Opdrachten week 6 les 2 – 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.

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.

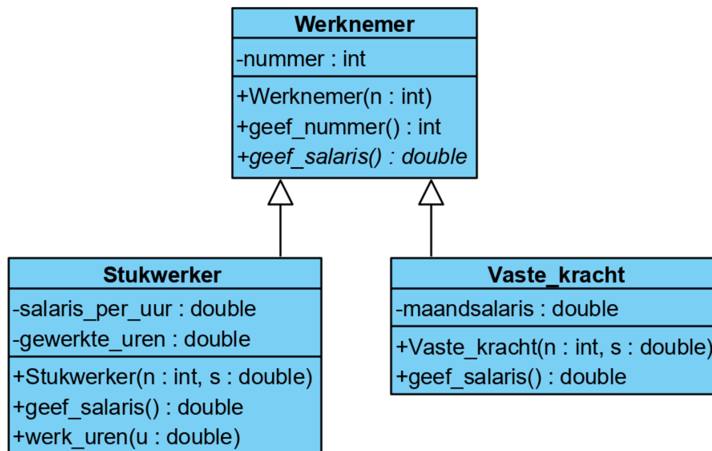
De programmeur heeft een class hiërarchie ontworpen die is weergegeven in het UML-klassendiagram van figuur 1. Het programma dat deze classes implementeert en gebruikt is gegeven in `werk1.cpp`.

6.2.1 Bestudeer het programma `werk1.cpp` 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?

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



Figuur 1: De relaties tussen de verschillende soorten werknemers.

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?

6.2.2 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 `setprecision(2)` netjes afgerond wordt. `setprecision(2)` rondt inderdaad netjes af op 2 cijfers achter de decimale punt. Maar als door een klein afrondfoutje in de berekening, het resultaat $347.624999999999999\ldots$ 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[2].

In het gegeven programma 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 5.4 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.

Het programma is nu eenvoudig uit te breiden met een nieuw soort werknemer. In `werk2.cpp` is het programma uitgebreid met een nieuw soort werknemer: stukwerker. Deze stukwerker werkt voor een vaste stukprijs.

6.2.3 Vul de ontbrekende code in, om het programma `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.
```

[2] Zie `https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.472.4807&rep=rep1&type=pdf`.

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

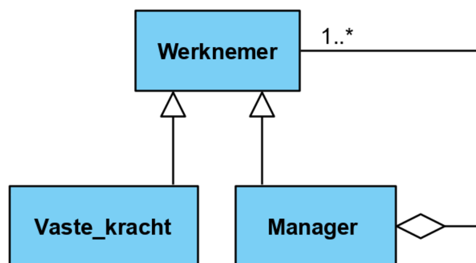
Merk op dat dit programma uitgebreid kan worden zonder de code van de classes `Werknemer`, `Freelancer` en `Vaste_kracht` en van de functie `print_salaris()` aan te passen. Als deze code in aparte .cpp bestanden was geplaatst, dan zou deze code ook niet gecompileerd hoeven te worden. Dit wil zeggen dat 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.

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 `werk3.cpp` is het programma uitgebreid met een de class `Manager`.



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

6.2.4 Vul de ontbrekende code in, om het programma `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.
```