Opdrachten kwartaal 4 week 4 – Overerving en polymorfisme

Door het gebruik van overerving en polymorfisme in C++ kun je software maken die gesloten is voor aanpassingen maar toch open is voor uitbreidingen[1].

Je leert deze les hoe je:

- met behulp van overerving een 'is een'-relatie tussen classes implementeert;
- een polymorfe functie kunt definiëren die je zowel voor objecten van een bepaalde class als voor objecten van de, van deze class, afgeleide classes kunt gebruiken;
- er voor kunt zorgen dat software eenvoudig uit te breiden is, zonder dat bestaande code gewijzigd hoeft te worden, door overerving en polymorfisme toe te passen.

Freelacers en vaste krachten

In een bedrijf werken verschillende soorten werknemers:

- freelance werknemers. Deze werknemers verdienen een vast bedrag per gewerkt uur.
- vaste krachten. Deze werknemers verdienen een vast bedrag per maand.

Binnen de salarisadministratie van het bedrijf heeft elke werknemer een registratienummer.

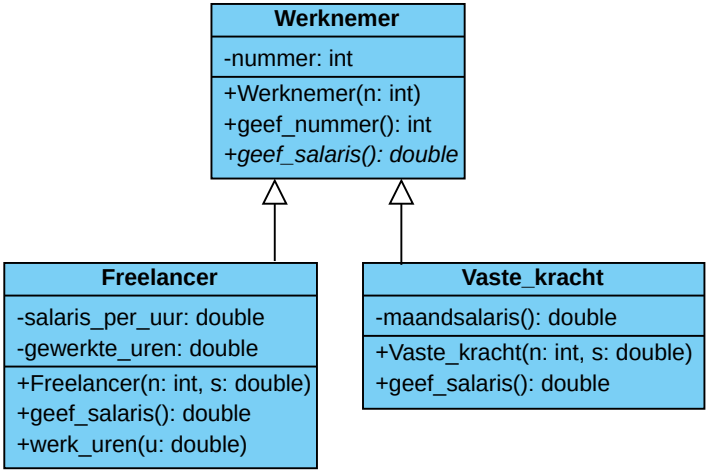
De programmeur die de specificaties voor het C++-programma voor de salarisadministratie heeft opgesteld, heeft bedacht dat er rekening mee moet worden gehouden dat er in de toekomst andere soorten werknemers moeten worden toegevoegd. Bijvoorbeeld stukwerkers die werken voor een vaste stukprijs. Het programma moet dus zoveel mogelijk onafhankelijk van het concrete soort werknemer gemaakt worden.

De programmeur heeft een class hiërarchie ontworpen die is weergegeven in het UML-klassendiagram van figuur 1. Deze classes zijn gedeclareerd en gedefinieerd in `werknemer.cppm`, `werknemer.cpp`, `freelancer.cppm`, `freelancer.cpp`, `vaste_kracht.cppm` en `vaste_kracht.cpp`. In `print_maandsalaris.cppm` en `print_maandsalaris.cpp` is een functie gedeclareerd en gedefinieerd die het maandsalaris van een werknemer print. In `run_werk1.cpp` is een programma geschreven dat gebruik maakt van de classes `Freelancer` en `Vaste_kracht` en van de functie `print_maandsalaris()`. De bijbehorende test is gegeven in `test_werk.cpp`.

Alle bestanden die nodig zijn voor de onderstaande opdrachten zijn te vinden in: `werk.zip`[2]

[1] Zie eventueel `https://nl.wikipedia.org/wiki/Open/closed-principe`.

[2] `https://github.com/HR-ELEKTRO/ems3/raw/refs/heads/master/Opdrachten/progs/werk.zip`



Figuur 1: De relaties tussen de verschillende soorten werknemers.

4.4.1 Bestudeer de hierboven genoemde `.cppm` en `.cpp` bestanden en beantwoord de volgende vragen:

A De functie `print_maandsalaris` heeft als parameter een referentie naar een object van de class `Werknemer`. Deze functie wordt echter aangeroepen met als argument het object `f` van de class `Freelancer` en ook met het object `v` van de class `Vaste_kracht`. Waarom geeft de compiler daar geen foutmelding op?

B De functie `print_maandsalaris` is dus aan te roepen met objecten van verschillende soorten werknemers. Hoe noemen we zo'n functie?

C In de functie `print_maandsalaris` wordt onder andere de memberfunctie `w.geef_nummer()` aangroepen. Welke code wordt hierdoor uitgevoerd?

D Is het mogelijk om de functie `geef_nummer` in een, van `Werknemer` afgeleide, class te overridden? Waarom wel/niet?

E In de functie `print_maandsalaris` wordt onder andere de memberfunctie `geef_salaris()` aangroepen. Welke code wordt hierdoor uitgevoerd?

F Hoe noemen de manier waarop de aanroep van de memberfunctie `geef_salaris()` 'gebonden' wordt met de uit te voeren code?

4.4.2 Als je het programma uitvoert, dan zie je dat alle salarissen 0.00 Euro zijn. Dat komt omdat de code van de overridden functies `geef_salaris()` nog niet is ingevuld. Vul

de code in om het salaris van de betreffende soort werknemer te bepalen. De juiste uitvoer is:

```
Maand 1:
Werknemer: 1 verdient:  2163.00 Euro.
Werknemer: 2 verdient:  1873.53 Euro.
Maand 2:
Werknemer: 1 verdient:   347.63 Euro.
Werknemer: 2 verdient:  1873.53 Euro.
```

Let op! In plaats van `347.63` kan het salaris van werknemer 1 in maand 2 ook bepaald zijn op `347.62`. Het juiste antwoord is $13.5 \times 25.75 = 347.625$. Je zou misschien verwachten dat deze uitkomst door het gebruik van de format specifier `:>8.2f` netjes afgerond wordt. `:>8.2f` rondt inderdaad netjes af op 2 cijfers achter de decimale punt. Maar als door een klein afrondfoutje in de berekening, het resultaat 347.624999999999999999... is, dan wordt deze waarde (netjes) naar beneden afgerond. Het gebruik van floating point getallen is altijd lastig. Als je echt alles wilt weten lees dan het artikel: *What Every Computer Scientist Should Know About Floating-Point Arithmetic* van David Goldberg[3].

In de gegeven code wordt gebruikt gemaakt van enkele C++ taalconstructies die in de les niet zijn behandeld. Om de werking van het programma te begrijpen is het niet nodig om deze details te kennen. Maar voor iemand die alles wil weten:

- De class `Werknemer` heeft een virtuele destructor. Wat een destructor is kun je lezen in paragraaf 5.3 van het dictaat. Waarom het verstandig is om de base class `Werknemer` een virtuele destructor te geven, kun je lezen in paragraaf 5.5 van het dictaat.
- De memberfunctie `geef_salaris` is in de class `Werknemer` als een *pure virtual* memberfunctie gedefinieerd. De class `Werknemer` wordt daarmee een zogenoemde *Abstract Base Class* (ABC). Wat een ABC is kun je lezen in paragraaf 4.4 van het dictaat. Waarom het verstandig is om de class `Werknemer` een ABC te maken, kun je lezen in paragraaf 18.6 van het dictaat.
- Vanuit de constructor van `Freelancer` wordt de constructor van `Werknemer` aangeroepen. Dit is nodig omdat de class `Freelancer` het private datamember `nummer` van de class `Werknemer` niet kan gebruiken. Hoe dit werkt, kun je lezen in paragraaf 4.5 van het dictaat.

[3] Zie https://dl.acm.org/doi/pdf/10.1145/103162.103163.

In de gegeven testcode wordt gebruikt gemaakt van een test macro `EXPECT_THAT` die je nog niet eerder hebt gezien. Om de testcode te gebruiken is het niet nodig om deze details te kennen. Maar voor iemand die alles wil weten:

- Om te testen of het salaris van de freelancer in week 2 correct is berekend moeten we controleren of het salaris 347.63 of 347.62 is, zoals uitgelegd op pagina 3. Dit doen we door de uitkomst van de functie `geef_salaris()` in de macroe `EXPECT_THAT` te vergelijken met een patroon door gebruik te maken van een zogenoemde sting matcher[4]. Met de string matcher `testing::MatchesRegex` kun je een reguliere expressie[5] gebruiken om te controleren of een string aan een bepaald patroon voldoet. In dit geval is het patroon `"Werknemer: 1 verdient:  347.6[23] Euro.\n"`. De `[23]` betekent dat op die plaats in de string een 2 of 3 mag staan.
- Om `EXPECT_THAT` te kunnen gebruiken moet de header file `gmock/gmock.h` in het testprogramma opgenomen worden.

Stukwerkers

Het programma is eenvoudig uit te breiden met een nieuw soort werknemer. In `stukwerker.cppm` en `stukwerker.cpp` is een nieuw soort werknemer `Stukwerker` gedeclareerd en gedefinieerd. Deze stukwerker werkt voor een vaste stukprijs. In `run_werk2.cpp` is een programma geschreven dat gebruik maakt van de classes `Freelancer`, `Vaste_kracht` en `Stukwerker` en van de functie `print_maandsalaris()`. De bijbehorende test (`test2`) is nog uitgecommentarieerd gegeven in `test_werk.cpp`.

4.4.3 Pas de `CMakeLists.txt` aan, zodat de module `Stukwerker` gecompileerd wordt en `run_werk2.cpp` gecompileerd en gelinkt wordt tot `run_werk2`. Vul de ontbrekende code in `stukwerker.cppm` en `stukwerker.cpp` in, om het programma `run_werk2.cpp` correct te laten werken. De juiste uitvoer is:

```
Maand 1:
Werknemer: 1 verdient:  2163.00 Euro.
Werknemer: 2 verdient:  1873.53 Euro.
Werknemer: 3 verdient:  1771.35 Euro.
Maand 2:
```

[4] Zie eventueel: `http://google.github.io/googletest/reference/matchers.html#string-matchers`.

[5] Met reguliere expressies kun je eenvoudige maar ook complexe patronen beschrijven, zie eventueel: `https://en.wikipedia.org/wiki/Regular_expression`.

```
Werknemer: 1 verdient:   347.63 Euro.
Werknemer: 2 verdient:  1873.53 Euro.
Werknemer: 3 verdient:     0.00 Euro.
```

Dit wordt gecontroleerd in de test `test2` in `test_werk.cpp`. Pas de `CMakeLists.txt` en de code in `test_werk.cpp` aan om deze test te kunnen uitvoeren.

Merk op dat dit programma uitgebreid kan worden zonder de code van de classes `Werknemer`, `Freelancer` en `Vaste_kracht` en van de functie `print_salaris()` aan te passen. Omdat deze code in aparte `.cpp` bestanden is geplaatst, hoeft deze code ook niet gecompileerd te worden. Dit wil zeggen dat het niet nodig is om de unittests van genoemde classes en functies opnieuw uit te voeren. De code is immers niet gewijzigd. Je hoeft dus ook niet bang te zijn dat deze uitbreiding bestaande functionaliteit heeft aangepast.

Managers

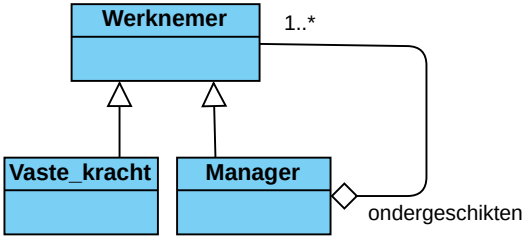
Er is nog een soort werknemer waarmee je het programma kunt uitbreiden. Een manager is een werknemer die andere werknemers leiding geeft. De meeste managers hebben zelf ook weer een manager boven zich. Het bedrijf dat we als voorbeeld nemen heeft een originele manier bedacht om het salaris van een manager te bepalen. Het salaris van de manager is twee maal zo hoog als het gemiddelde salaris van de mensen waaraan deze manager direct leiding geeft.

Het programma moet natuurlijk ook het salaris van managers kunnen berekenen. De relatie tussen manager en werknemer is als volgt:

- een manager *is een* werknemer;
- een manager *heeft een (of meer)* werknemer(s) (onder zich).

De relaties tussen de classes `Werknemer`, `Vaste_kracht` en `Manager` zijn gegeven in figuur 2. In `manager.cppm` en `manager.cpp` is een nieuw soort werknemer `Manager` gedeclareerd en gedefinieerd. In `run_werk3.cpp` is een programma geschreven dat gebruik maakt van de classes `Freelancer`, `Vaste_kracht`, `Stukwerker` en `Manager` en van de functie `print_maandsalaris()`. De bijbehorende test (`test3`) is nog uitgecommentarieerd gegeven in `test_werk.cpp`.

4.4.4 Pas de `CMakeLists.txt` aan, zodat de module `manager` gecompileerd wordt en `run_werk3.cpp` gecompileerd en gelinkt wordt tot `run_werk3`. Vul de ontbrekende code in



Figuur 2: De relaties tussen de classes `Werknemer`, `Vaste_kracht` en `Manager`.

`manager.cppm` en `manager.cpp` in, om het programma `run_werk3.cpp` correct te laten werken. De juiste uitvoer is:

```
Werknemer: 1 verdient:   711.90 Euro.
Werknemer: 2 verdient:  2163.00 Euro.
Werknemer: 3 verdient:  1873.53 Euro.
Werknemer: 4 verdient:  3165.62 Euro.
Werknemer: 5 verdient:  2036.18 Euro.
Werknemer: 6 verdient:  5201.80 Euro.
```

Dit wordt gecontroleerd in de test `test3` in `test_werk.cpp`. Pas de `CMakeLists.txt` en de code in `test_werk.cpp` aan om deze test te kunnen uitvoeren.