Principes van objectgeoriënteerd werken

Objectgeoriënteerde analyse en ontwerp

Wat is een object?

Wat maakt een object een object?

- identiteit
- data / toestand / state
- methodes / gedrag / behavior

Een object is een samenhangend geheel van data en methodes om met die data te werken.

Een object is een instantie van een klasse.

OO: basisprincipes

Wat zijn de basisideeën van object-oriëntering?

- modulariteit
- encapsulering
- overerving
- polymorfisme

Waarom objectgeoriënteerd werken?

Om onderhoudsvriendelijker code te krijgen.

OO: basisprincipes

Modulariteit

Objecten zijn logische eenheden.

Objecten zijn uitwisselbaar.

OO: basisprincipes

Encapsulering (data hiding)

Waar je beter afblijft zit verstopt onder de motorkap.

Een object verbergt zijn implementatie voor ‘onbevoegden’.

OO: basisprincipes

Overerving

De functionaliteit van een object uitbreiden of specialiseren.

OO: basisprincipes

Polymorfisme

Meerdere objecten kunnen een bepaalde taak uitvoeren.

Welk object een taak uitvoert wordt pas op het moment zelf bepaald.

OO: beperkingen

Beperkingen van 'klassieke' objecten

De code van het object moet vastliggen at compile time.

At runtime is het dus niet mogelijk om attributen toe te voegen aan een object (toestand uitbreiden), of om de implementatie van methodes te veranderen (gedrag veranderen).

Functies zijn geen objecten.

SOLID OO

Wanneer ben je goed bezig met OO?

Goede OO-code is SOLID.

Single purpose
Open/closed
Liskov substitution
Interface segregation
Dependency inversion

SOLID OO

Single purpose principle

Elke klasse heeft 1 duidelijke verantwoordelijkheid.

Elke klasse heeft 1 reden om gewijzigd te worden.

Aanwijzingen dat dit principe toegepast wordt:

- klassen zijn beperkt qua omvang
- naam van de klasse omschrijft het doel
- klassen zijn gemakkelijk te ordenen (mappen in project)

SOLID OO

Open/closed principle

Open for extension: een object moet uitbreiding toelaten

Closed for modification: het gedrag van een object mag niet wijzigbaar zijn

Aanwijzingen dat dit principe toegepast wordt:

- functionaliteit wordt toegevoegd met nieuwe klassen, niet door veranderen van bestaande klassen
- gebruik van overerving
- gebruik van interfaces

SOLID OO

Liskov substitution principle

Een object moet altijd vervangen kunnen worden door een meer gespecialiseerd object.

Is min of meer verzekerd als je overerving gebruikt,

én het ‘contract’ van de interface respecteert.

SOLID OO

Interface segregation principle

Een object mag niet afhankelijk zijn van methodes die het niet gebruikt (heeft dus enkel methodes die het echt nodig heeft).

Aanwijzing dat dit toegepast wordt:

- er zijn veel kleine interfaces in plaats van enkele uitgebreide interfaces

SOLID OO

Dependency inversion principle

High-level modules mogen niet van low-level modules afhankelijk zijn, maar moeten afhankelijk zijn van abstracties.

Abstracties mogen niet van details afhankelijk zijn. Details moeten afhankelijk zijn van abstracties.