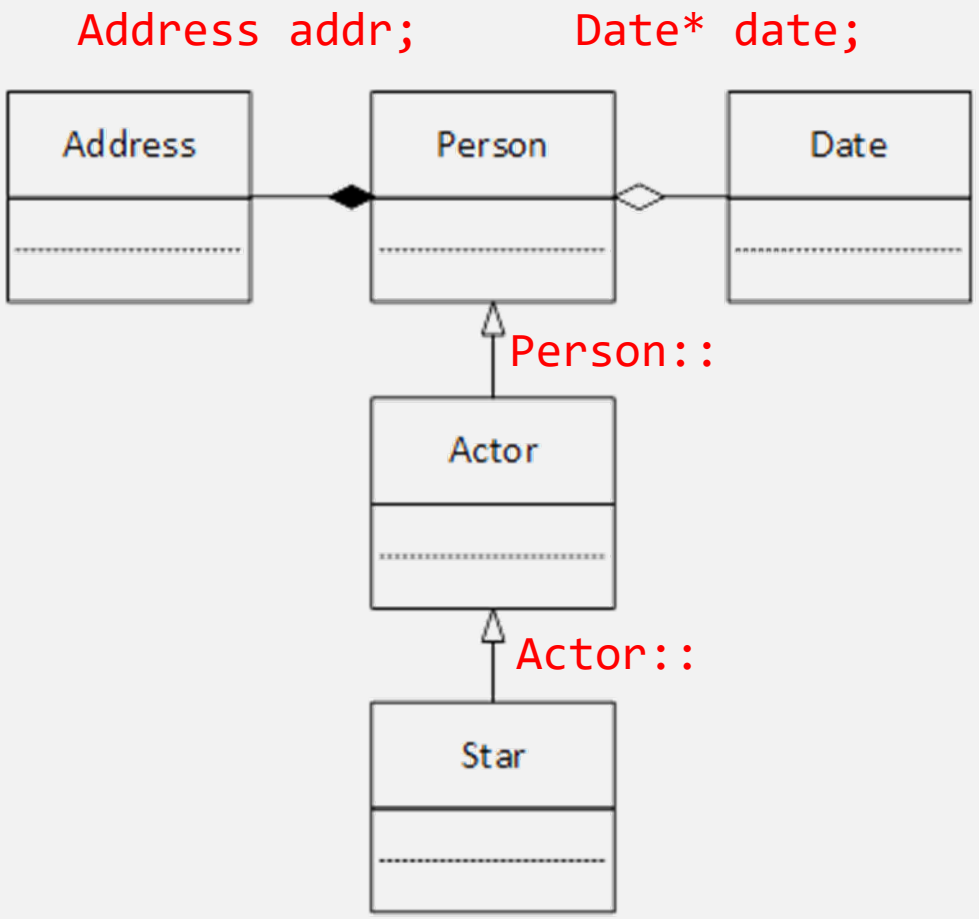
ACTOR 4 EXAMPLE

Chaining Operators And Function Calls

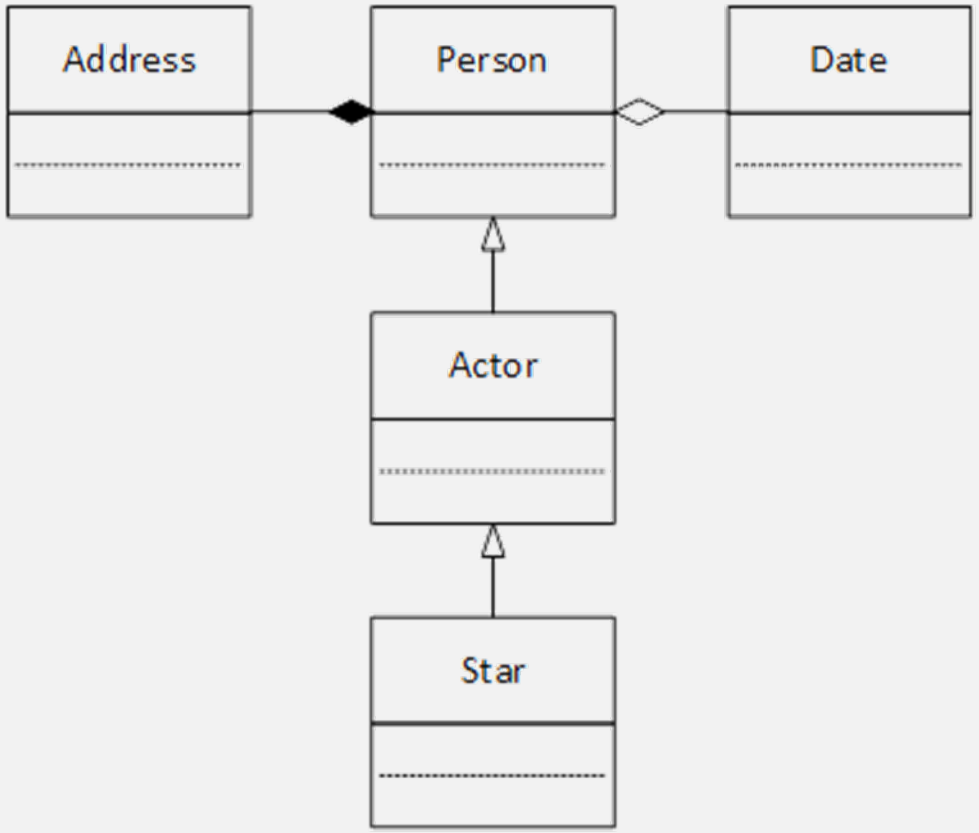
Delroy A. Brinkerhoff



ACTOR 4 UML CLASS DIAGRAM

C++ implements

- Inheritance with superclass name and the scope resolution operator
- Composition with a member name
- Aggregation with a member pointer



ACTOR 4 UML CLASS DIAGRAM

Focus on syntax

Constructor calls

Member function calls

I/O operators

- operator<< (extractor)
- operator>> (inserter)

CONSTRUCTOR CHAINING

- `Date() {}`
- `Address() {}`
- `Person() {}`
- `Actor() {}`
- `Star() {}`

- `Address(string s, string c) : street(s), city(c) {}`
- `Person(string n, string s, string c)`
  `: name(n), addr(s, c), date(nullptr) {}`
- `Actor(string n, string a, string s, string c)`
  `: Person(n, s, c), agent(a) {}`
- `Star(string n, string a, double b, string s, string c)`
  `: Actor(n, a, s, c), balance(b) {}`

Person SETTER FUNCTIONS

```
void setDate(int y, int m, int d)
{
    if (date != nullptr)
        delete date;
    date = new Date(y, m, d);
}
```

```
void setDate(Date* d)
{
    if (date != nullptr)
        delete date;
    date = d;
}
```

CHAINING FUNCTIONS WITH INHERITANCE

Actor

```
void display()
{
    Person::display();
    cout << agent << endl;
}
```

Star

```
void display()
{
    Actor::display();
    cout << balance << endl;
}
```

- The `display` function is simple but illustrates the general member function calling syntax

Person display FUNCTION

- Composition uses the member variable's name and the dot operator
- Aggregation
  - Should check for a null pointer
  - Uses the pointer member's name and the arrow operator

```
void display()
{
    cout << name << endl;
    addr.display()
    if (date != nullptr)
        date->display();
}
```

```
friend ostream& operator<<(ostream& out, Star& me)
{
    out << (Actor &)me << me.balance << endl;
    return out;
}

friend istream& operator>>(istream& in, Star& me)
{
    in >> (Actor &)me;
    cout << "Balance: ";
    in >> me.balance;
    return in;
}
```

Star I/O FUNCTIONS

```
friend ostream& operator<<(ostream& out, Actor& me)
{
    out << (Person &)me << me.agent << endl;
    return out;
}

friend istream& operator>>(istream& in, Actor& me)
{
    in >> (Person &)me;
    cout << "Agent: ";
    getline(in, me.agent);
    return in;
}
```

Actor I/O FUNCTIONS

```
friend ostream& operator<<(ostream& out, Person& me)
{
    out << me.name << " " << me.addr << endl;
    if (me.date != nullptr)
        out << *me.date << endl;
    return out;
}

friend istream& operator>>(istream& in, Person& me)
{
    cout << "Name: ";
    getline(in, me.name);
    in >> me.addr;
    Date* d = new Date;
    in >> *d;
    me.setDate(d);
    return in;
}
```

Person I/O FUNCTIONS

INSTANTIATING OBJECTS AND CALLING FUNCTIONS

```
Star s1("John Wayne",
    "Cranston Snort",
    50000000, "115 Elm",
    "Ogden");

s1.setDate(1960, 12, 25);
s1.display();
cout << s1 << endl;
```

```
Star* s4 = new Star;

cin >> *s4;
Date* d2 = new Date;
cin >> *d2;
s4->setDate(d2);
s4->display();
cout << *s4;
```