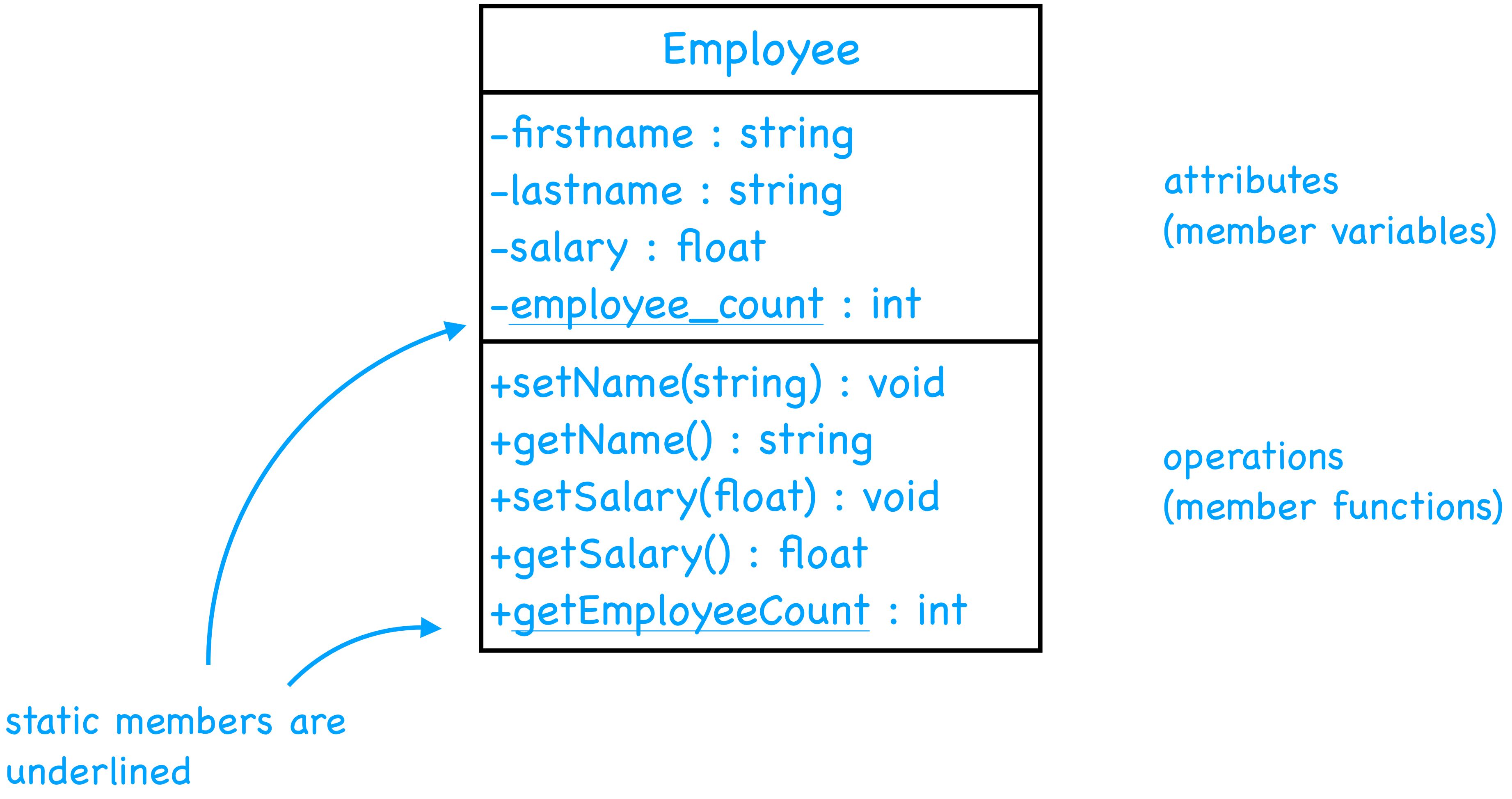
How to model an employee?

UML Class Diagram

Encapulation of ‘state’ and ‘behavior’



How will we use Employee objects?

main.cpp

```
#include <iostream>
#include <string>
#include <vector>
#include "Employee.hpp"

using namespace std;

int Employee::employee_count = 0;

int main()
{
    Employee e1("Jack", "Black", 35000.0);
    Employee e2("Tom", "Jones", 25000.0);

    vector<Employee> evec;
    evec.push_back(e1);
    evec.push_back(e2);

    for (int i = 0; i < Employee::GetEmployeeCount(); i++)
        cout << evec[i].GetName() << ' ' << evec[i].GetSalary()  << endl;
}
```

initialize the static class variable

create some Employee objects

create a vector of Employee objects (vector is like Java's ArrayList)

(can use evec.size() instead)

Employee.hpp

```cpp
#ifndef Employee_hpp
#define Employee_hpp

#include <string>

using namespace std;

class Employee
{
public:
    Employee();
    Employee(string fn, string ln, float sal);
    void SetName(string fn, string ln) {first_name = fn; last_name = ln;}
    string GetName() {return last_name + ',' + first_name;}
    void SetSalary(float sal) {salary = sal;};
    float GetSalary();
    static int GetEmployeeCount(){return employee_count;}
private:
    string first_name;
    string last_name;
    float salary;
    static int employee_count;
};

#endif /* Employee_hpp */
```

Employee.cpp

```cpp
#include "Employee.hpp"

Employee::Employee()
{
    first_name = " ";
    last_name = " ";
    salary = 0.0;
    employee_count++;
}

Employee::Employee(string fn, string ln, float sal)
{
    first_name = fn;
    last_name = ln;
    salary = sal;
    employee_count++;
}

float Employee::GetSalary()
{
    return salary;
}
```

Dynamically Allocated Employee Objects

```
#include <iostream>
#include <string>
#include <vector>
#include "Employee.hpp"

using namespace std;

int Employee::employee_count = 0;

int main()
{
    Employee* e1 = new Employee("Jack", "Black", 35000.0);
    Employee* e2 = new Employee("Tom", "Jones", 25000.0);
    Employee* e3 = new Employee("Jan", "Smith", 28000.0);

    vector<Employee*> evec;
    evec.push_back(e1);
    evec.push_back(e2);
    evec.push_back(e3);

    for (int i = 0; i < evec.size(); i++)
        if ( evec[i]->GetName() == "Jones,Tom")
        {
            delete evec[i];
            evec.erase(evec.begin() + i);
        }

    for (int i = 0; i < evec.size(); i++)
        cout << evec[i]->GetName() << ' ' << evec[i]->GetSalary()  << endl;
}
```

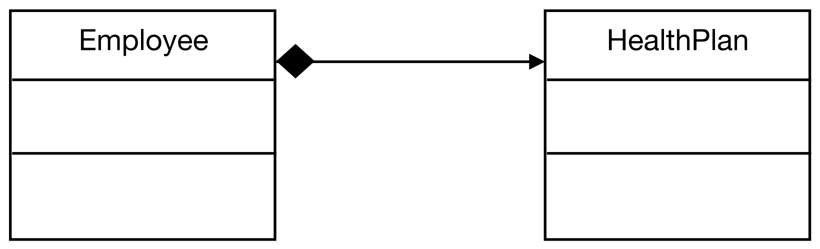
delete the `Employee` object

remove the element from the vector

```cpp
#ifndef Employee_hpp
#define Employee_hpp

#include <iostream>
#include <string>

using namespace std;

class Employee
{
public:
    Employee();
    Employee(string fn, string ln, float sal);
    ~Employee(){cout<<"dtor\n";}
    void SetName(string fn, string ln) {first_name = fn; last_name = ln;}
    string GetName() {return last_name + ',' + first_name;}
    void SetSalary(float sal) {salary = sal;};
    float GetSalary();
    static int GetEmployeeCount(){return employee_count;}
private:
    string first_name;
    string last_name;
    float salary;
    static int employee_count;
};

#endif /* Employee_hpp */
```

a destructor for `Employee` objects, called if the object is deleted or goes out of scope

UML Class Diagram for Composition



an Employee **'has a'** HealthPlan

```cpp
#include <iostream>
#include <string>
#include <vector>
#include "Employee.hpp"
#include "HealthPlan.hpp"

using namespace std;

int Employee::employee_count = 0;

int main()
{
    HealthPlan* anthem = new HealthPlan("Anthem", 1500.0, 200.0);
    Employee* e1 = new Employee("Jack", "Black", 35000.0);
    Employee* e2 = new Employee("Tom", "Jones", 25000.0);
    Employee* e3 = new Employee("Jan", "Smith", 28000.0, anthem);

    vector<Employee*> evec;
    evec.push_back(e1);
    evec.push_back(e2);
    evec.push_back(e3);

    for (int i = 0; i < evec.size(); i++)
        if ( evec[i]->GetName() == "Jones,Tom")
        {
            delete evec[i];
            evec.erase(evec.begin() + i);
        }

    for (int i = 0; i < evec.size(); i++)
        cout << evec[i]->GetName() << ' ' << evec[i]->GetHealthPremium()  << endl;
}
```

```cpp
#ifndef Employee_hpp
#define Employee_hpp

#include <iostream>
#include <string>
#include "HealthPlan.hpp"

using namespace std;

class Employee
{
public:
    Employee();
    Employee(string fn, string ln, float sal, HealthPlan* health_plan=NULL);
    ~Employee();
    void SetName(string fn, string ln) {first_name = fn; last_name = ln;}
    string GetName() {return last_name + ',' + first_name;}
    void SetSalary(float sal) {salary = sal;};
    float GetSalary() {return salary;};
    float GetHealthPremium();
    static int GetEmployeeCount(){return employee_count;}
private:
    string first_name;
    string last_name;
    float salary;
    HealthPlan* health_plan;
    static int employee_count;
};

#endif /* Employee_hpp */
```

HealthPlan.hpp

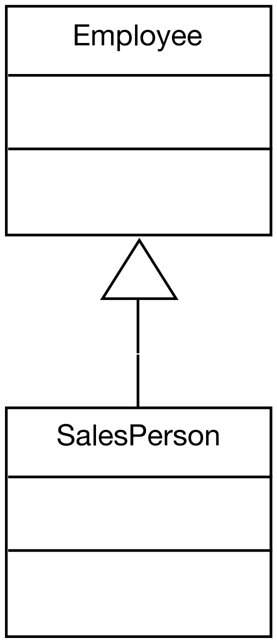
```
#ifndef HealthPlan_hpp
#define HealthPlan_hpp

#include <string>

using namespace std;

class HealthPlan
{
public:
    HealthPlan(string name, float premium, float copay);
    float GetPremium(){return premium;}
private:
    string name;
    float premium;
    float copay;
};
#endif /* HealthPlan_hpp */
```

HealthPlan.cpp

```
#include "HealthPlan.hpp"

HealthPlan::HealthPlan(string name, float premium, float copay)
{
    this->name = name;
    this->premium = premium;
    this->copay = copay;
}
```

We want to model a special type of Employee

Inheritance



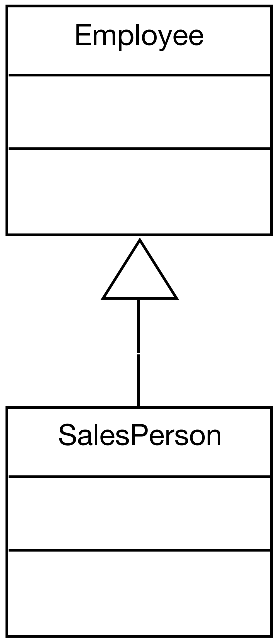
a SalesPerson **'is a'** Employee

a SalesPerson inherits all of the member variables and all of the member functions of Employee, but may add new members to specialize the class

We want to model a special type of Employee

<u>Inheritance</u>

the base class
(super class,
parent class)



the derived class
(subclass,
child class)

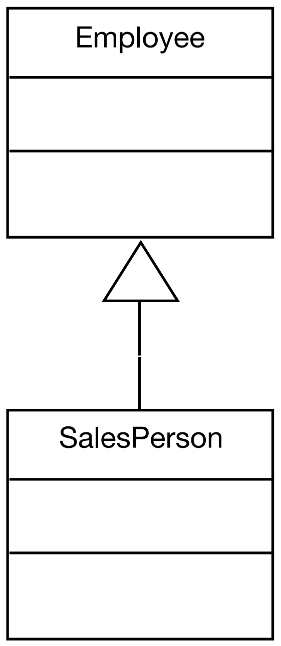
a SalesPerson **'is a'** Employee

a SalesPerson <u>inherits</u> all of the member variables
and all of the member functions of Employee,
but may add new members to specialize the class

We want to model a special type of Employee

Inheritance

the base class
(super class,
parent class)



the derived class
(subclass,
child class)

a SalesPerson **'is a'** Employee

a SalesPerson inherits all of the member variables
and all of the member functions of Employee,
but may add new members to specialize the class

What's special about a SalesPerson?

Maybe the way that their pay is calculated...

```
#ifndef SalesPerson_hpp
#define SalesPerson_hpp

#include <iostream>
#include "Employee.hpp"

class SalesPerson : public Employee
{
public:
    SalesPerson(string fn, string ln, float sal, HealthPlan* health_plan=NULL);
    void SetCommission(float comm) {commission = comm;};
    void SetNumSales(int ns) {num_sales = ns;};
    virtual float CalculatePay() {return salary + num_sales*commission;};
private:
    float commission;
    int num_sales;
};

#endif /* SalesPerson_hpp */
```

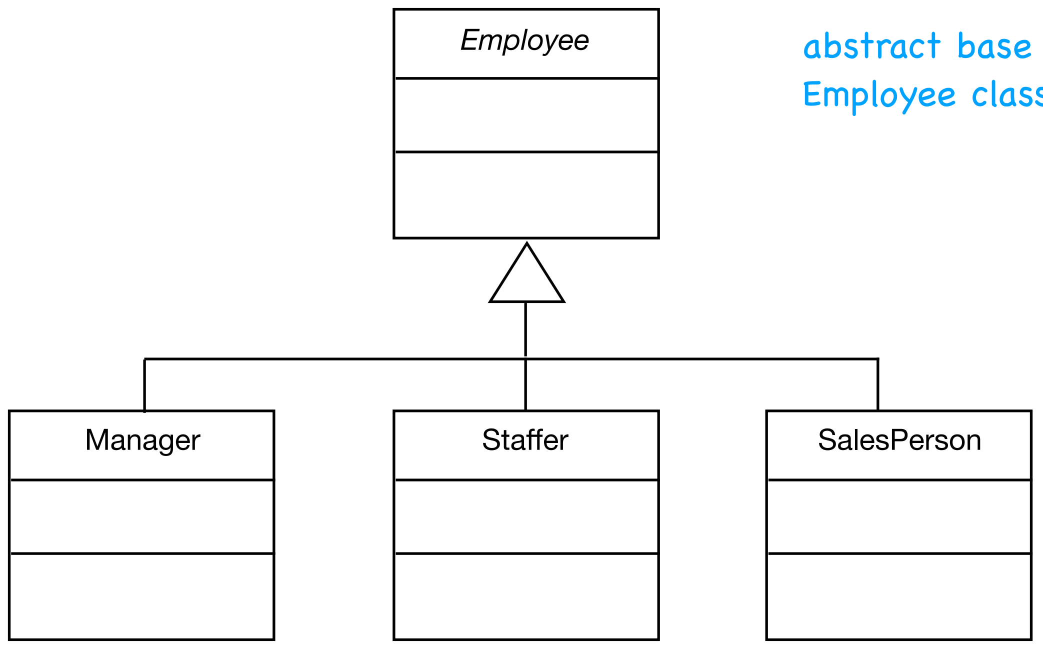
```
#include "SalesPerson.hpp"

SalesPerson::SalesPerson(string fn, string ln, float sal, HealthPlan*
health_plan):Employee(fn, ln, sal, health_plan)
{

    commission = 0.0;
    num_sales = 0;
}
```

```cpp
#ifndef Employee_hpp
#define Employee_hpp

#include <iostream>
#include <string>
#include "HealthPlan.hpp"

using namespace std;

class Employee
{
public:
    Employee();
    Employee(string fn, string ln, float sal, HealthPlan* health_plan=NULL);
    virtual ~Employee();
    void SetName(string fn, string ln) {first_name = fn; last_name = ln;}
    string GetName() {return last_name + ',' + first_name;}
    void SetSalary(float sal) {salary = sal;};
    float GetSalary() {return salary;};
    virtual float CalculatePay() = 0;
    float GetHealthPremium();
    static int GetEmployeeCount(){return employee_count;}
protected:
    string first_name;
    string last_name;
    float salary;
    HealthPlan* health_plan;
    static int employee_count;
};
#endif /* Employee_hpp */
```

```cpp
#include <iostream>
#include <string>
#include <vector>
#include "Employee.hpp"
#include "SalesPerson.hpp"
#include "HealthPlan.hpp"

using namespace std;

int Employee::employee_count = 0;

int main()
{
    HealthPlan* anthem = new HealthPlan("Anthem", 1500.0, 200.0);
    SalesPerson* e1 = new SalesPerson("Jack", "Black", 35000.0);
    SalesPerson* e2 = new SalesPerson("Tom", "Jones", 25000.0);
    SalesPerson* e3 = new SalesPerson("Jan", "Smith", 28000.0, anthem);
    e3->SetCommission(0.5);
    e3->SetNumSales(1000);

    vector<Employee*> evec;
    evec.push_back(e1);
    evec.push_back(e2);
    evec.push_back(e3);

    for (int i = 0; i < evec.size(); i++)
        if ( evec[i]->GetName() == "Jones,Tom")
        {
            delete evec[i];
            evec.erase(evec.begin() + i);
        }

    for (int i = 0; i < evec.size(); i++)
        cout << evec[i]->GetName() << ' ' << evec[i]->CalculatePay()  << endl;
}
```

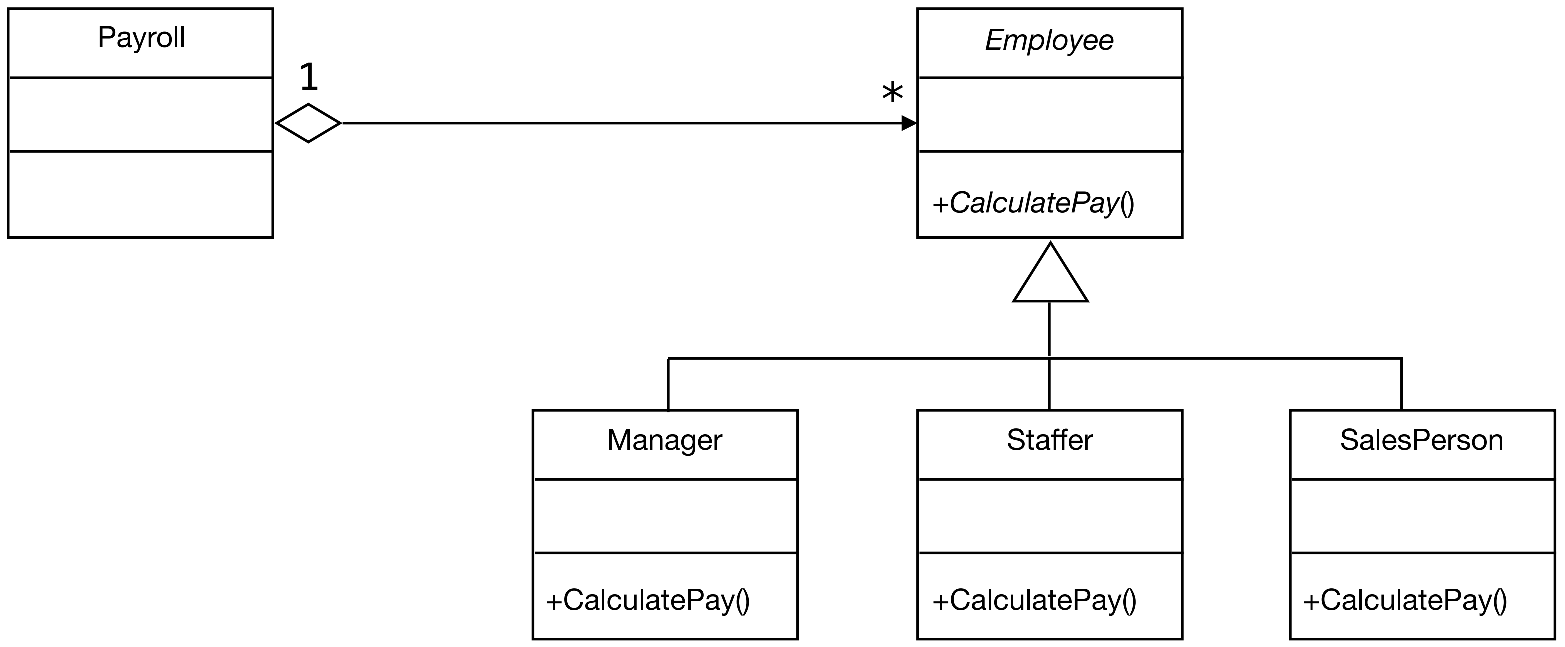
Polymorphism

UML Class Diagram of an <u>Inheritance Hierarchy</u>



abstract base class for the
Employee class hierarchy

UML Class Diagram for a Complete System



Requirements:

A system is needed to keep track of all employees at a company. The employee’s name, id number and birthday are kept along with the way their monthly pay is calculated. There are three types of employees: managers, who have a fixed salary, staffers who have an hourly wage, and salespersons who have a base salary and a commision for the number of units they sell.

(some) nouns -> objects

Requirements:

A system is needed to keep track of all employees at a company. The employee’s name, id number and birthday are kept along with the way their monthly pay is calculated. There are three types of employees: managers, who have a fixed salary, staffers who have an hourly wage, and salespersons who have a base salary and a commision for the number of units they sell.

(some) verb phrases -> member functions