SE2 Beroepsproduct 3 - Object georiënteerd programma

Deze toets test alle programmeervaardigheden die je tot op heden bij SE2 hebt verworven. Het is toegestaan om tijdens de toets de voor dit vak gebruikte boeken, aantekeningen en uitwerkingen (op papier of lokaal op laptop) na te slaan. Iedere vorm van communicatie is tijdens de toets verboden (bijvoorbeeld het gebruik van mail- of chatclients). Bij constatering van bovenstaande wordt de toegang tot de toets ontzegd en hier melding van gemaakt bij de examenkamer.

Duur van de toets De beschikbare tijd voor deze toets is 200 minuten. Merk op dat er geen pauzes ingeroosterd zijn: de eindtijd valt dus niet gelijk met de standaard lesuren. De toets wordt afgenomen vanaf 08:45u en eindigt om 12:05u.

Beoordeling

Bij deze toets moet je laten zien dat je de bij SE2 geleerde OOP concepten beheerst. Deze concepten komen terug in de voor dit deel van het vak opgestelde rubrics. Het uiteindelijke cijfer wordt bepaald na afloop van het verplichte feedbackgesprek. Deze feedback wordt mondeling gegeven en kan gebruikt worden voor het uiteindelijke P-assessment.

De beoordeling van de toets gebeurt op basis van de rubrics die gedefinieerd zijn voor dit beroepsproduct.[1] Je zult een voldoende behalen voor de toets als ieder leerdoel aangetoond is. Hierbij geldt vanzelfsprekend dat hoe beter en correcter het leerdoel in je oplevering verwerkt is, hoe hoger je uiteindelijke eindscore zal zijn.

Het inleveren van de toets

Na afronding van de implementatie pak je de map waarin je je applicatie ontwikkeld hebt in als zip- of rar-bestand. Dit bestand lever je in op Canvas bij de assignment “Toets object georiënteerd programma”.

Structuur van dit document

Deze toets is op te delen in twee grote onderdelen. Het belangrijkste zijn de requirements, te vinden op pagina 4. In deze sectie staat beschreven waar de applicatie aan moet voldoen ten aanzien van de beoordeling. De rest van het document biedt achtergrondinformatie bij het te ontwikkelen programma. Gebruik deze informatie wanneer relevant bij het implementeren van iedere requirement.

[1] Zie voor de rubrics het blokboek van semester 2, hoofdstuk over SE2: https://portal.fhict.nl/Blokboeken/Blokboek%20S%20S2.pdf

Opdrachtomschrijving

Voor deze toets wordt een applicatie ontwikkeld waarmee een bedrijf het vervoer van personen en goederen kan administreren. Telefonisch worden verzoeken geplaatst voor een taxirit of een pakkettransport. Op basis van de in het systeem opgenomen vervoersmogelijkheden, wordt een voertuig geselecteerd waarmee het vervoer verzorgd gaat worden. Wanneer een voertuig weer terugkomt in de garage, wordt het bedrag bepaald dat klanten (achteraf) dienen te betalen. De te ontwikkelen applicatie dient ondersteunend te zijn aan dit proces.

Klassendiagram

In figuur 1 is het domeinmodel te vinden dat gemaakt is op basis van bovenstaande opdrachtomschrijving. Onderstaande opmerkingen lichten afwegingen toe die tot het ontwerp geleid hebben.

Centrale Dit is de klasse waar het formulier mee communiceert. De twee attributen in deze klassen vormen de basis voor de kostenberekeningen: een starttarief wat de basis is voor iedere rit en de prijs per verbruikte liter. Het is mogelijk om de lijst met alle voertuigen op te vragen (*geefAlleVoertuigen*), evenals de voertuigen die aan het rijden zijn (*uitgeredenVoertuigen*).

Het reserveren van ritten is op twee manieren mogelijk: óf het aantal passagiers wordt opgegeven; óf het gewicht en volume van de vracht. Hiervoor zijn de *reserveerRit*-operaties bedoeld. In deze methode wordt uit de lijst van voertuigen een geschikt voertuig gezocht. Als deze gevonden is wordt het kenteken van dit voertuig dat de rit kan verzorgen geretourneerd; wanneer er geen voertuig beschikbaar is wordt de *GeenVoertuigBeschikBaar*-exception opgegooid.

Met *ritAfronden* wordt tot slot een voertuig teruggeplaatst in de garage, en wordt via het kenteken en de afgelegde kilometers de prijs bepaald van het transport. Zie hiervoor ook de requirements voor de later toe te voegen *Rit*-klasse.

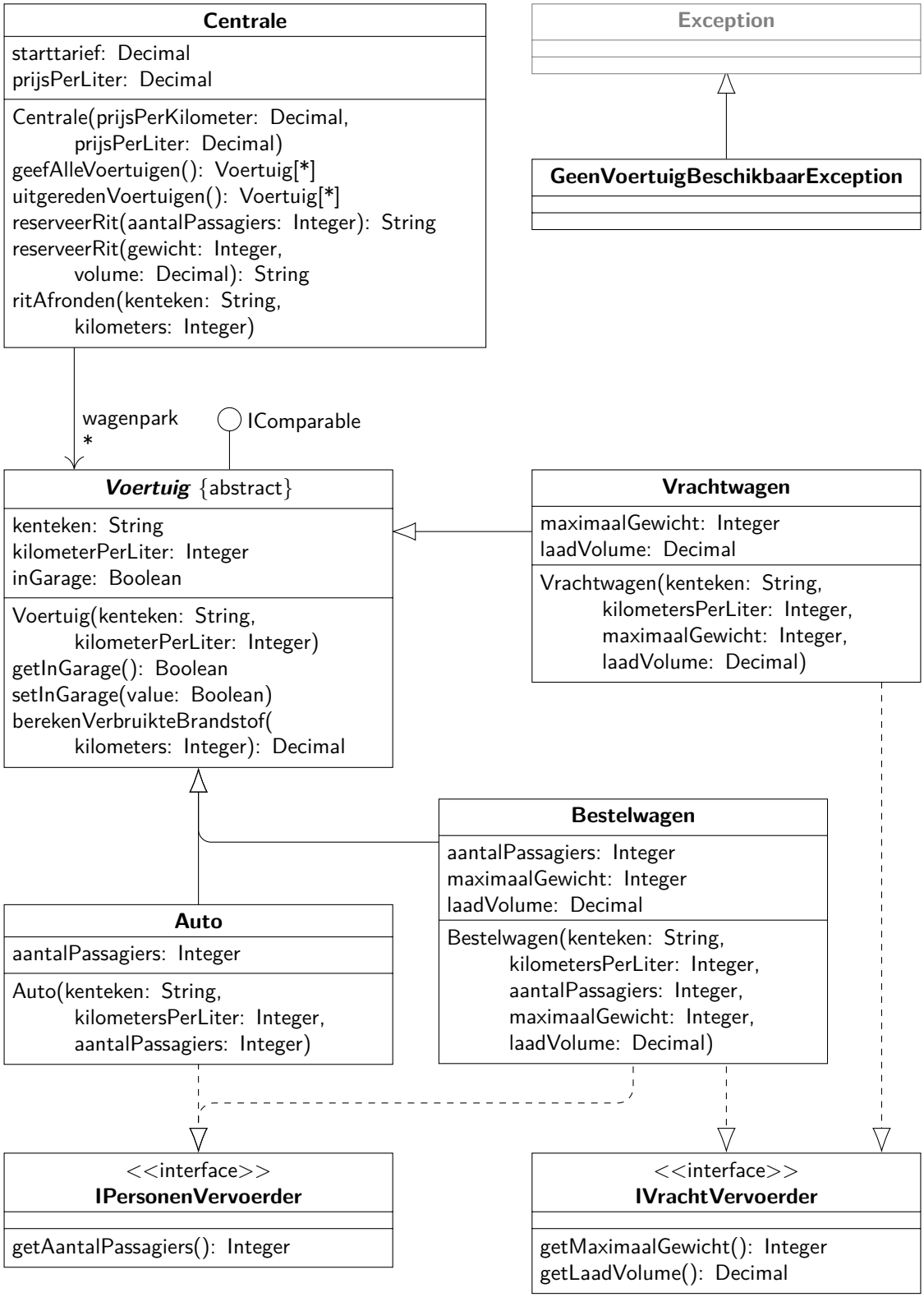
Voertuig Ieder voertuig heeft een *kenteken*, verbruik (*kilometerPerLiter*) en beschikbaarheid voor transport (*inGarage*). De kosten van een rit worden bepaald door de *berekenVerbruikteBrandstof*-operatie aan te roepen. Voertuigen kunnen gesorteerd worden op kenteken. Gebruik de *ToString*-operatie om het kenteken te tonen en ook om weer te geven of een voertuig in de garage staat of niet.

IPersonenVervoerder Deze interface beschrijft een voertuig dat personen kan vervoeren.

IVrachtVervoerder Deze interface beschrijft een voertuig dat vracht kan vervoeren.

Auto, Bestelwagen, Vrachtwagen Deze klassen beschrijven de voertuigtypes die op dit moment in het systeem aanwezig zijn.

GeenVoertuigBeschikbaarException Exception die opgegooid wordt wanneer er een opgevraagd kenteken niet beschikbaar is in de centrale, of wanneer er geen voertuig beschikbaar is voor een rit. De *Exception*-klasse die lichtgedrukt getoond is, refereert aan de reeds bestaande klasse binnen C#: je dient dus alleen de *GeenVoertuigBeschikbaarException*-klasse zelf te maken.



Figuur 1: Een eerste ontwerp van het klassendiagram.

Requirements

Maak een applicatie die voldoet aan de gegeven opdrachtomschrijving. Om je op weg te helpen is er een project gegeven waarin al een formulier is ontworpen. Deze kun je hergebruiken, maar wijzigingen zijn natuurlijk toegestaan. Het is toegestaan om attributen, operaties en klassen toe te voegen en code verplaatsen. Zoals eerder vermeld is het mogelijk dat je naar een onderbouwing van deze wijzigingen gevraagd wordt bij het feedbackgesprek.

Zorg ervoor dat de opgeleverde applicatie de onderstaande functionaliteiten biedt. De genoemde punten staan in willekeurige volgorde: het is geen vereiste om in deze volgorde de implementatie te verzorgen.

- Het klassendiagram dient correct en volledig omgezet te zijn naar code. Verbeteringen mogen doorgevoerd zijn, zolang deze onderbouwd kunnen worden tijdens het feedbackgesprek. De functionaliteit die in figuur 1 gegeven is moet terug te vinden zijn in de uiteindelijke implementatie.
- Bij het aanmaken van de centrale dienen de voertuigen uit een bestand ingeladen te worden. Zie de bijlage voor een overzicht van de data in `voertuigen.csv`.
- De applicatie dient op een logische manier om te gaan met fouten en incorrecte gegevens. Het inlezen van bestanden mag nooit tot een crash leiden. Zorg voor goede exception handling en maak zelf een exception aan waar dit nuttig is.[2]
- Maak unit tests aan voor de operaties in de *Voertuig*-klasse.
- Een nieuwe eis die na het opstellen van het klassendiagram naar voren is gekomen, is dat het mogelijk moet zijn om gereden ritten bij te houden. Maak hiertoe een nieuwe klasse *Rit* aan:
  - Per rit wordt de datum, het aantal gereden kilometers en het voertuig opgeslagen.
  - De rit klasse wordt gebruikt voor het bepalen van de kosten van een rit. Het starttarief wordt aan deze operatie als argument meegegeven, evenals het aantal gereden kilometers.
  - Na het aanmaken van een *Rit*-instantie wordt deze toegevoegd aan de collectie van ritten welke bestaat in de *Centrale*-klasse.
  - Op het scherm moet de verzameling van ritten te zien zijn, waarbij tenminste de datum, aantal kilometers en prijs zichtbaar moet zijn.
  - Ritten zijn sorteerbaar op datum en daarnaast ook op het aantal gereden kilometers.
  - Ritten dienen bij het afsluiten van de applicatie te worden opgeslagen in een tekstbestand.

Zorg er voor dat alle code op de juiste plaats staat. Probeer zo volledig mogelijk te zijn in je uitwerking en neem de moeite om extra's en uitbreidingen toe te passen als je daar reden toe ziet.

[2] Zie *String converteren naar decimal* voor hulp bij conversies van kommagetallen.

Bijlagen

In te lezen voertuigen

De volgende voertuigen dienen ingelezen te worden bij het aanmaken van de centrale. Deze informatie is afkomstig uit het `voertuigen.csv` bestand. Mocht je moeite hebben om dit bestand in te lezen, maak dan in de constructor van de *Centrale*-klasse onderstaande voertuigen aan om verder te kunnen werken.

Type	Kenteken	Km/Liter	# passagiers	Max. gewicht	Laadvolume
Auto	`13-XFB-7`	20	4		
Auto	`77-ZXB-3`	16	4		
Auto	`91-LPB-4`	14	4		
Auto	`96-KTJ-6`	12	6		
Bestelwagen	`VD-329-N`	16	1	856	4.8
Bestelwagen	`5-VDP-40`	13	2	1098	5.2
Vrachtwagen	`VR-40-60`	8		1370	30.0
Vrachtwagen	`VR-43-94`	6		6801	56.0

Tabel 1: Overzicht van de te importeren voertuigen.

String converteren naar decimal

Het kan ook zo zijn dat je bij het inlezen van kommagetallen uit de bestanden onverwachte waarden krijgt. De reden hiervoor is dat in de bestanden, bedragen opgeslagen zijn met een punt als decimaalteken. In Nederland gebruiken we dit teken als scheidingsteken voor duizendtallen. Om dit probleem op te lossen is onderstaand voorbeeld toe te passen bij het inlezen van de tarieven.

```
using System.Globalization;

string s = "123.45"; // Dit kan incorrect ingelezen worden als "12345,00"
Decimal.Parse(s, CultureInfo.InvariantCulture);
```

String splitsen op karakter

Om een string op te splitsen in meerdere onderdelen, is de volgende code te gebruiken:

```
string line = "abc-def-ghi";
string[] parts = line.Split('-'); // Geeft ["abc", "def", "ghi"]
```

String converteren naar enumeratiewaarde

Als je op zoek bent naar een manier om een waarde uit de bestanden om te zetten naar een enumeratie kan onderstaand voorbeeld hergebruikt worden.

```
public enum Kleur { Rood, Groen, Blauw }
Kleur k = (Kleur)Enum.Parse(typeof(Kleur), "Rood");
```