Object-oriented programming vs Functional programming

Een van de grootste voordelen dat object geörienteerd programmeren ons tijdens deze taak heeft gegeven is het kunnen werken met states binnen objecten. Het is enorm gemakkelijk om te kunnen weten waar de instantie aan toe is en zo het verdere verloop te kunnen bepalen. Het is ook heel gemakkelijk om te kunnen controleren welk gebruik is toegestaan en verboden voor extern gebruik. Zo zou het mogelijk kunnen zijn om bij sommige van onze strategieën in ons voorbeeld bijvoorbeeld specifieke logica te kunnen verbergen voor extern gebruik.

Het was ook heel erg gemakkelijk om met overerving te kunnen werken voor onze strategy’s om zo te kunnen specifiëren welk type er terug gestuurd wordt uit bepaalde functies. Zo blijft alle code netjes en getypeerd zonder onnodige verassingen. Dit kan bijvoorbeeld wel het geval zijn bij talen die niet erg OO gefocust zijn zoals javascript.

Het werken met klassen zorgt er ook voor dat er heel modulair gewerkt kan worden, men kan heel makkelijk code toevoegen en verwijderen in verschillende bestanden (= classes). Zo kan er bijvoorbeeld in ons voorbeeld perfect nog in aparte files andere strategy’s bij onze calculator toegevoegd worden zoals vierkantswortels en dergelijke.

Deze modulariteit zorgt ook voor heel veel herbruikbaarheid. We kunnen de code op meerdere plaatsen herbruiken en hoeven niet voor elke use-case de code te repliceren, wat dan ook weer de foutenmarge naar beneden haalt.

Een nadeel van object georiënteerd te programmeren is volgens ons dat wanneer er met een database (of andere situaties waar models bijgehouden moet worden) dient gewerkt te worden er telkens vrij veel moet gebeuren qua ‘modellen’ vooraleer er relatief simpele acties kunnen uitgevoerd worden. OO brengt op die manier met momenten veel overhead wanneer dat vermeden zou kunnen worden. Op dat vlak kan functioneel programmeren zeker simpelere en snellere opties bieden.

Functioneel programmeren zorgt hoofdzakelijk voor minder fouten in verband met type checking. Het legt meer een nadruk op wat er gedaan moet worden tegenover hoe het gedaan moet worden, met een nadruk op efficiëntie en en optimisatie. In ons geval merkte we wel dat bij de javascript kant we met momenten minder problemen hadden met banale typeringsfoutjes.

Het is in het algemeen ook veel gemakkelijker om te begrijpen wat er waar en hoe gebeurt bij FP, dit is zeker ook te zien waneer er simpele clickhandlers worden toegevoegd in onze javascript applicatie bijvoorbeeld.

Al bij al kan er gesteld worden dat beide programmeerstijlen een plaats hebben in de development wereld. De één is niet beter dan de andere of omgekeerd. Beide bieden bepaalde voor- en nadelen, waarvan we er enkele in deze paper hebben vermeld.