GOP - Opdracht 4
Plaatsen reserveren in een theater

Deze laatste opdracht bekijken we weer het theater, dat we ook al bij OpiJ1 gezien hebben. Het reserveren en kopen van plaatsen gebeurt nu met een grafische interface.

In deze opdracht vragen we u
– de grafische interface te bestuderen en er enkele vragen over te beantwoorden
– een datalaag toe te voegen aan de applicatie die de voorstellingsgegevens en bestaande stoelbezettingen inleest uit een database, en nieuwe bezettingen en klanten ook wegschrijft.

1 De bouwsteen: de domeinlaag

Ook bij deze opdracht is weer een bouwsteen. Deze bevat een volledige implementatie van het theater, inclusief een grafische user interface.
Het ontwerpklassendiagram met de domeinklassen van de domeinlaag wordt getoond in figuur 1.
Dit is niet precies hetzelfde theater als in opdracht 4 van OpiJ1. Er zijn methoden verdwenen, maar ook een aantal methoden toegevoegd.

klasse Theater

Alle operaties die de status van het theater wijzigen lopen nu via de klasse Theater. Het theater bevat een associatie met één voorstelling, namelijk de huidige gekozen voorstelling. Detailinformatie over de verschillende methoden vindt u verder in de code.

klasse Voorstelling

Het toevoegen van een gui maakte het toevoegen van een aantal methoden noodzakelijk (of in elk geval handig) in de klasse Voorstelling.

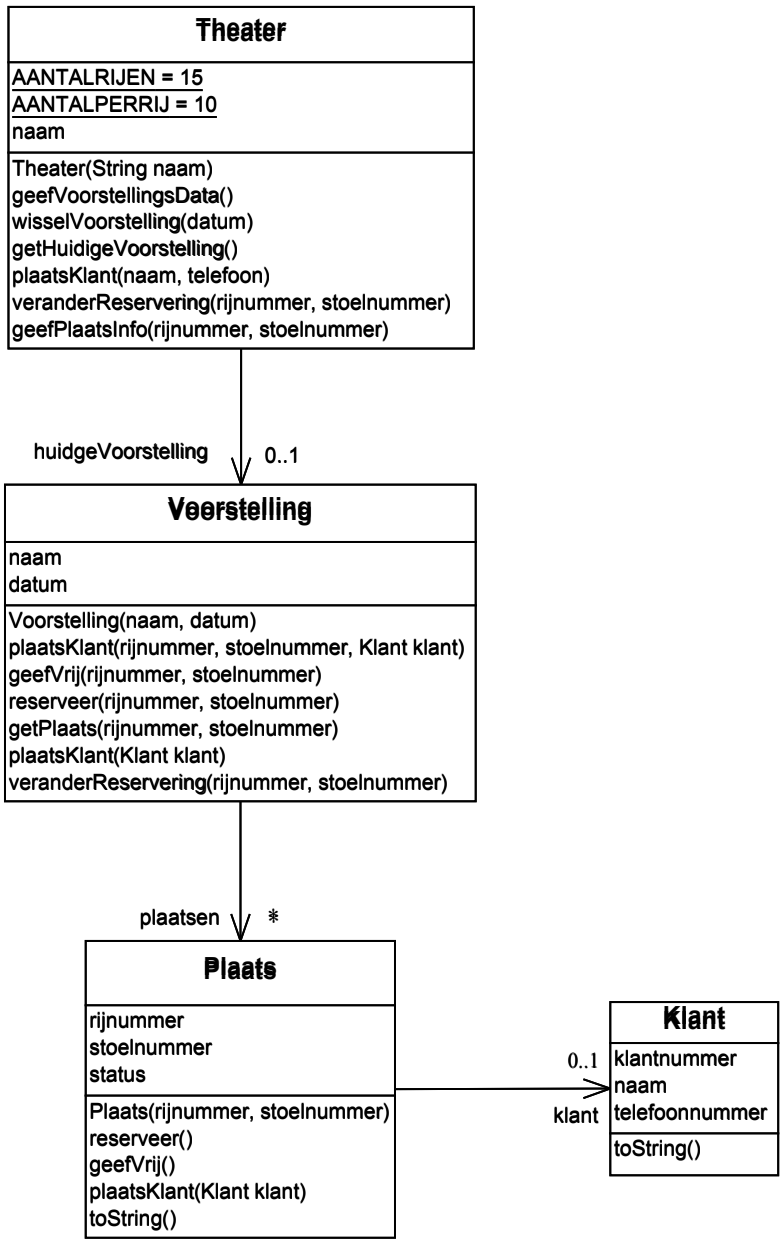
klasse Plaats

De klasse Plaats is nu een subklasse van Observable en waarschuwt zijn Observers bij elke wijziging van de status. Dit wordt uitgelegd in leereenheid 14.

2 De bouwsteen: de datalaag

In de package theaterdata bevinden zich nu twee klassen: Klantbeheer en Voorstellingbeheer, zoals getoond in figuur 2.

Deze klassen nemen een voorschot op een echte datalaag met een achterliggende database.



FIGUUR 1 Het ontwerpklassendiagram van de package theater in de bouwsteen bij deze opdracht (zonder get-methoden)

Klantbeheer

De klasse Klantbeheer beheert een lijst met klanten. Deze klasse heeft een methode geefKlant die een klant teruggeeft met de gegeven naam en het gegeven telefoonnummer. Dat kan een al bestaande klant zijn, maar als zo'n klant er niet was, dan wordt die eerst gemaakt en aan de lijst toegevoegd.

Voorstellingbeheer

Voorstellingbeheer beheert een vast aantal voorstellingen (en wel twee). De methode geefVoorstellingsData levert de data van alle voorstellingen *die nog moeten plaatsvinden*; de methode geefVoorstelling(datum) levert de voorstelling op de gegeven datum (of null als zo'n voorstelling niet bestaat). NB: We gaan er gemakshalve van uit dat er nooit twee voorstellingen op één dag zijn.

Klantbeheer
hoogsteklantnummer ArrayList<Klant> klanten
init() getVolgendKlantNummer() geefKlant(naam, telefoon)

Voorstellingbeheer
ArrayList<Voorstelling> voorstellingen
init() geefVoorstellingsData() geefVoorstelling(datum)

FIGUUR 2 Klantbeheer en Voorstellingbeheer

In deze opdracht dient u deze klassen straks zo aan te passen dat ze gegevens lezen uit en wegschrijven naar een database.

In de datalaag bevinden zich ook twee klassen die nu geen functie hebben maar die u kunt gebruiken bij het implementeren van de databaselaag.

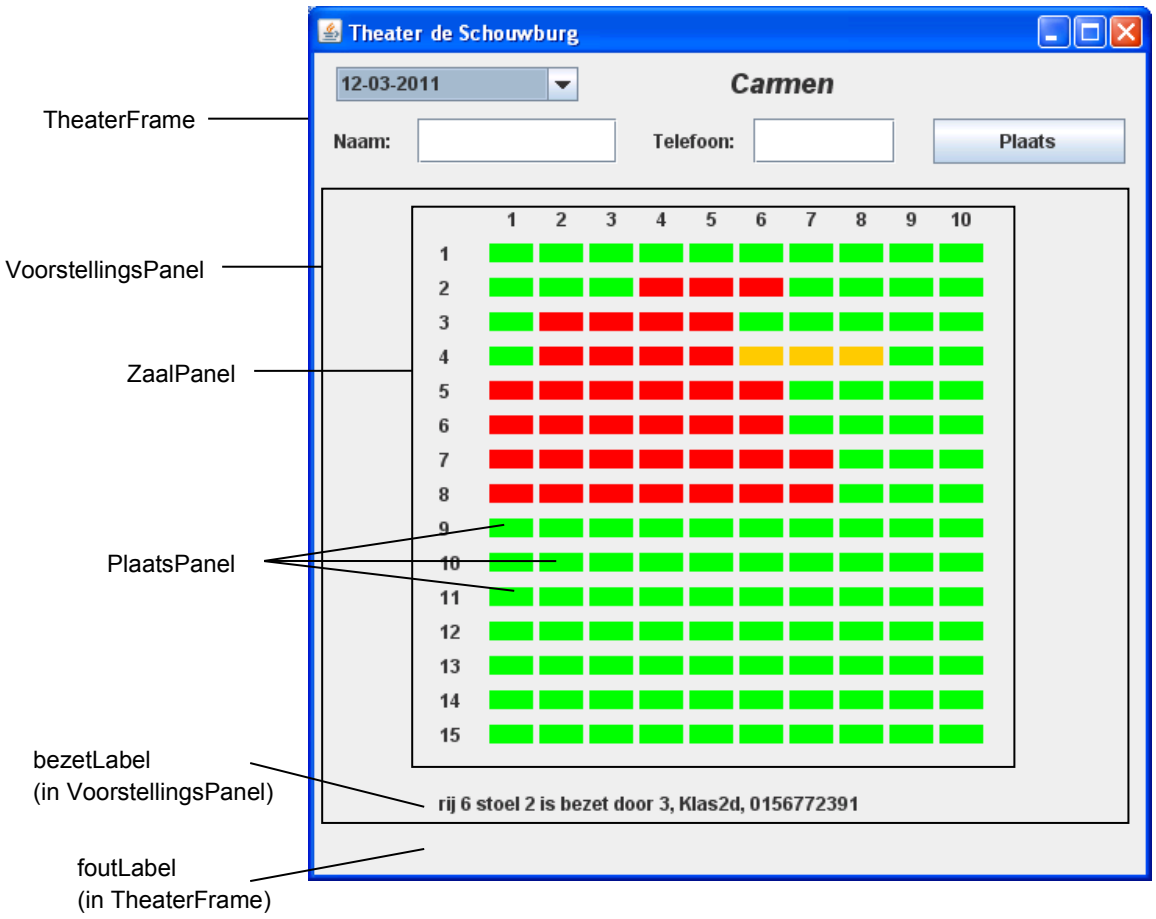
DBConst

De klasse DBConst bevat databaseconstanten voor een MySQL database met de naam theater. De applicatie moet met gegeven constanten kunnen werken.

Connectiebeheer

De klasse Connectiebeheer bevat (lege) methoden om een connectie met de database te openen en te sluiten.

3 De bouwsteen: de grafische interface



FIGUUR 3 Grafische interface voor Theater de Schouwburg

De grafische interface bestaat uit vier klassen als geïllustreerd in figuur 3.

Het hoofdvenster van de interface is de klasse TheaterFrame. De meeste vaste onderdelen daarin zitten in de "kop": een keuzemenu, twee invoervelden, drie labels en een knop. Onderin bevat het frame ook een label foutLabel waarin foutmeldingen geschreven kunnen worden.

Het theaterFrame bevat verder een voorstellingsPanel, dat de reserveringen en de bezetting van de bovenin gekozen voorstelling weergeeft.

Het voorstellingsPanel bevat op zijn beurt een zaalPanel en een label bezetLabel, dat klantgegevens van bezette plaatsen toont.

Het zaalPanel bestaat uit labels voor rij- en stoelnummers plus voor elke plaats in de zaal een plaatsPanel. De kleur van de plaatsPanel geeft de status aan: groen is vrij; oranje is gereserveerd; rood is bezet.

Door klikken op een plaatsPanel wordt de status van de plaats veranderd van vrij naar gereserveerd, en omgekeerd. Wanneer de muis boven een plaatsPanel wordt gehouden, dan wordt de statusinformatie van de plaats getoond in het bezetLabel.

Wanneer bovenin de gegevens van een klant worden ingevoerd en er op de knop Plaats wordt geklikt, dan wordt die klant geplaatst op alle gereserveerde plaatsen. Je kiest dus eerst plaatsen uit en voert dan pas de klantgegevens in.

– Start de klasse TheaterFrame als applicatie en experimenteer met de interface, tot u begrijpt hoe die werkt. Wissel ook van voorstelling (en weer terug).
NB: Er wordt niets opgeslagen.
– Bestudeer de code en maak dan opdracht 4.1 over de event handling in deze code.

OPDRACHT 4.1

De code voor de event handling bij het (de)selecteren en aanwijzen van plaatsen staat in de klasse VoorstellingsPanel, die een binnenklasse MuisLuisteraar bevat.

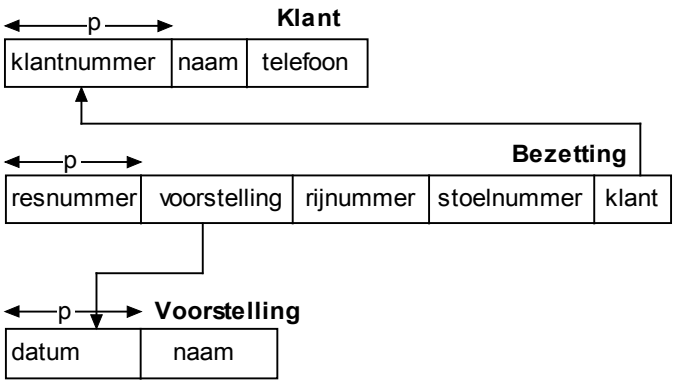
a Implementeert de klasse MuisLuisteraar de interface MouseListener? Licht dit toe.

b Waar in de code wordt er een instantie van MuisLuisteraar gekoppeld aan een component? Welke component is dat?

c Het was ook mogelijk geweest om de binnenklasse MuisLuisteraar op te nemen in de klasse PlaatsPanel. Bij constructie van een nieuw PlaatsPanel wordt dan een instantie van die binnenklasse gemaakt en als luisteraar gekoppeld aan dat PlaatsPanel. Om efficiency redenen is hier niet voor gekozen, hoewel de code iets eenvoudiger zou zijn. Licht dit toe.

4 De databaselaag

Het belangrijkste onderdeel van deze opdracht is het uitbreiden van het programma met een databaselaag.



FIGUUR 4 Strokendiagram database theater

Figuur 4 toont het strokendiagram van de database, die we zo eenvoudig mogelijk hebben gehouden (onrealistisch eenvoudig, maar anders wordt de datalaag te groot).

De volgende tabel toont de gebruikte datatypes. De sleutels zijn vet gedrukt.

Tabel	Veld	Datatype
Klant	**klantnummer**	INTEGER
	naam	VARCHAR(45)
	telefoon	VARCHAR(45)
Bezetting	**resnummer**	INTEGER (AUTOINCREMENT)
	voorstelling	DATE
	rijnummer	INTEGER
	stoelnummer	INTEGER
	klant	INTEGER
Voorstelling	**datum**	DATE
	naam	VARCHAR(45)

Bij de bouwsteen zit een databasescript theaterScript.sql voor een database met de gegeven structuur. Deze database bevat een aantal klanten en voorstellingen.

Belangrijk!

Voor u aan opdracht 4.2 begint, moet
– MySQL geïnstalleerd zijn.
– de database theater is gecreëerd met behulp van het databasescript theaterScript.sql. Geef de database de juiste rechten (gebruiker cppjava, wachtwoord theater).

OPDRACHT 4.2

Ontwerp en implementeer nu een datalaag, met de volgende functionaliteit:

Done – Bij het starten van de applicatie wordt er connectie gemaakt met de database.

Done – Bij het afsluiten van de applicatie wordt deze connectie weer afgesloten.

– Bij het selecteren van een nieuwe voorstelling wordt de zaalbezetting van die voorstelling ingelezen en getoond in de grafische user interface.

– Wanneer een klant geplaatst wordt, wordt de bijbehorende bezetting meteen naar de database geschreven. De klant zelf wordt alleen in de database opgenomen als het een nieuwe klant is (die nog niet in de database stond).

– Een fout (zoals een SQLException) leidt tot een voor een gebruiker begrijpelijke foutmelding in de grafische user interface (gebruik hiervoor het label foutLabel in TheaterFrame).

Aanwijzingen en stappenplan

– Definieer een klasse TheaterException. Instanties hiervan worden vanuit de datalaag opgegooid en afgehandeld in TheaterFrame

– U mag alle methoden van de klasse(n) in de datalaag statisch maken.

– Beide beheersklassen bevatten een methode init. Hierin kunt u alle benodigde initialisaties uitvoeren.

– Bij het SQL type DATE hoort het Java-type java.sql.Date. Conversie van een variabele datum van type GregorianCalendar naar het type java.sql.Date gaat als volgt:

```
java.sql.Date sqlDatum = new java.sql.Date(datum.getTimeInMillis());
```

De omgekeerde conversie gaat als volgt:

```
GregorianCalendar datum = new GregorianCalendar();
datum.setTimeInMillis(sqlDatum.getTime());
```

– Implementeer en test stap voor stap. We suggereren het volgende stappenplan:

a. Zorg eerst dat u connectie kunt maken met de database door de methoden van de klasse Connectiebeheer te implementereren. Done De benodigde constanten zijn gegeven in de klasse DBConst. Geef de klasse een methode main en test daarin of u inderdaad de verbinding kunt maken en weer sluiten.

b. Done Zorg dat bij het starten van de applicatie de database geopend wordt.

c. Pas de methode geefVoorstellingsData van de klasse Voorstellingbeheer aan, zodat deze nu alle data van voorstellingen inleest uit de database. Test het inlezen vanuit een methode main in Voorstellingbeheer.

d. Pas de methode geefVoorstelling van de klasse Voorstellingbeheer aan. Deze klasse leest alle gegevens van gevraagde voorstelling (naam en bezetting) uit de database en maakt daarmee een voorstellings-object.
U mag voorlopig voor iedere ingelezen plaats die bezet is, een nieuwe instantie maken van Klant voor de klantgegevens van die plaats.
Het attribuut voorstellingen is nu niet meer nodig. Verwijder dit.

e. Zorg dat bij het starten van de applicatie de data van de voorstellingen (voor zover in de toekomst) getoond worden in het keuzemenu in TheaterFrame en dat bij het kiezen van één van die voorstellingen, ook de naam en de bezetting getoond worden. Denk hierbij ook aan het afhandelen van de TheaterExceptions.
f. Pas de methode geefKlant in Klantbeheer aan. Deze methode levert een Klant-object op op grond van een naam en een telefoonnummer. Deze methode moet er ook voor zorgen dat de klant, als die nog niet in de database aanwezig was, aan de database wordt toegevoegd. Zit de klant echter wel al in de database, dan mag die niet nogmaals worden toegevoegd.
Voor nieuwe klanten dient een klantnummer gemaakt te worden. Pas daartoe de methode geefHoogsteKlantnummer aan. Deze methode dient in de database te zoeken wat het hoogst uitgegeven nummer is, en daar één bij op te tellen. U kunt het maximum bepalen met het SQL-statement:

```
SELECT MAX(klantnummer) FROM klant
```

Test weer vanuit een methode main in de datalaag of alle toegevoegde methoden het juiste resultaat opleveren. Attributen hoogsteKlantnummer en klanten kunnen nu verwijderd worden uit Klantbeheer.
g. Voeg een methode toe aan Voorstellingbeheer die een nieuwe bezetting kan wegschrijven. Parameters van deze methode zijn de datum van de voorstelling, het rijnummer en stoelnummer en het klantnummer van de klant. Test weer eerst vanuit main.
h. Zorg er voor dat een nieuwe bezetting, gemaakt in de gui, in de database wordt opgenomen.
i. Zorg ervoor dat bij het sluiten van de applicatie, de connectie met de database gesloten wordt.
j. Test nu de hele applicatie. Maak nieuwe bezettingen en wissel daarbij van voorstelling. Sluit de applicatie af, open hem opnieuw en kijk of de bezettingen nog te zien zijn.

5 Inleveren

Lever het volgende in:
– Een word- of pdf-document met de uitwerking van opdracht 4.1, plus een toelichting op uw ontwerp van de databaselaag.
– Het gezipte project.