# Software Engineering

## Assignment 2: Design By Contract

*3 Ba INF 2018-2019*

Evelien Daems
Lars Van Roy

November 18, 2018

In het algemeen vereisen we bij elke functionaliteiten dat er geen operatie kan worden verricht op null objecten en dat de teruggegeven waarden eveneens geen null object kunnen bevatten.
Class Cart
*@requires({"quantity>=0", "item!=null"})*
*@ensures("$old($this.contents().get(item))+quantity==$this.contents().get(item)")*
*public void addItem(Item item, int quantity){...}*
Bij het toevoegen van een item aan een Cart object is het belangrijk dat het item een bestaand en genstantieerd object is. Want voor de gebruiker heeft het geen zin dat er een ijdel object wordt toegevoegd aan het winkelwagentje. Dit zou eveneens problemen kunnen geven bij verdere bewerkingen op Cart en dus onderweg voor errors kunnen zorgen. De vereiste dat een posititef aantal moet worden toegevoegd, is natuurlijk triviaal want het is onrealistisch dat een gebruiker een negatief aantal items in zijn of haar winkelwagen wil.
Na het toevoegen van een item moet er gegarandeerd worden dat er niets is mis gegaan tijdens het toevoegen. We zullen dus eisen dat de oude winkelwagen is geupdate met het gevraagde item en bijhorende hoeveelheid.

*@requires({"$old($this.contents().containsKey(item)==true", "item!=null", "quantity>=0"})*
*@ensures("$this.contents().get(item)==$old($this.contents().get(item))-quantity")*
*public void removeItem(Item item, int quantity) {...}*
Voor een item kan verwijderd worden is het belangrijk te controleren of dit ook effectief aanwezig is in de content van het Cart object. Quantity moet een positief getal zijn om een correct verloop van de removeItem() functionaliteit te garanderen. Indien een gebruiker toevallig een negatief getal zou toekennen aan de parameter dan voldoet het resultaat van de functie niet meer aan de verwachtingen die men heeft bij het uitvoeren. Dan zou het verwijderen resulteren in het toevoegen van items. Idem aan de functie addItem() eisen we de garantie dat een item effecief verminderd is in hoeveelheid met aantal quantity.

*@requires({"$this.contents()!=null"})*
*@ensures({"$result>=0", "$result!=null"})*
*private float getCostFloat(){...}*
Deze functie hebben we zelf toegevoegd om de totale waarde van de winkelwagen in float notatie te berekenen. De contents van het object moet genitialiseerd zijn om de kost te kunnen berekenen en het resultaat kan niet negatief zijn. Indien dit wel het geval is zou de webshop de klant gaan betalen om producten te kopen, wat helemaal niet de bedoeling is. Het resultaat moet ook bestaan indien alles goed is verlopen.

*@ensures({"$result != null, $result.length()!=0"})*
*public String getCost() {...}*
Dit is de functionalieit die de totale prijs van de winkelwagen zal teruggeven in tekst formaat. Als gebruiker verwachten we een geldige return waarde die nooit leeg kan zijn als alles goed is verlopen.

Class Catalog
*@requires({"item!=null", "$this.findMatch(item) == null"})*
*@ensures("$old($this.getNumberOfItems())+1==$this.getNumberOfItems()")*
*public void addItem(Item item) {...}*
Voor het toevoegen van een item aan de catalogus wordt er door de klant evreist dat alle items uniek moeten zijn op basis van hun naam, beschrijving en prijs. Dit moet eerst worden gecontroleerd voordat we het item gaan toevoegen aan de catalogus. Nadien moet men verifiëren of het item met succes is toegevoegd aan de catalogus. Anders zou de gebruiker er vanuitgaan dat dit item is toegevoegd en dit item gaan oproepen in een ander deel van het systeem om dan pas te ontdekken dat dit item nooit toegevoegd is geweest.

*@requires({"item!=null"})*
*public Item findMatch(Item item){...}*
Een eigen geschreven functionaliteit om aan de specificatie te voldoen waarin beschreven staat dat een Item uniek moet zijn in de zin van een unieke naam, beschrijving en prijs in de catalogus. Hier is het wel toegestaan om een return waarde te hebben van null aangezien dit betekent dat geen enkel item werd gevonden dat overeenkwam met de input.

Class Category
*@ensures("$this.getCategories().length()==10")*
*public Category(){...}*
De Catalog moet verplicht standaard 10 categoriën bevatten die staan opgesomd in de specificaties van de klant. Dus zorgen voor een postconditie die kijkt of bij het aanmaken van de Catalog de lijst ook effectief bestaat uit deze categoriën. Het is niet nodig om per categorie te gaan kijken of er wel de juiste in staan want dit zit hardcoded in het programma. Het is enkel nodig om te kijken of de lijst niet leeg is en wat er dus op zou kunnen duiden dat er iets is misgegaan bij het aanmaken van het object.

*@requires({"category.length()!=0", "!$this.getCategories().contains(category)"})*
*@ensures({"$this.getCategories().contains(category)==true", "$old($this.getCategories().size())+1==$thi*
*public void addCategory(String category){...}*
Voor het toevoegen van een categorie toe te laten, willen we eerst kijken of categorie geen lege string is, want het is onmogelijk om een lege categorie te hebben. Ten tweede moeten alle categoriën uniek zijn en controleren we of de in te voegen string als geen deel uitmaakt van de categoriën.

Class Item
*@requires({"price ¿ 0"})*
*public Item(String name, String desc, float price) {...}*
De prijs van een item in de catalogus moet een niet-nul positief getal zijn volgens de specificaties.