Hertentamen OOAD/Scrum Make IT Work Cohort 13

2018-12-21

Naam	
Studentnr.	
Inlevertijd (door surveillant in te vullen	

Opmerkingen

1. Voor dit tentamen heb je een startproject en een UML diagram nodig. Die zijn allemaal beschikbaar op Moodle. Download die files naar je laptop. Je krijgt dan de volgende files:
    a. SequenceDiagram1-Loterij.uxf
    b. Hertentamen-OOAD-Startproject.zip
2. Je moet voor deze toets ook enkele theoretische vragen beantwoorden. Maak zelf een bestand met antwoorden aan en upload een **pdf** naar Moodle.
3. Upload na afloop drie documenten naar Moodle: Het eclipse project (opdracht 2), een zip-file met de modellen voor opdrachten 1+3+5 en het pdf document met de antwoorden voor opdracht 4.
4. Upload zowel de **pdf** als de **uxf** files van de modellen (voor de zekerheid)
5. Deze toets bestaat uit 4 pagina's.
6. Onderlinge communicatie en communicatie met derden is niet toegestaan, ook niet via het internet.
7. Na afloop van het tentamen dient dit opgaveblad bij de surveillant te worden ingeleverd. Uitwerkingen in Moodle waarbij geen bijbehorend opgaveblad is ingeleverd kunnen niet worden nagekeken!
8. Als je het tentamen inlevert checkt de docent of de upload in Moodle is gelukt. Verlaat het lokaal niet voordat die check is uitgevoerd en je toestemming hebt gekregen!
9. Eindcijfer = 1 + aantal punten / 10

Beoordeling

Opdracht 1: 15 punten
Opdracht 2: 15 punten
Opdracht 3: 20 punten
Opdracht 4: 20 punten
Opdracht 5: 20 punten
Totaal: 90 punten.

Context: Startproject beschrijving

Het startproject bevat een stuk van een Loterij programma waar klanten loten kunnen kopen en controleren of ze een prijs gewonnen hebben. Er zijn 2 soorten loten: *hele* loten en *halve* loten: de halve loten keren alleen de helft van het gewonnen bedrag uit.

Klanten kunnen zich registreren bij de Loterij en kunnen daarna loten kopen. Ook kunnen ze controleren of gegeven een getrokken lotnr zij een prijs gewonnen hebben of niet, en zo ja, hoeveel.

Het startproject bevat alleen een klein (onvolledig!) deel van het programma om het project begrijpelijk en hanteerbaar te houden. Normaalgesproken zou zo'n programma veel meer functionaliteit bevatten.

Er is een aantal mogelijke use cases gedefinieerd voor dit programma:

- registreren van een klant
- kopen van een lot (zowel een heel lot als een half lot)
- doen van een trekking (een winnend (random) lotnr wordt getrokken)
- bepalen van het gewonnen bedrag van een klant

Let op, we maken onderscheid tussen de actor 'Klant' en zijn software representatie (de klasse) die in het Loterij programma gebruikt wordt.

Use case: registreren van een klant

Actors: Klant
Scenario: Een nieuwe klant registreert zich bij een loterij door zichzelf aan te melden. Daarbij geeft de klant zijn naam. De loterij registreert deze klant en geeft deze klant een unieke id. De klant kan deze id gebruiken in het vervolg om zich te identificeren bij de loterij.

Use case: kopen van een half lot/heel lot

Actors: Klant
Scenario: Een klant kan een half lot/heel lot kopen bij de loterij. De klant geeft zijn eigen id. De loterij controleert of de klant geregistreerd is. Zo niet, dan wordt -1 geretourneerd. Anders wordt er een half lot/heel lot aangemaakt en een random lotnr wordt ervoor gekozen. Dit lot wordt bij de klant geregistreerd en het lotnr wordt geretourneerd.

Use case: doen van een trekking

Actors: Klant
Scenario: Een klant kan een trekking doen bij de loterij en daarbij trekt de loterij een winnend lotnr (een random getal). Dit getal wordt geretourneerd. De klant kan later aan de loterij vragen of hij iets gewonnen heeft op dit winnend lotnr.

Use case: bepalen van het gewonnen bedrag van een klant

Actors: Klant
Scenario: Een klant kan opvragen hoeveel geld hij/zij gewonnen heeft gegeven een winnend lotnr. De klant geeft zijn eigen id en het winnend lotnr. De loterij controleert of de klant wel geregistreerd is. Zo ja, dan wordt voor alle gekochte loten van de klant gecontroleerd of het winnende lotnr op dat gekochte lot gevallen is en de gewonnen prijzen (wat voor hele loten anders is dan voor halve loten) worden bij elkaar opgeteld en geretourneerd. Als de klant niet geregistreerd is, wordt -1 geretourneerd.

Moodle

Op Moodle staat een model: een sequence model (van het kopen van een lot door een klant). Het model is beschikbaar in pdf en in UMLet formaat.

Er staat ook een Eclipse startproject op Moodle (`startproject`). In dat project zijn een aantal klassen opgenomen. Hiervan is het meeste afgemaakt. Maar in de meeste klassen zijn er één of meerdere methoden nog niet geïmplementeerd. Deze zijn gemerkt met het commentaar: "OPDRACHT 1: Implementeer deze methode". Er is ook een unit-test gemaakt. Bestudeer de code en zorg dat je het project begrijpt.

De omschrijving van de opdracht en de modellen zijn voldoende gedetailleerd om een werkend project te krijgen. Een voorbeeld unit-test is meegeleverd, gebruik dit om je antwoord te testen!

Tentamen Vragen

Opdracht 1 (15pt)

In `Sequence-Diagram1-Loterij.pdf` is een sequence diagram gegeven van de use-case "kopen van een half lot/heel lot". Implementeer de code voor de bijbehorende methoden `koopHalfLot(…)` **en** `koopHeelLot(…)` in `Loterij.java, Klant.java, HeelLot.java` en `HalfLot.java`. Let op, er is maar 1 Sequence diagram maar deze kun je gebruiken voor beide methoden met een kleine triviale aanpassing.

Opdracht 2 (15pt)

Maak een UML klassendiagram voor het Loterij project. Gebruik daarbij de code na implementatie van bovenstaande use-case als basis. Als je code in Opdracht 1 niet gelukt is, kun je nog relevante informatie uit het Sequence Diagram halen voor het klassendiagram.

- Je hoeft geen klassen te tekenen voor collection klassen uit de standaard bibliotheken van Java (List, HashMap, ArrayList), **maar wel** voor Random!
- Je mag alle getters/setters/toString methodes uit het UML klassendiagram weglaten, die spreken voor zich.
- Vergeet de juiste *associaties*, *cardinaliteiten*, *attributen* en *visibilities* niet!

Opdracht 3 (20pt)

De use case "bepalen van het gewonnen bedrag van een klant" is nog niet gemodelleerd. Modelleer het gedrag van de methode `keerGewonnenBedragUit(…)` in `Loterij.java` in een UML Sequence Diagram. Let er op dat je alles modelleert inclusief aanroepen op de klassen uit de standaard bibliotheken van Java! Gebruik de gegeven implementatie als basis. Licht, indien nodig, je antwoord kort en bondig toe.

Opdracht 4 (20pt) (beantwoord in een apart pdf document)

SOLID is een acroniem voor een aantal *design principles* in object-geörienteerd programmeren om betere code te schrijven.

a) De 'D' in SOLID staat voor *dependency inversion*. Leg in je eigen woorden uit wat deze design principle inhoudt. Wees concreet in je antwoord, illustreer eventueel met een voorbeeld.
b) Waar kun je in het ontwerp van het Loterij project een voorbeeld van dependency inversion zien (in de klassen van Loterij, dus niet in de klassen die bij de standaard bibliotheken van Java horen)? Licht je antwoord kort en bondig toe. Als het er niet in zit, licht toe waar je het principe zou kunnen toepassen in dit project. Wees concreet in je antwoord.

Opdracht 5 (20pt) (geen startproject nodig)

Beschouw een kleine, simpele frisdrank-automaat. De automaat heeft maar ruimte voor 1 flesje. Je kunt de volgende dingen met de automaat doen:

- Een muntje inwerpen – hierna zit er een muntje (geld) in de automaat (als er al een muntje in zat, dan kun je daarna niet 2 flesjes eruit halen; het is een dom apparaat: er zit geld in of niet).
- De automaat bijvullen – hierna zit er een flesje in de automaat (er past maar 1 flesje in de automaat, als er al een flesje in zat gebeurt er bij het bijvullen niets).
- Op de "retourknop" drukken – als er een muntje in de automaat zat, dan krijg je het terug (daarna is de automaat weer zonder muntje); anders gebeurt er niets.
- Op de "uitgifteknop" drukken – als er een muntje en een flesje in de automaat zit, dan krijg je een flesje uit de automaat (daarna is die automaat weer zonder munt en zonder flesje); anders gebeurt er niets.

Maak een UML State diagram voor deze frisdrank-automaat. Bedenk goed wat de "toestanden" zijn en wat de "events" zijn die je van de ene toestand naar de andere toestand doet overgaan. Vergeet niet je "start" en "eind" toestanden aan te geven.