

Analyse

Je schrijft eerst je Use Case om daar vervolgens de analyse uit te halen. De analyse is eigenlijk de kern van je nog te schrijven programma.

Objecten:

Je objecten kan je, nadat je je Use Case hebt geschreven, gemakkelijk uitfilteren. Zo vind je bijvoorbeeld in een kaartspel, de deler, de speler, de kaarten etc.

Daarna moet je uit je Use Case de werkwoorden zoeken. Deze heb je nodig om te kijken wat elk object moet gaan doen. Voorbeeld, de deler verdeelt de kaarten, de deler berekend de waarde, de deler start nieuw spel etc.

Als dit alles gedaan is wordt het tijd om een CRC te schrijven (Class Responsibility Collaboration)

Responsibilities

Collaborations

Kaart:

- Noem naam
- Geef punten waard

Deck:

- Aantal kaarten
- Kaarten mengen
- Geef volgend kaart

Dealer:

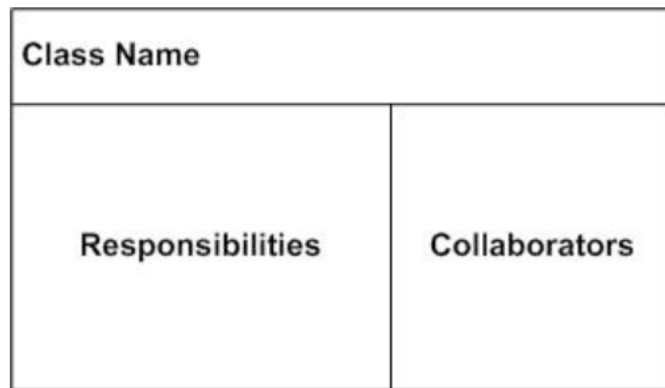
- Start nieuw spel
 - Verdeel nieuwe kaart
- Hand
Speler, Deck

Speler:

- Meer kaarten?
 - Vraag om kaart
 - Laat hand zien
 - Waarde van hand
- Hand
Dealer
Hand

Hand

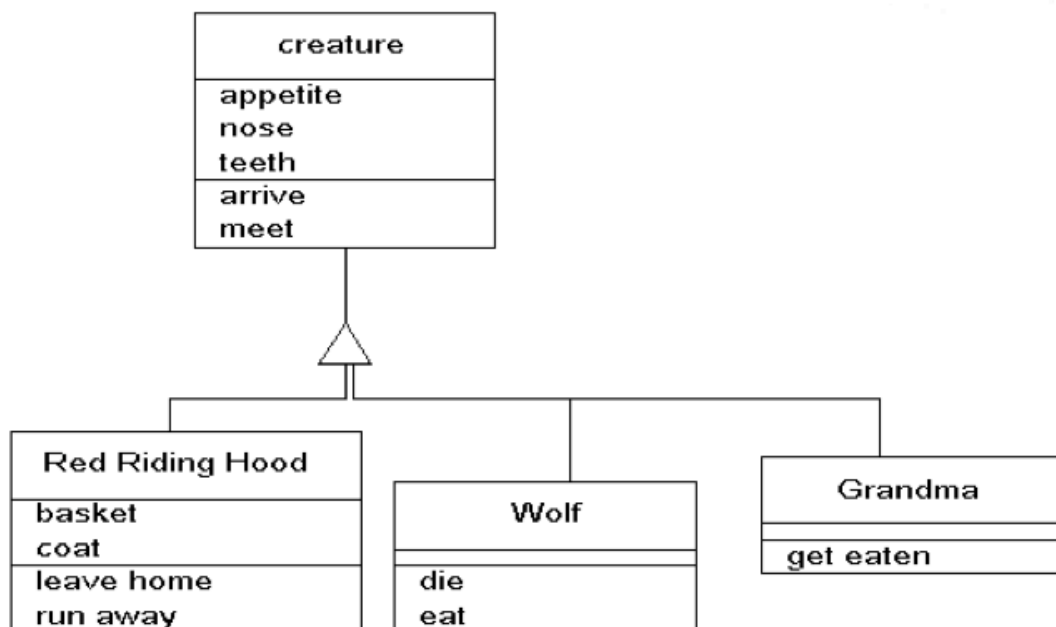
- Geef totaal punten
 - Voeg kaart toe
 - Laat hand zien
- Speler
Dealer



Inheritance

Als je een super class gebruikt om de meest voorkomende zaken door te geven aan onderliggende classes.

Voorbeeld:



Association

Dependency:

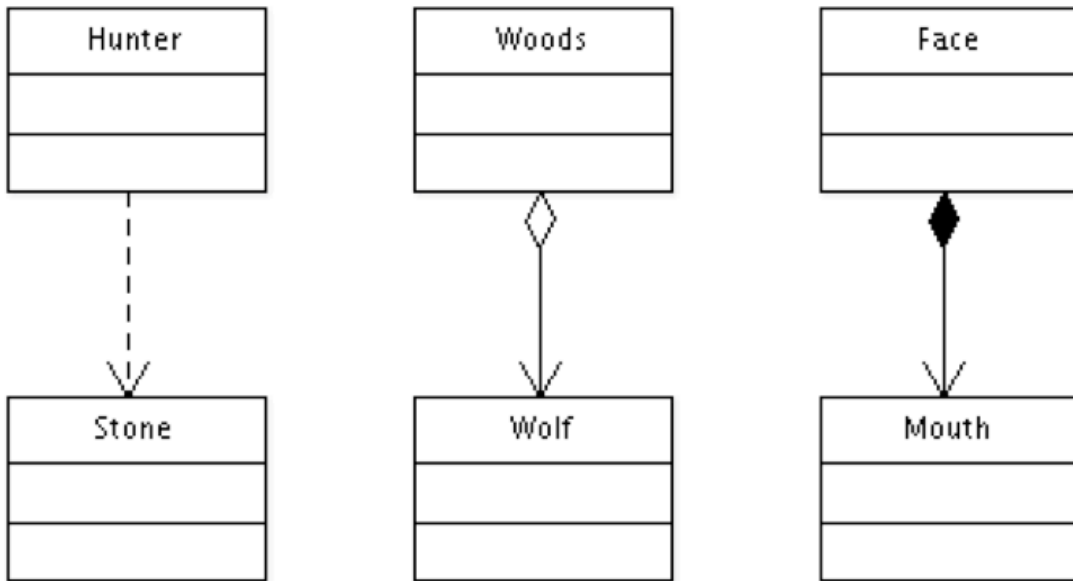
Een class moet soms vertrouwen op een andere class. Dit omdat hij anders zijn opdracht niet uit kan voeren.

Aggregation:

Een symbolische relatie tussen twee classes/objecten. De ene class kopieert (private copy) een functionaliteit van de andere class.

Composition:

De aparte objecten moeten samenwerken om iets te bereiken. Als voorbeeld, wielen, een stuur en een motor zijn los van elkaar vrij waardeloos, maar door deze aan elkaar te linken (composition) wordt het toch iets wat kan rijden.



Dependency

Aggregation

Composition

Access Modifiers

Public:

Dit houdt in dat de functie of variabelen aangeroepen kan worden van buiten de klasse.

Private

Dit houdt in dat de functie of variabelen allen beschikbaar is voor de eigen klasse en "child" klasse.

Protected

Dit houdt in dat de variabelen of functie in de klasse kan worden aangeroepen en dus niet overgeërfd wordt naar een "child" klasse.

Internal

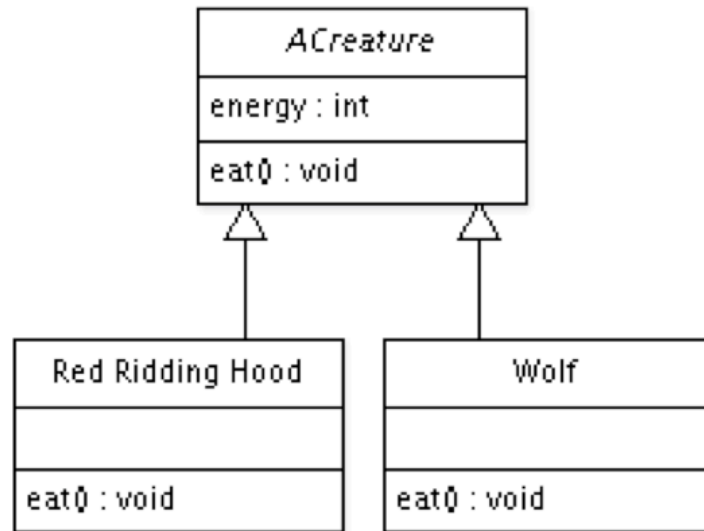
Geeft alleen interne toegang



Public, Private, *Protected*, *Internal*

Abstract Class:

De class hoogst aangegeven, dit zou bijvoorbeeld de interface kunnen zijn.



Polymorphism:

Zo kunnen objecten van verschillende types reageren op hun eigen manier. Ondanks dat het aanroepen hetzelfde is zou de output verschillend kunnen zijn.

