# Наследование

Наследование (генерализация, inheritance, derivation, generalization) – отношение между классами «частное-общее».

Если класс B наследуется от класса A, значит любой объект класса B является также объектом класса A.
A – базовый класс, родитель (superclass, parent)
B – подкласс, производный класс (subclass derived class)

# Смысл наследования

- Производный класс наследует все свойства базового класса

- Производный класс может уточнять некоторые свойства базового класса, а также расширять набор свойств.

- Принцип подстановки Лисковой (Liskov Substitution Principle, LSP): всякий объект производного класса может быть использован, как объект базового класса.

# Реализация наследования

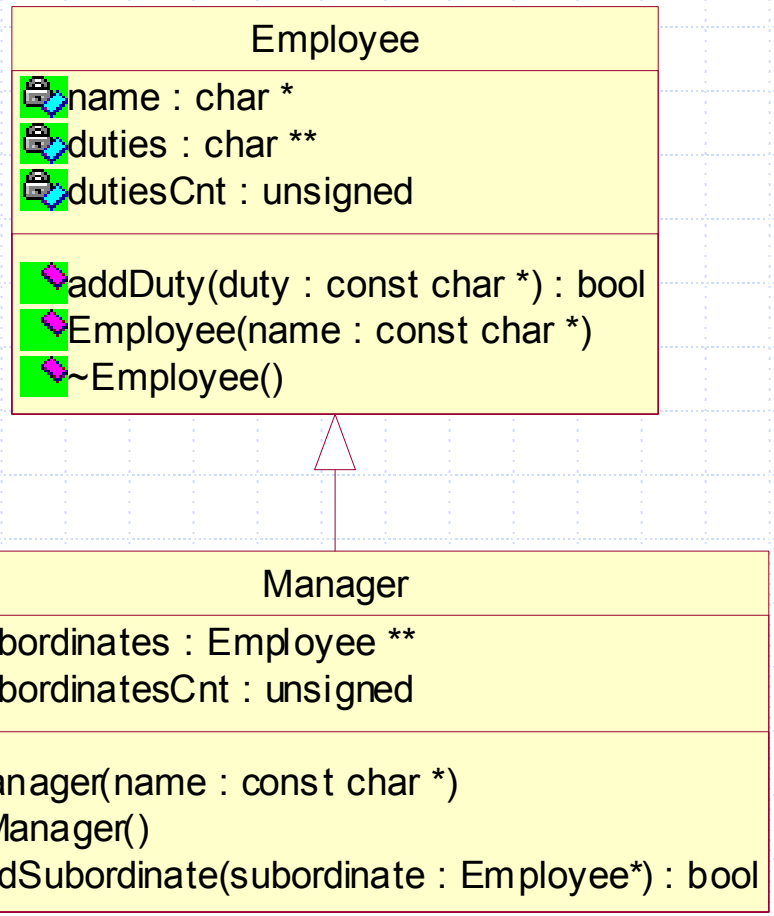
```
class Employee{
private:
        unsigned dutiesCnt;
        char ** duties;
        char * name;
public:
        Employee (const char *
            name);
        bool addDuty (const char *
            duty);
        virtual ~Employee ();
};
```

```
class Manager : public  Employee{
private:
        unsigned subordinatesCnt;
        Employee ** subordinates;
public:
        Manager (const char *
            name);
        bool addSubordinate (
            Employee * subordinate);
        virtual ~Manager ();
};
```

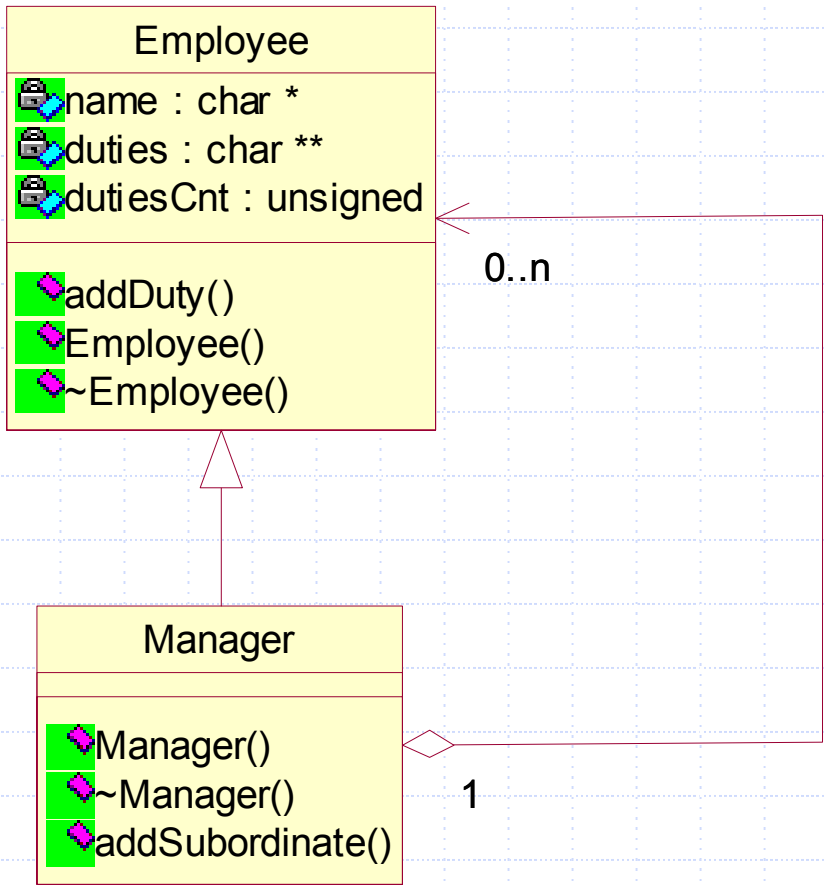
# Использование наследования

```
Manager vasya (“Вася”);
Employee * eptr = &vasya;
vasya.addDuty (“Мыть полы”);
eptr -> addDuty (“Выносить мусор”);

Employee petya (“Петя”);
vasya.addSubordinate (petya);
eptr -> addSubordinate (); //ошибка

Manager *mptr = &petya; //ошибка
```

# Нотация UML (Unified Modeling Language)



# Агрегация в UML



# Доступ к полям при наследовании

SuperClass

| Inheritance type | public | protected | private |
|---|---|---|---|
| public | | | |
| protected | | | |
| private | | | |

# Доступ к полям при наследовании

SuperClass

Inheritance type

| | public | protected | private |
|---|---|---|---|
| public | public | protected | X |
| protected | protected | protected | X |
| private | private | private | X |

# Конструкторы

```
Manager::Manager (const char* name) : Employee (name){
        subordinatesCnt = 0;
        subordinates = 0;
}
```

Порядок вызова конструкторов:

1. Конструкторы базовых классов
2. Конструкторы полей класса
3. Конструктор класса

# Деструкторы

Порядок вызова деструкторов:

1. Деструктор класса
2. Деструкторы полей
3. Деструкторы базовых классов

Если класс участвует в наследовании, то деструктор всегда должен быть виртуальным.

# Полиморфизм

Полиморфизм – способность объектов одного и того же класса в одних и тех же условиях по-разному реагировать на одинаковые сообщения.

Динамический полиморфизм – способ реализации полиморфизма, где объекты производных классов переопределяют (доопределяют) поведения базовых.

# Виртуальные методы

```
class A{
public:
        /*virtual*/ void printHello () const;
        virtual ~A ();
};
class B : public A{
public:
        /*virtual*/ void printHello () const;
        virtual ~B ();
};
void A::printHello () const{
        printf ("Hello (A)\n");
}
void B::printHello () const{
        printf ("Hello (B)\n");
}
```

```
B b;
A &a = b;
a.printHello ();
b.printHello ();
```

```
Hello (A)
Hello (B)
```

виртуальные методы

```
Hello (B)
Hello (B)
```

невиртуальные методы

# Таблица виртуальных функций

Всякий класс, содержащий хотя-бы один виртуальный метод имеет таблицу виртуальных функций.

В любом объекте такого класса есть скрытый атрибут, содержащий указатель на таблицу виртуальных функций того класса, из которого он был создан.

При приведении объекта к базовому классу указатель не изменяется.

# Соблюдение принципа подстановки Лисковой

```
void A::printHello () const{
        printf (“Hello (A)\n”);
}
void B::printHello () const {
        A::printHello ();
        printf (“Hello (B)\n”);
}
```

```
B b;
A &a = b;
a.printHello ();
b.printHello ();
```

```
Hello (A)
Hello (B)
Hello (A)
Hello (B)
```

# Последовательность вызовов конструкторов и деструкторов

```
class A{
        A (){
                printf (“A constructor”);
        }
        virtual ~A (){
                printf (“A destructor”);
        }
};
class B : public B{
        B (){
                printf (“B constructor”);
        }
        virtual ~B (){
                printf (“B destructor”);
        }
};
```

```
int main (){
        B b;
        return 0;
}
```

Что напечатается?