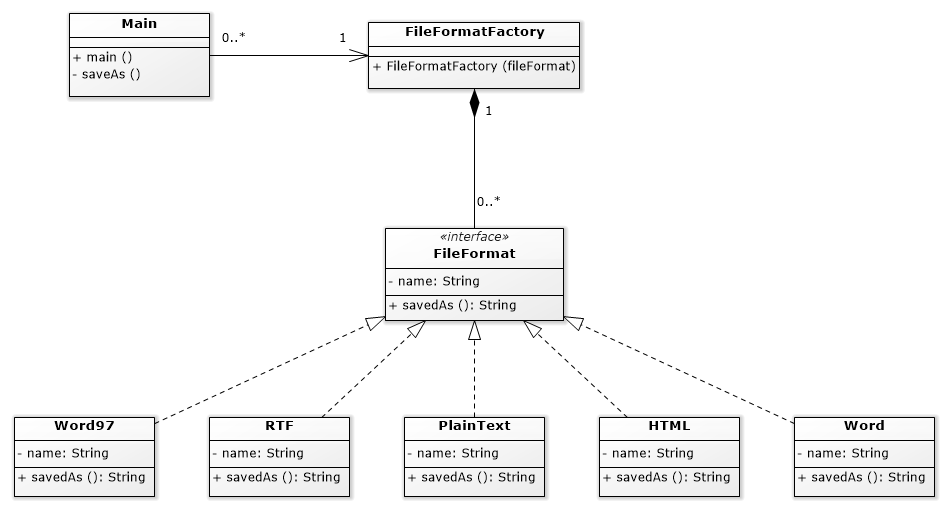
Assignment 1

Eric Boot & Maarten de Klerk

# Scenario 1

Strategy pattern

Voor deze opdracht moet er makkelijk een nieuw fileformat toegevoegd worden aan een systeem zonder al te veel code te schrijven. De coupling moet dus zo laag mogelijk blijven. Het Strategy pattern is hier ideaal voor. We hebben het volgende UML class diagram opgesteld:



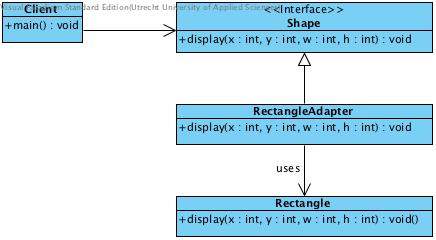
In het UML class diagram wordt het duidelijk dat je alleen een klasse hoeft toe te voegen die het FileFormat interface implementeert. Verder moet je 1 regel toevoegen aan de FileFormatFactory om daadwerkelijk een FileFormat te maken.

Voor de code zie \src\main\java\Assignment\_1\scenario\_1

# Scenario 2

Adapter pattern

Bij deze opdracht moet er een methode display() worden aangeroepen met parameters. Hiervoor is het handigste om het Adapter Pattern te gebruiken, omdat de adapter klasse op verschillende manieren moet werken.



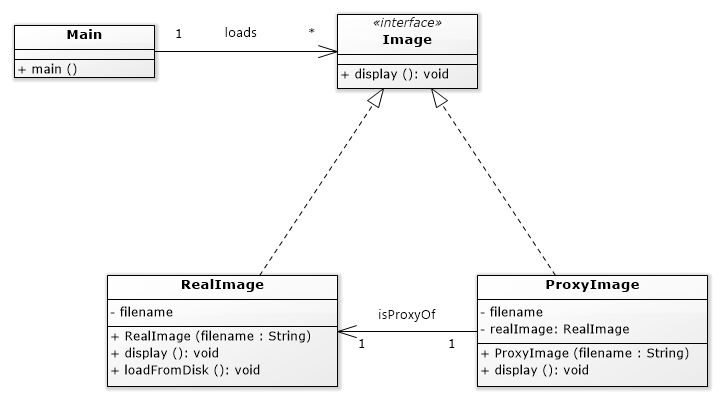
Voor het toevoegen van een nieuw object, moet er een nieuwe Adapter klasse worden gemaakt die de Shape klasse implementeert en de gewone object klasse gebruikt.

Voor de code zie \src\main\java\Assignment\_1\scenario\_2

# Scenario 3

Proxy pattern

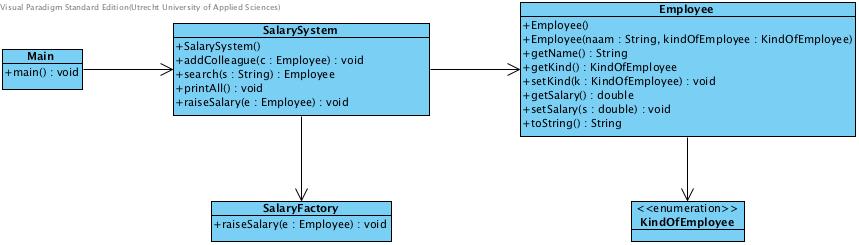
Een bedrijf heeft ons gevraagd om het laden van hun plaatjes te optimaliseren. Een goede manier is om eerst een “nep” plaatje te laden voor dat je het “echte” plaatje laad. Dit kan makkelijk verwezenlijkt worden met behulp van het Proxy pattern. Zie ons UML class diagram hier onder:



In dit pattern wordt eerst een proxy image aangemaakt en het echte plaatje word geladen. Wanneer het echte plaatje geladen is wordt het proxy plaatje niet meer geladen.

Voor de code zie \src\main\java\Assignment\_1\scenario\_3

# Scenario 4



Bij deze verbeterde code is het nu overzichterlijker om te runnen; kunnen er minder fouten worden gemaakt bij het kiezen van KindOfEmployee en zijn de voorwaarden en werkwijze op het verhogen van het salaris beter te beheren.