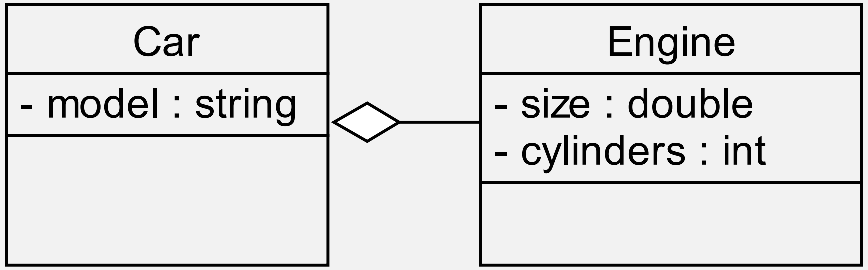
BUILDING AGGREGATION: WHOLE-PART WITH POINTERS

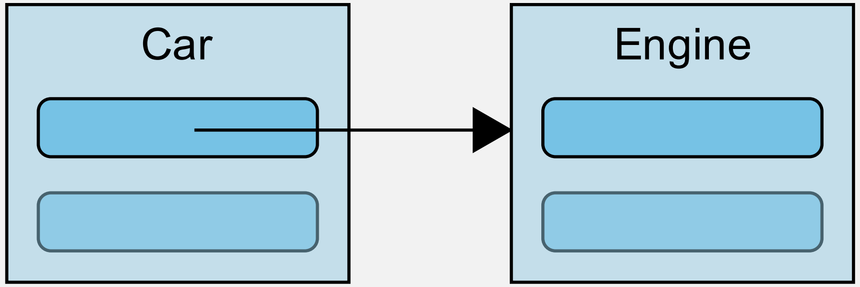
The whole *can* create its parts,

or the program can create the parts

Delroy A. Brinkerhoff

AGGREGATION



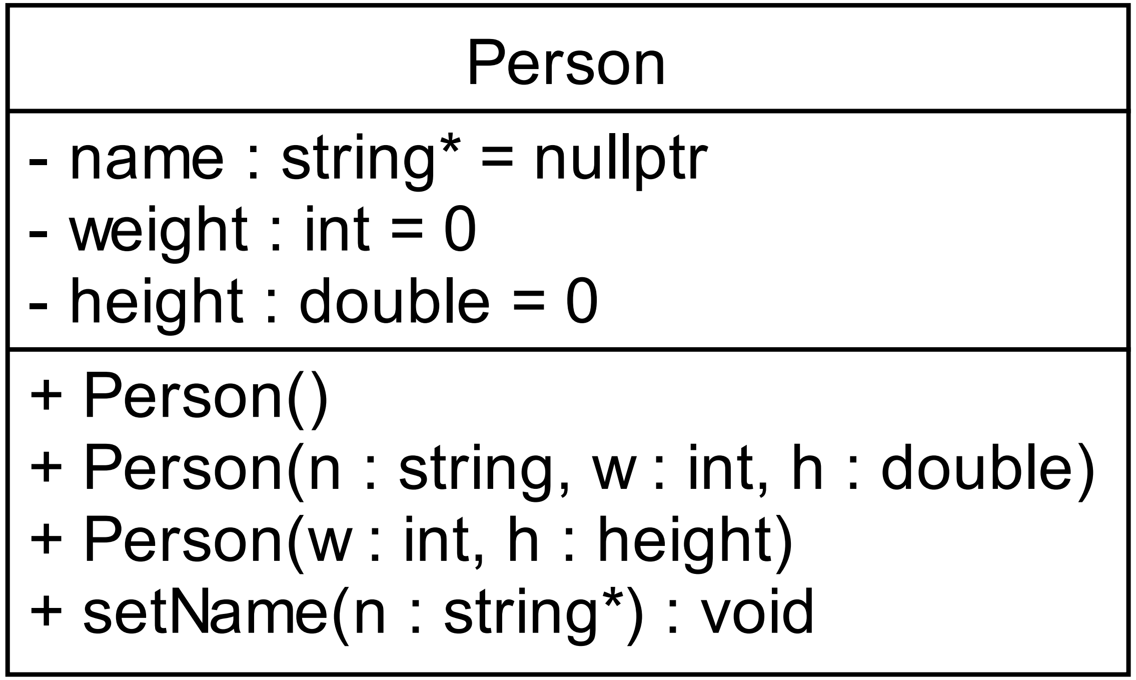


C++ implements aggregation with pointer member variables

- Variables are not shown as class attributes in UML diagrams
- Programmers translate the aggregation connector into a variable
- Variables are defined in class scope in the whole class

A pointer in the whole object points to an instance of the part object

PERSON CLASS



Person

- name : string* = nullptr
- weight : int = 0
- height : double = 0

+ Person()
+ Person(n : string, w : int, h : double)
+ Person(w : int, h : height)
+ setName(n : string*) : void

PERSON CLASS MEMBER FUNCTIONS

```
public:
    Person() : name(nullptr),
          weight(0), height(0) {}

    Person() {}

    Person(string n, int w, double h)
        : name(new string(n)),
          weight(w), height(h) {}

    Person(int w, double h)
        : name(nullptr),
          weight(w), height(h) {}
```

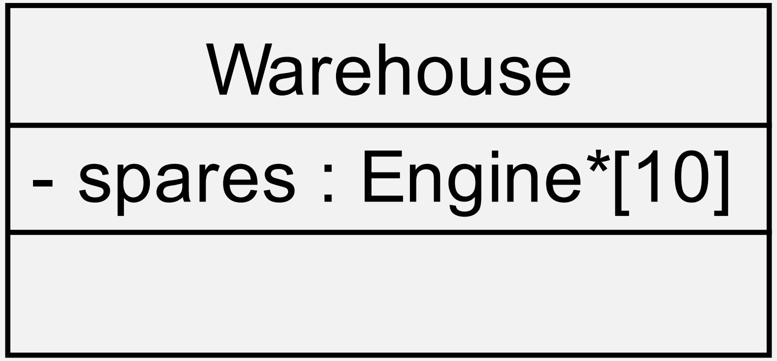
```
void setName(string* n)
{
    if (name != nullptr)
        delete name;
    name = n;
}
```

CONSTRUCTOR INITIALIZATION

```
class Engine
{
    private:
        double    size;
        int       cylinders;

    public:
        Engine(double s, int c)
            : size(s), cylinders(c) {}
};
```

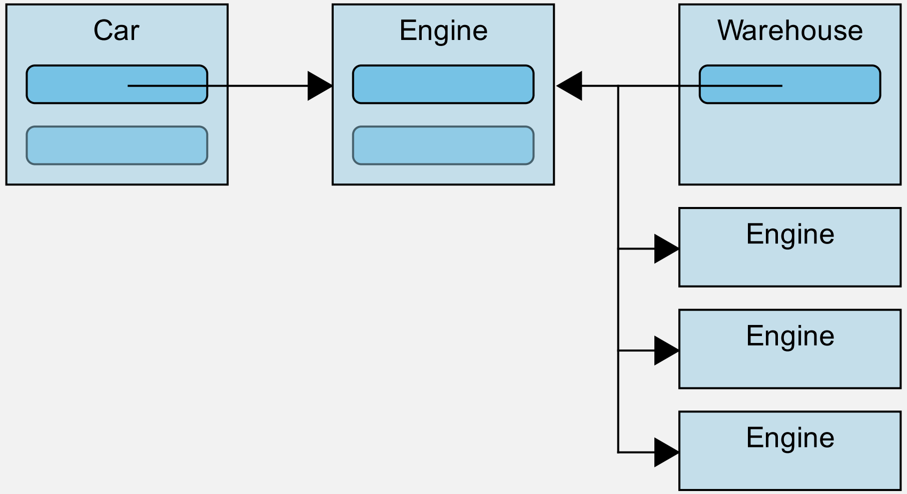
```
class Car
{
    private:
        Engine*   motor;
        string    model;

    public:
        Car(string m, double s, int c)
            : motor(new Engine(s, c)), model(m) {}

        Car(string m, Engine* e)
            : motor(e), model(m) {}
};
```

OWNERSHIP AND RESPONSIBILITY



Warehouse

- spares : Engine*[10]



Aggregation allows part sharing

When two wholes share a part

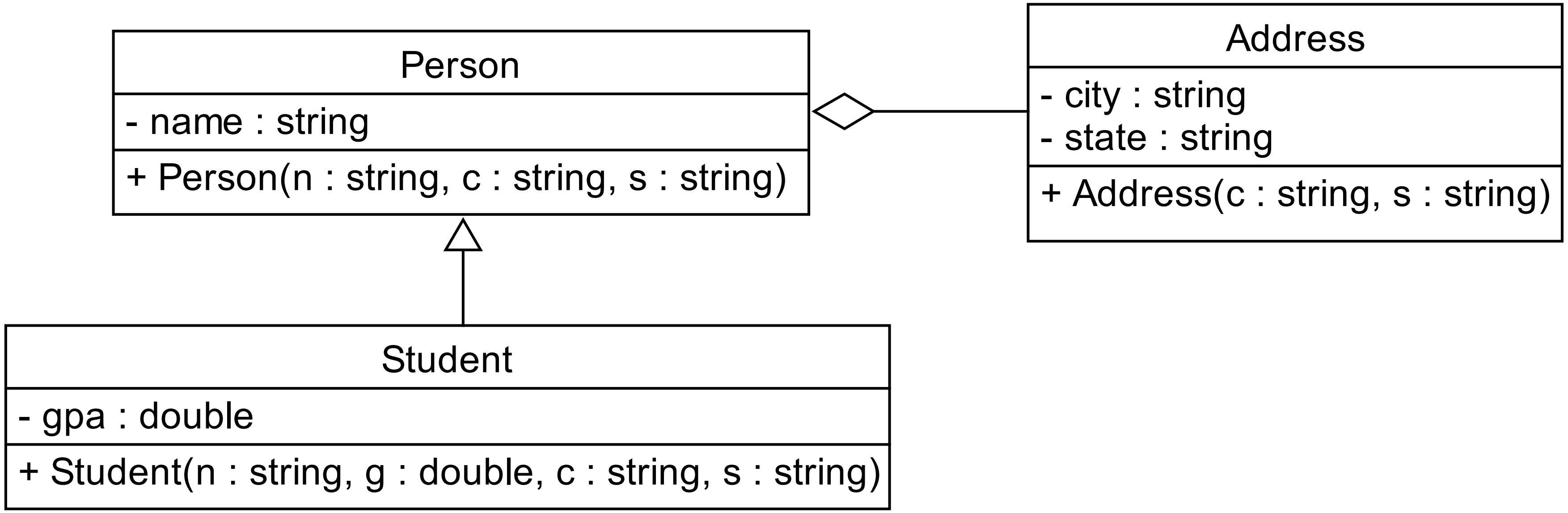
- Which whole “owns” the part?
- Which whole has responsibility for the part?
- Which whole destroys the part?

Creating the part is unimportant

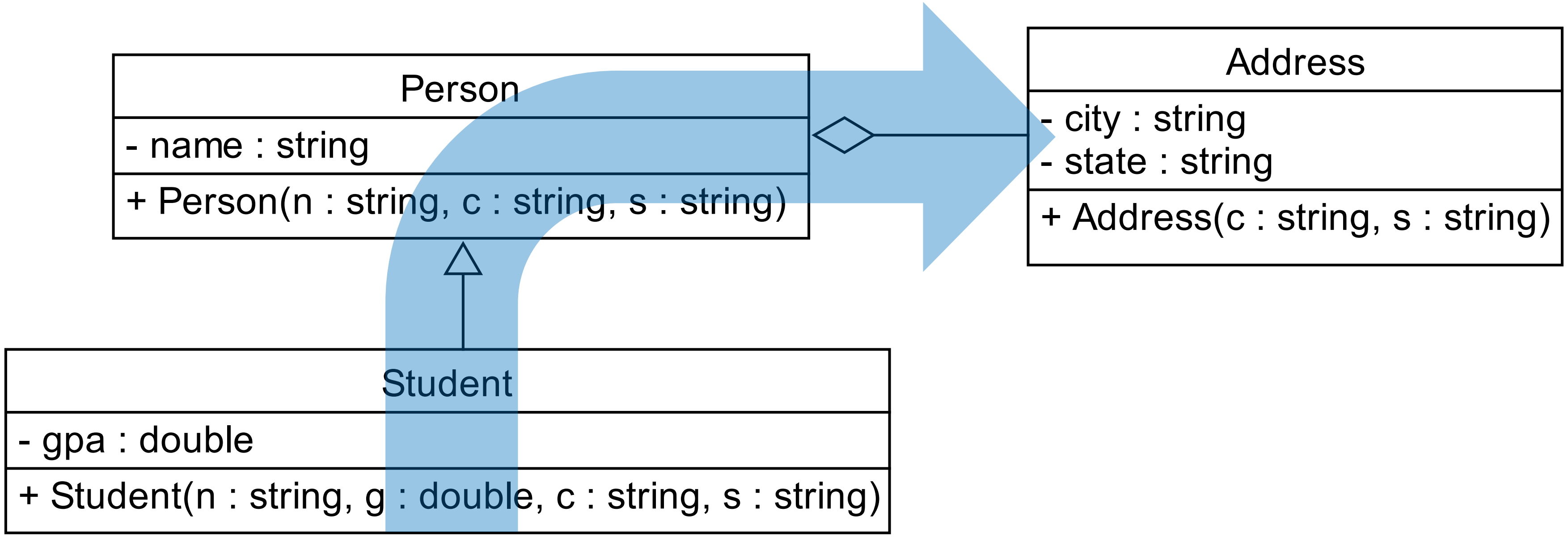
SETTER INITIALIZATION AND MANAGEMENT

```
class Car
{
    private:
        Engine*    motor = nullptr;
        string     model;

    public:
        Car(string s) : model(s) {}

        void set_motor(double s, int c);
        void set_motor(Engine* e);
};
```

```
void car::set_motor(double s, int c)
{
   if (motor != nullptr)
       delete motor;
   motor = new Engine(s, c);
}

void car::set_motor(Engine* e)
{
    if (motor != nullptr)
       delete motor;
    motor = e;
}
```


Person
- name : string
+ Person(n : string, c : string, s : string)
Address
- city : string
- state : string
+ Address(c : string, s : string)
Student
- gpa : double
+ Student(n : string, g : double, c : string, s : string)

INHERITANCE & AGGREGATION I



Person

- name : string

+ Person(n : string, c : string, s : string)

Address

- city : string
- state : string

+ Address(c : string, s : string)

Student

- gpa : double

+ Student(n : string, g : double, c : string, s : string)

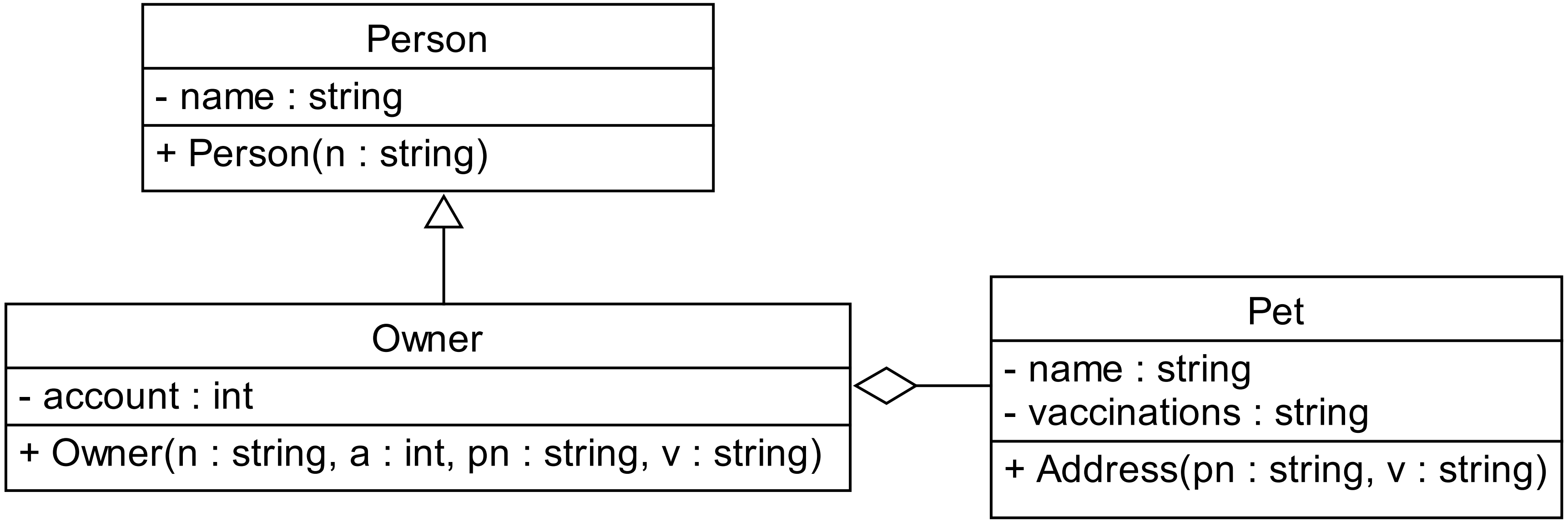
INHERITANCE & AGGREGATION I

MULTI-CLASS EXAMPLE 1

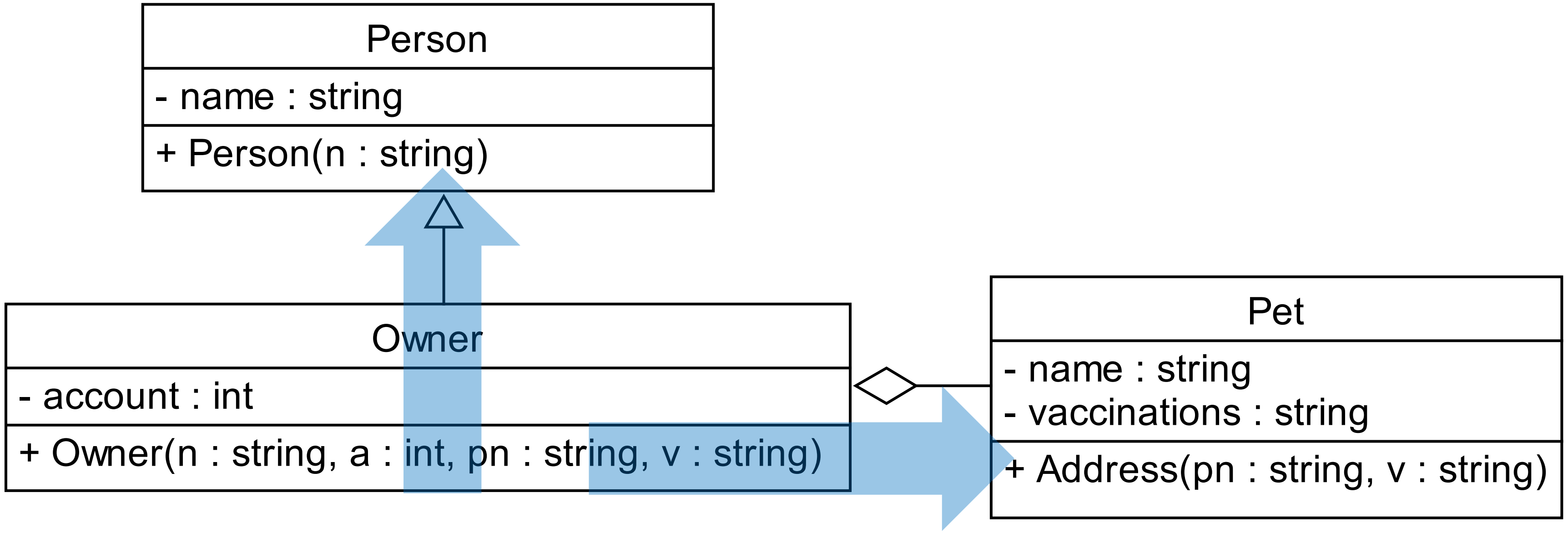
```
class Address
{
    private:
        string city;
        string state;
    public:
        Address(string c, string s)
        : city(c), state(s) {}
};
```

```
class Person
{
    private:
        string   name;
        Address* addr;    // aggregation
    public:
        Person(string n, string c, string s)
            : addr(new Address(c, s)),
              name(n) {}
};
```

```
class Student : public Person
{
    private:
        double gpa;
    public:
        Student(string n, double g, string c, string s) : Person(n, c, s), gpa(g) {}
};
```



INHERITANCE & AGGREGATION 2



Person

- name : string

+ Person(n : string)

Owner

- account : int

+ Owner(n : string, a : int, pn : string, v : string)

Pet

- name : string
- vaccinations : string

+ Address(pn : string, v : string)

INHERITANCE & AGGREGATION 2

MULTI-CLASS EXAMPLE 2

```
class Pet
{
    private:
        string name;
        string vaccinations;
    public:
        Pet(string pn, string v)
            : name(pn), vaccinations(v) {}
};
```

```
class Person
{
    private:
        string name;
    public:
        Person(string n) : name(n) {}
};
```

```
class Owner : public Person
{
    private:
        Pet* my_pet;     // aggregation
        int  account;
    public:
        Owner(string n, int a, string pn, string v)
            : Person(n), my_pet(new Pet(pn, v)), account(a) {}
};
```