Practicum OO – lente 2016 **Opgave 3**

Leerdoelen

Na het maken van deze opgave kun je

- klassen en objecten ontwerpen en gebruiken;
- een programma splitsen in de logica (model) en input–output (view);
- werken met de klassen `String` en `StringBuilder`.

Galgje met klassen en objecten

In deze opgave maken jij en je practicumpartner een implementatie van het beroemde spelletje *galgje*. Doel van het spelletje is dat de gebruiker een woord raadt door steeds een letter te noemen. In het begin zie je alleen maar puntjes in plaats van de letters van het te raden woord (b.v. `........`). Als de geraden letter in het woord voorkomt worden alle voorkomens van die letter getoond (b.v. `.a...aa.`). Als de letter niet voorkomt telt dit als een fout. Zodra de speler te veel foute letters heeft geraden 'hangt' de speler aan de galg en heeft daarmee het spel verloren. Het aantal fouten dat de speler mag hebben varieert per regio. Maximaal 7 tot 10 fouten wordt vaak gebruikt.

Op Blackboard vinden jullie een klasse `WoordLezer`. Een object uit deze klasse zal alle woorden lezen uit een file met de gegeven naam. Op Blackboard staat ook de file `woorden.txt` met een aantal voorbeeldwoorden. De methode `geefWoord` zal een pseudo-random woord uit de gegeven file opleveren. De file met woorden hoort in de basisfolder van je Java package te staan en mag je naar believen veranderen.

We geven deze klasse om te voorkomen dat je nu al zelf met exception handlers aan de slag moet. Bij het openen van een niet bestaande file zal Java een exception gooien om aan te geven dat er iets fout gaat.

Als het goed is zal de IDE deze klasse aan je project toevoegen als je de file in de `src` directory plaatst.

Model-View structuur

Het is altijd een goed idee om in een programma de logica, het echte werk, en de gebruikersinterface zo veel mogelijk van elkaar te scheiden. In Java betekent dit, dat hier verschillende klassen voor gebruikt worden. De klasse die de toestand bevat heet wel het *model*. Het model zal zelf geen enkele interactie met de gebruiker doen. Natuurlijk kan het model ook weer objecten bevatten en willekeurig complex zijn. Denk in dit voorbeeld aan het te raden woord, het aantal keren dat er al een foute letter geraden is, de letters van het woord die tot nu toe goed geraden zijn etc. Gebruik hiervoor de klasse `Galg`.

De *view* zorgt voor interactie tussen de gebruiker en het model. Hierdoor is het eenvoudig om een ander gebruikersinterface te maken voor een programma met een Model-View structuur.

Voor complexere programma's is het vaak een goed idee om de interactie klasse te splitsen in een input klasse, de *controller*, en een output klasse, de *view*. In het algemeen kunnen er ook meerdere controllers en views in een programma zijn. Door al deze taken netjes te verdelen over verschillende objecten blijft het geheel overzichtelijk. We komen hier later in de cursus op terug.

Model: De klasse Galg

Uit bovenstaande beschrijving is blijkt dat de volgende operaties met de klasse `Galg` nodig zijn.

- De huidige status van het spel opvragen aan de galg. De status bevat de juiste letters (de overige letters worden weergegeven als een puntje) en het aantal fouten dat de speler nog mag maken.
- De gebruiker raadt een nieuwe letter. Deze letter moet aan het model gemeld kunnen worden en het model zal zich aanpassen.

- De view moet kunnen achterhalen of het woord nu correct geraden is en of de gebruiker nog door mag gaan met raden.

  Er zijn minstens 3 mogelijke toestanden: **1)** het woord is geraden; **2)** het woord is niet geraden, maar de speler mag nog letters proberen; **3)** het woord is niet geraden en de speler 'hangt'.

Deze klasse wat attributen voor de interne administratie. Zo als in de algemene richtlijnen beschreven zijn deze attributen allemaal **private** .

De klasse `Galg` bevat tenminste twee constructoren: `Galg (String s)` start het spel met het gegeven woord en `Galg ()` zoekt een woord uit de file via de gegeven klasse `WoordLezer`.

String en StringBuilder

Het doel-woord en het woord zover als dat nu geraden is moeten bijgehouden worden als objecten van het type `String` of `StringBuilder`. Een `String` object kun je niet veranderen in Java.

Een object van het type `StringBuilder` kan wel veranderd worden. Deze klasse heeft alle methoden van `String` en een aantal extra methoden. Vooral de methoden `indexOf` en `setCharAt` kunnen heel handig zijn. Zie het www (b.v. `http://docs.oracle.com/javase/8/docs/api/`) voor nadere gegevens, als de JavaDoc die NetBeans je geeft niet voldoende is.

De View in dit spel

De view zorgt voor de interactie tussen de gebruiker en het `Galg` object Het vraagt de gebruiker steeds om de volgende letter en geeft feedback over de pogingen van de gebruiker. Als er geen letters meer geraden mogen of hoeven worden krijgt de gebruiker een mededeling over winst of verlies. Je mag inbouwen dat het spel hierna nog eens gespeeld kan worden met een nieuw woord.

Voor de gebruiker van het programma is het handig als het programma ook steeds verteld welke letters als geprobeerd zijn en niet voorkomen in het woord.

Opdracht

Implementeer galgje volgens de gegeven richtlijnen en test je programma. Let hierbij vooral op de verdeling van taken tussen model en view. Pas dit principe bij alle objectgeoriënteerde programma's toe.

Inleveren

Lever **zondag 21 februari, vóór 23:59 uur**, via Blackboard alle `.java` files uit de package in. Vergeet zeker je naam en die van je practicumpartner en jullie studentnummers niet.