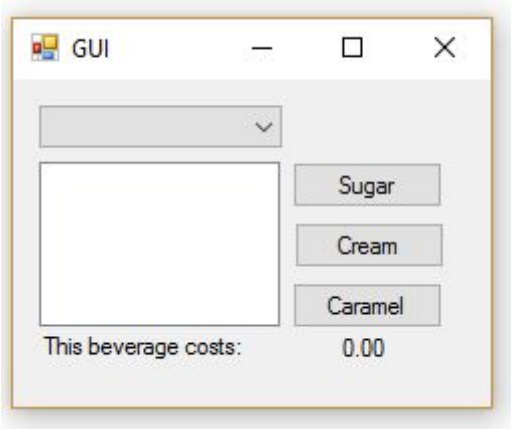
# Decorator Pattern Assignment

Tim Ackermans & Dominic Voets



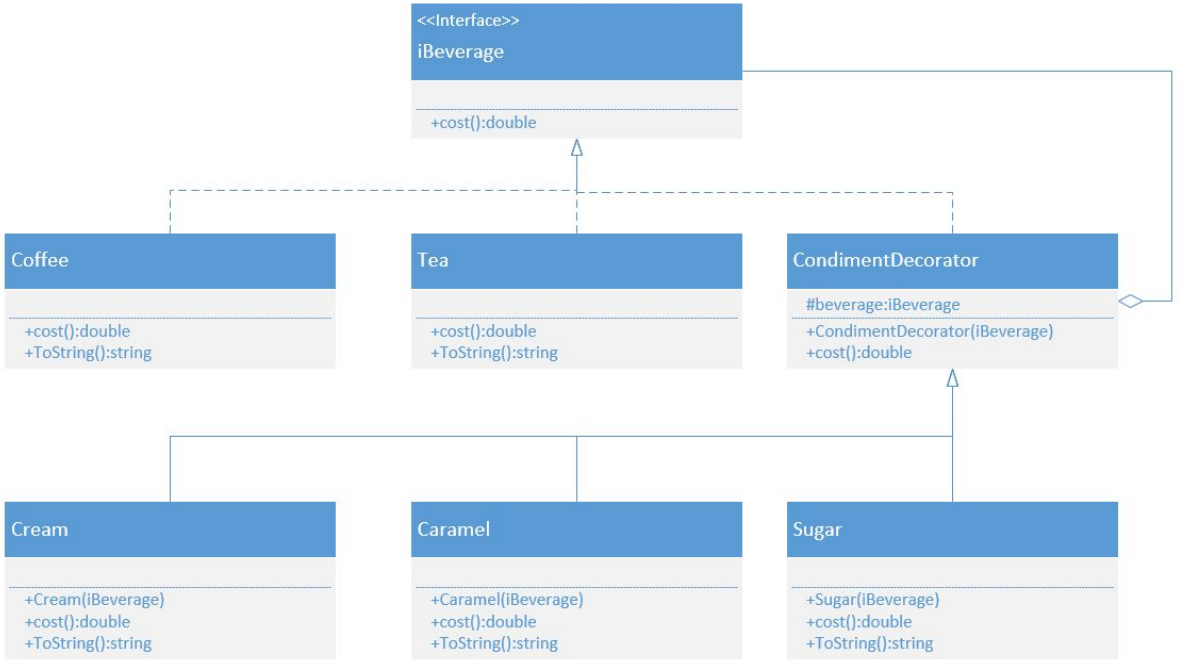
# Decorator pattern uitleg

Het decorator pattern is een patroon dat veel gebruikt wordt in bijvoorbeeld cafés of autodealers applicaties. Het kan worden gebruikt in situaties waar een duidelijk scheiding is tussen een basis bijvoorbeeld een koffie of Volkwagen Golf en extra's zoals koffiemelk of een airco. Als uitleg voorbeeld nemen we een koffietent.

Stel dat we klanten de optie willen geven voor koffie zwart, koffie met melk, koffie met suiker en koffie met melk en suiker. Als hier geen gebruik wordt gemaakt van het decorator pattern zou dat betekenen dat er iets van een drank interface is en daarnaast een implementatie voor elke optie. Als we daarnaast niet alleen koffie maar ook thee met dezelfde optie willen verkopen explodeert het klassediagram exponentieel.

Om dit tegen te gaan maakt dit patroon gebruik van een abstracte decorator klasse. Deze klasse implementeert hetzelfde interface als de base klasse en krijgt hier ook een instantie van mee. Dit betekend dat als we met een basis (koffie of thee) beginnen er vervolgens oneindig veel toevoegingen op kunnen doen.

Als we dit verwerken in een klassediagram krijgen we hetvolgende:




Het GUI maakt gebruikt van het iBeverage interface. Aan de hand van een combobox selecteren wij een basis beverage. Daarna kan aan de hand van knoppen extra's worden toegevoegd.

# Voor en nadelen van het Observer pattern

Reusability

- + De decorator klassen kunnen voor elke concrete klasse worden gebruikt

Maintainability

- + Het klassediagram blijft overzichtelijk als er veel beverages en decorators zijn.

Extensibility

- + Er kan onbeperkt nieuwe decorators en beverages toegevoegd worden
- + Er kunnen nieuwe concrete klassen worden gemaakt die alle decorators kan gebruiken
- - Als een kleine functionaliteit toegevoegd wordt moet er direct een nieuwe klasse worden gemaakt

# Feedback

## Onze feedback aan Marco en Jip

- Je kunt beter in de form_load een override tostring gebruiken voor het toevoegen van de options. Aangezien je de uitleg dan binnen de klasse houdt.
- Tuning options is overbodig
- Schoonheidsfoutje: de label wordt niet correct geupdated.
- Klassediagram ziet er goed uit.

## Hun feedback aan ons

- De condiment heeft nu geen responsibility. Doordat de condiment decorator nog geen responsibility heeft moet het een interface zijn. Maar dit is niet volgens de pattern. De decorater base zou de responsibility moeten hebben om de iBeverage bij te houden.
    - Aan de hand van dit commentaar hebben ons klassediagram gewijzigd. De CondimentDecorator is een abstracte klasse met constructor. De onderliggende klasses nemen deze constructor over.
- De user interface geeft niet duidelijk weer dat het gaat over een Decorator pattern. Het zou een factory pattern kunnen zijn.
    - Wij erkennen dat dit uit het GUI niet direct duidelijk is. Wij vinden echter dat dit niet van belang is voor de gebruiker.
- In de code heet een variabel “theBeverage”. De laatst toegevoegde condiment wordt als waarde gezet naar deze variabel. Een condiment is geen beverage/product. Dus deze naam is misleidend.

- Hier zijn we het niet helemaal mee eens, een condiment op zich is inderdaad geen beverage maar wel een toevoeging op de eerste beverage. Daarnaast kan een condiment nooit als op zichzelf staand object worden gebruikt.

# Demo

https://youtu.be/L3nqkNa4Pow