Inheritance

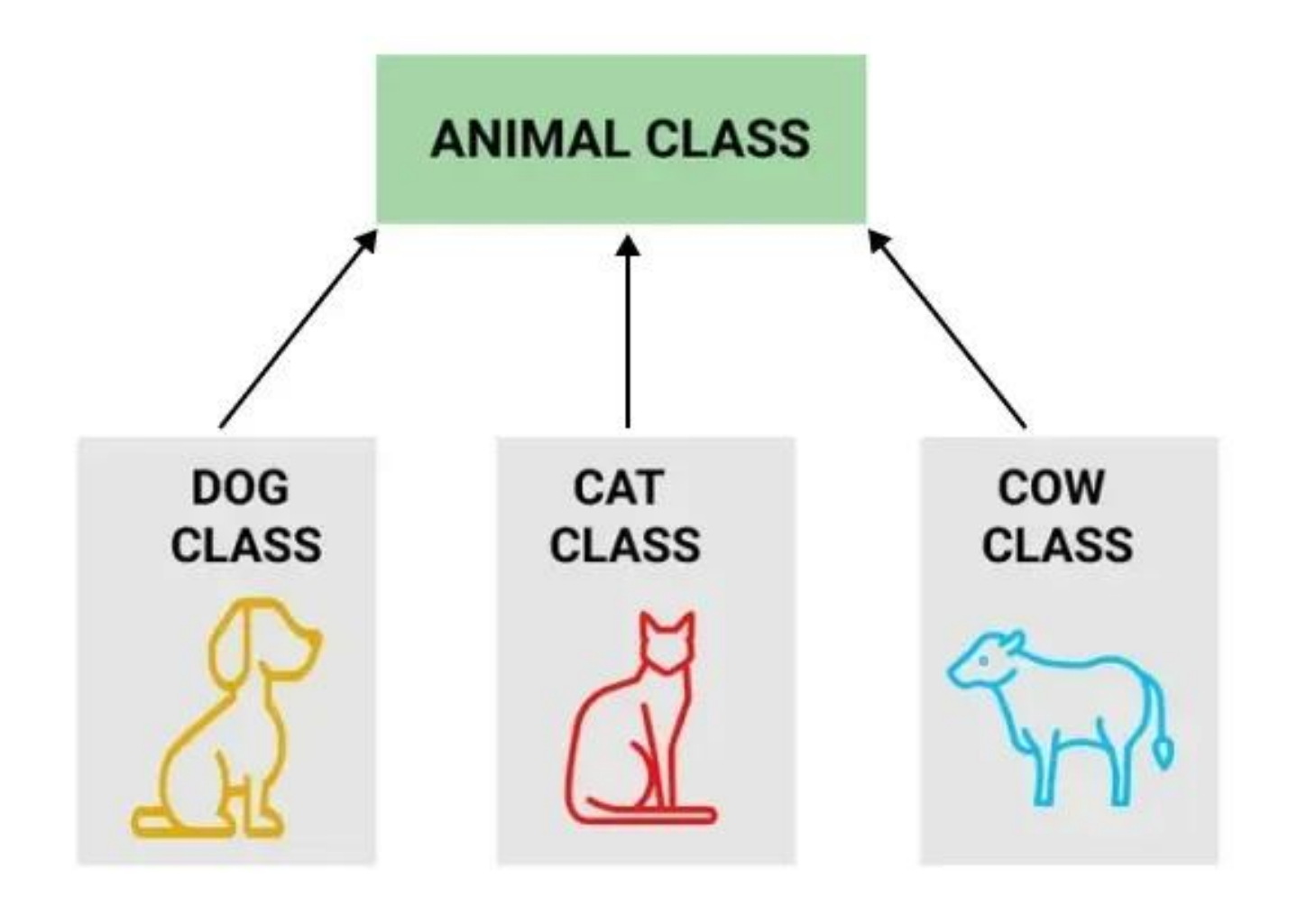
Samenvatting

- Wat betekent het in C++?
- Hoe gebruik je het in echte projecten?
- Wat zijn de voordelen en nadelen?
- Wat zijn goede voorbeelden?
- Waar gaat het vaak fout?

Wat betekent het in C++?

Inheritance is een manier om herhaling te voorkomen door attributen op een hoger niveau door te geven aan lagere klassen.

De hogere klassen noemen we "Parent classes" en de lagere klassen noemen we "Child classes"



Voorbeeld

Voor dit voorbeeld maken we een class voertuig binnen deze voertuig class heeft elke voertuig een merk dan hebben we een child class genaamd auto onder voertuig heeft elke auto een soort.

```cpp
#include <string>
using namespace std;


class Voertuig {
    private:
        string merk;

    public:
        Voertuig(const string& merk) : merk(merk) {}
};

class Auto : public Voertuig {
    private:
        string naam;

    public:
        Auto(const string& merk, const string& naam) : Voertuig(merk), naam(naam) {}
};

int Main() {
    Auto auto1("Lamborghini", "Aventador");
    Auto auto2("Bughatti", "Chiron");
}
```

Hoe gebruik je het in echte projecten?

In de praktijk gebruik je inheritance vooral om:

- **Codehergebruik**
- **Structuur en logica**
- **Polymorfisme**

```cpp
class Animal {
public:
    virtual void makeSound() {
        std::cout << "Some generic animal sound" << std::endl;
    }
};
```

```cpp
class Employee {
public:
    void clockIn() { std::cout << "Clocked in.\n"; }
    virtual void work() { std::cout << "Working...\n"; }
};

class Developer : public Employee {
public:
    void work() override { std::cout << "Writing code.\n"; }
};

class Manager : public Employee {
public:
    void work() override { std::cout << "Managing the team.\n"; }
};
```

```cpp
class Dog : public Animal {
public:
    void makeSound() override {
        std::cout << "Woof!" << std::endl;
    }
};

class Cat : public Animal {
public:
    void makeSound() override {
        std::cout << "Meow!" << std::endl;
    }
};
```

Wat zijn de voor- en nadelen?

Voordelen	Nadelen
Hergebruik van code	Fragiele basisklasse
Voorkomt duplicatie van code uit een andere klasse	Diamant-probleem
Eenvoudig uitbreidbaar	





Wat zijn goede voorbeelden?

- Dieren
- Parentklasse
- Overerving

```cpp
#include <iostream>
using namespace std;

// Basisklasse (ouderklasse)
class Dier {
public:
    void eten() {
        cout << "Het dier eet voedsel." << end
    }

    void slapen() {
        cout << "Het dier slaapt." << endl;
    }
};
```

```cpp
class Hond : public Dier {
public:
    void blaffen() {
        cout << "De hond blaft!" << endl;
    }
};

// Afgeleide klasse Kat
class Kat : public Dier {
public:
    void miauwen() {
        cout << "De kat miauwt!" << endl;
    }
};
```

Wat zijn goede voorbeelden?

- Dieren
- Hoofdklasse
- Overerving

```
32  int main() {
33      Hond mijnHond;
34      Kat mijnKat;
35
36      // Hond gebruikt functies van Dier
37      mijnHond.eten();
38      mijnHond.slapen();
39      mijnHond.blaffen();
40
41      cout << endl;
42
43      // Kat gebruikt functies van Dier
44      mijnKat.eten();
45      mijnKat.slapen();
46      mijnKat.miauwen();
47
48      return 0;
49  }
```

```
Het dier eet voedsel.
Het dier slaapt.
De hond blaft!

Het dier eet voedsel.
Het dier slaapt.
De kat miauwt!
```

Grotere schaal voorbeeld

- Dierentuin

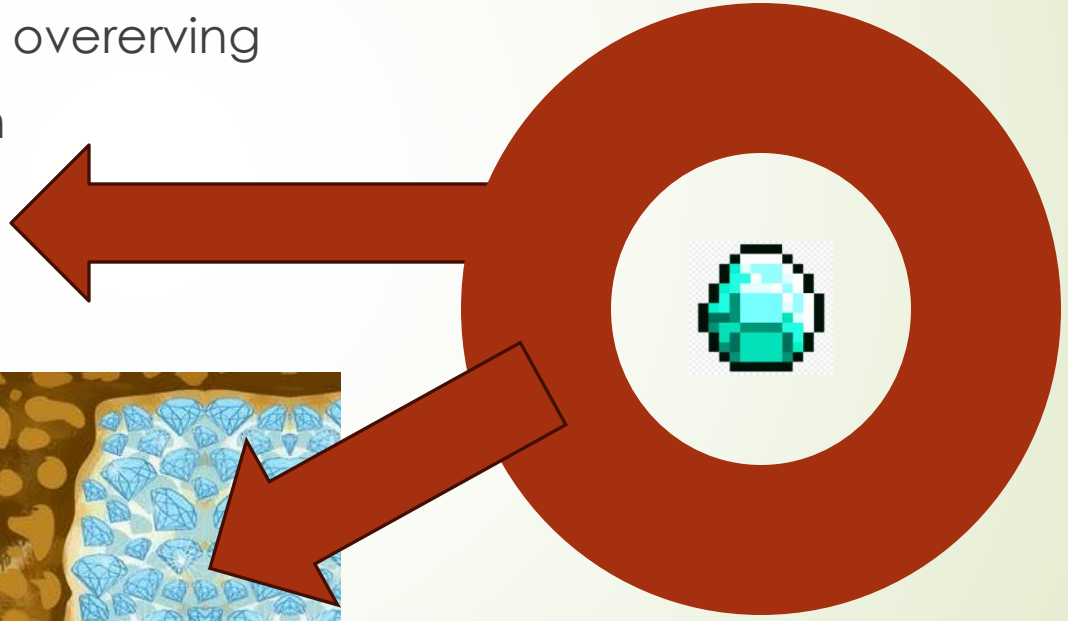
```cpp
// ----------------------
// Basisklasse (ouderklasse)
// ----------------------
class Dier {
protected:
    string naam;
    string voedsel;

public:
    Dier(string n, string v) : naam(n), voedsel(v) {}

    void eten() {
        cout << naam << " eet " << voedsel << "." << endl;
    }

    void slapen() {
        cout << naam << " slaapt." << endl;
    }

    // Virtuele functies = polymorfisme
    virtual void maakGeluid() = 0; // abstract, moet door sub
    virtual void bewegen() = 0;    // abstract, moet door sub

    virtual ~Dier() {} // virtual destructor voor correct ge
};
```

```cpp
// ----------------------
// Subklassen per categorie
// ----------------------
class Zoogdier : public Dier {
public:
    Zoogdier(string n, string v) : Dier(n, v) {}
    void bewegen() override {
        cout << naam << " loopt op vier poten." << endl;
    }
};

class Vogel : public Dier {
public:
    Vogel(string n, string v) : Dier(n, v) {}
    void bewegen() override {
        cout << naam << " vliegt door de lucht." << endl;
    }
};
```

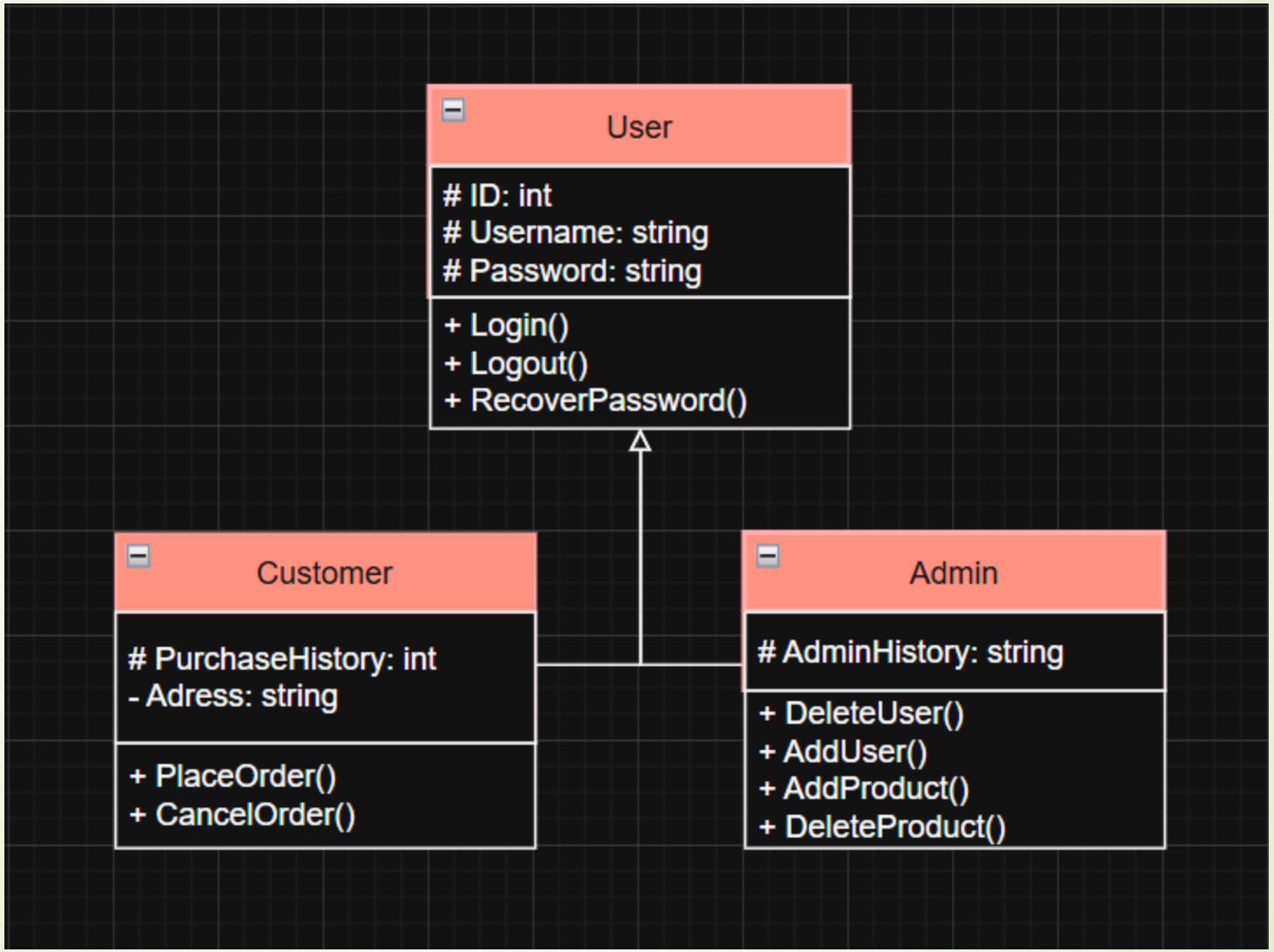
```cpp
// ----------------------
// Specifieke dieren
// ----------------------
class Hond : public Zoogdier {
public:
    Hond(string n) : Zoogdier(n, "vlees") {}
    void maakGeluid() override {
        cout << naam << " blaft!" << endl;
    }
};

class Kat : public Zoogdier {
public:
    Kat(string n) : Zoogdier(n, "vis") {}
    void maakGeluid() override {
        cout << naam << " miauwt!" << endl;
    }
};

class Papegaai : public Vogel {
public:
    Papegaai(string n) : Vogel(n, "zaden") {}
    void maakGeluid() override {
        cout << naam << " praat en maakt geluiden!" << endl;
    }
};

class Adelaar : public Vogel {
public:
    Adelaar(string n) : Vogel(n, "vlees") {}
    void maakGeluid() override {
        cout << naam << " krijst!" << endl;
    }
};
```

Waar gaat het vaak fout?

- Laptop gister niet opgeladen en kan inheritance dus niet gebruiken
- Verkeerd gebruik van overerving
- Te diepe hiërarchieën
- Diamond probleem
- Te sterke koppeling



UML Diagram voorbeeld inheritance



Bronnen

- https://www.uml-diagrams.org/inherited-property.html
- https://en.cppreference.com/book/intro/inheritance (De officiele referentie in cpp)
- ChatGPTHoeGebruikJeInheritanceInEchteProjecten?
- https://www.geeksforgeeks.org/cpp/inheritance-in-c (wat betekent het in c++?)
- https://30dayscoding.com/blog/advantages-and-disadvantages-of-inheritance-in-cpp (voor en nadelen)
- https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/
- ChatGPT en Andere generatieve AI
- https://stackoverflow.com/questions/418308/what-does-the-symbol-mean-in-a-uml-class-diagram
- https://stackoverflow.com/questions/406081/why-should-i-avoid-multiple-inheritance