COMPOSITIE ONDERZOEK

Inhoudsopgave

Wat is object compositie en aggregratie?

- Klassen opgebouwd uit objecten van andere klassen

- ‘Has a’ relatie

- Gebruikt in grote projecten

- Is wanneer een klasse is samengesteld uit objecten van andere klassen, maar de hoofdklasse is niet de eigenaar van de member variables

- Member variable onafhankelijk van hoofdklasse

- Pointers/ referenties naar member variable

Voor en nadelen

- Compositie is flexibeler dan overerving
- Minder complexiteit
- Een losser verband
- Een eenvoudigere en betere keuze dan overerving

- Verkeerd worden gebruikt
- verschillende levenscyclus en relaties kan uitdagend zijn

Voor en nadelen aggregatie

- Herbruikbare code
- Flexibiliteit
- Modelleren van complexere systemen

- Complexiteit verhogen van structuren van het systeem
- Bouwt geen hiërarchisch verband
- Onnodige koppelingen tussen klassen

Implementatievoorbeelden

C++

```cpp
class Car {
private:
    Engine engine;
    Wheel wheels[4];  // Composition with multiple instances of a component
public:
    // Constructor initializes Engine and Wheel objects
    Car(int hp, int ws) : engine(hp), wheels{Wheel(ws), Wheel(ws), Wheel(ws), Wheel(ws)} {}

    void displaySpecifications() const {
        std::cout << "Engine: " << engine.getHorsepower() << " HP\n";
        std::cout << "Wheel size: " << wheels[0].getSize() << " inches\n";
    }
};

int main() {
    Car myCar(300, 18);
    myCar.displaySpecifications();  // Outputs: Engine: 300 HP \n Wheel size: 18 inches
    return 0;
}
```

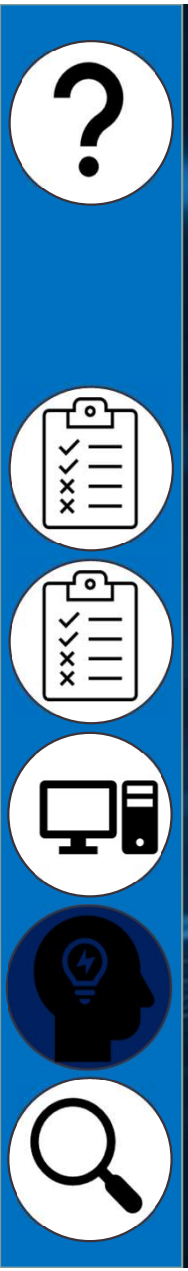
C++

```cpp
#include <iostream>

class Engine {
private:
    int horsepower;   // Engine has a horsepower attribute
public:
    Engine(int hp) : horsepower(hp) {}  // Constructor initializes horsepower
    int getHorsepower() const {
        return horsepower;
    }
};

class Wheel {
private:
    int size;  // Wheel has a size attribute
public:
    Wheel(int s) : size(s) {}  // Constructor initializes size
    int getSize() const {
        return size;
    }
};
```

```cpp
#include <iostream>
#include<string.h>
using namespace std;
class Tech{
public:
int house;
string city, state;
Tech(int house_no, string city, string state)
{ this->house = house_no;
this->city = city;
this->state = state;
}
};
class Person
{
private:
Tech* address;
public:
string name;
Person(string name, Tech* address)
{
this->name = name;
this->address = address;
}
void display()
{
cout<< name<< " "<< " "<< address->house<<" "<<address->city<< " "<<address->state<<endl;
}
};
int main(void) {
Tech add1= Tech(112 ,"Bandra","Mumbai");
Tech add2 = Tech(222, "Jahanpanah","New Delhi");
Person p1 = Person("Raj",&add1);
Person p2 = Person("John",&add2);
cout << "Name of the Person" << " and " << "Address"<< endl<<endl;
p1.display();
p2.display();
return 0;
}
```



Waar gaat het vaak fout?

- Vaak pointers en referenties
- Object verwijderd
- Pointers verwijzen nergens meer naar
  Geen eigenaarschap
- Meerdere keren verwijderd of aangepast
- Kans dat object niet opgeruimd wordt
- Memory leak als gevolg

Volgorde is belangrijkAls de volgorde verkeerd in initializer wordt gezet wordt het overgeschreven.,

,

Errors en onverwacht gedrag als gevolg,

Als het originele object wordt verwijderd worden pointers en referenties ook verwijderd Dit zorgt voor error, crashes of memory leaks

Bronnen

Covers Composition and Agregation - Composition - C++ Tutorial (Part 36)

Compositon vs inheritance - link

Composition | Evolved - link

Composition voorbeeld - Composition code

Aggregatie voorbeeld : Aggregratie code

Aggregatie definitie/voordelen: Aggregatie definitie

Aggregatie vs Inheritance (voordelen/nadelen): Aggregatie vs Inheritance

Waarschuwingen over het gebruik van aggregatie: link

Foutmelding met composite: link

Shallow copy (compositie): link

(What is aggregation and compensation)link

Stack difference between aggregation and composition (link)

Einde

Schrijvers: Emma, Amira, Erin, Jente & Maud

Onderzoekers: Vigo, Luuk, Alea, Jorn, Nathaniël, Laurens & Ying

Github: Tess, Sarah, Ryan, Joni & Ruben

Presentatoren: Emma, Jente, Maud & Luuk