe door de display() functie van het commando aan te roepen). Wanneer er geen commando's meer in de lijst zitten, print je "IDLE" op het scherm.

Maak verder ook een functie ***addCommand()***, waaraan je een Action type en een String waarde meegeeft. In deze functie wordt een nieuw Command object aangemaakt en dit wordt dan aan de lijst met commando's van de robot toegevoegd.

Maak een main functie waarin je een robot aanmaakt, enkele commando's aan hem geeft en enkele keren *executeNext()* uitvoert. Controleer of alles correct werkt.

Opdracht 3

We werken verder aan de vorige opdracht.
Maak een klasse ***PrioBot***. Deze werkt gelijkaardig aan de *Robot* klasse, dus mag daar van afgeleid worden. Veel functies zullen hetzelfde blijven, dus moeten niet overschreven worden. Probeer dus zo veel mogelijk code te hergebruiken uit de *Robot* superklasse.

Zorg er voor dat er in deze afgeleide klasse aan elk commando een prioriteit meegegeven kan worden. Probeer de bestaande Command klasse in *Robot* aan te passen zodat die voor beide soorten robots kan dienen.

Elke keer als *executeNext()* uitgevoerd wordt bij de *PrioBot*, moet het commando met de hoogste prioriteit uitgevoerd worden. Bij commando's met gelijke prioriteit maakt de volgorde van de commando's niet uit. Kies zelf de Collection om dit probleem op te lossen.