

# Наследяване - основни концепции

Обектно ориентирано програмиране - семинар 2022/2023

# Проблем:

Искаме да имаме софтуерна компания, която трябва да има следните типове служители и информация свързана с тях:

- Developer - име, години, департамент, проект, основен програмен език
- QA - име, години, департамент, тип (manual/automation/hybrid)
- DevOps - име, години, проект
- BusinessDev - име, години, проект, имейл

Кои полета се повтарят? Как можем да предотвратим тази дубликация?




# Наследяване

Механизъм за построяване на отношения между класовете. Този механизъм позволява даден клас да наследи друг. Класът, който наследява се нарича наследник или производен клас. Класът, от който се наследява се нарича предшественик или базов (основен) клас.



## Наследяване (2)

Процесът на наследяване по отношение на базовите и производните класове се изразява в следното:

- производният клас наследява структурата на основния т.е. неговите полета методи и свойства;
  - производният получава достъп до наследените от основния клас членове;
  - освен наследените, производният клас може да има свои собствени членове (полета, свойства и методи);
  - достъпът на методите на производния клас до наследените членове от основния се регламентират от модификаторите за достъп `public`, `private` и `protected`
  - производният клас, от своя страна може да е базов за други класове;
  - един клас може да е основен за няколко производни. По този начин се получава йерархична структура, породена от механизма на наследяване;
- 

# Наследяване срещу композиция

- Композицията представя релацията “has” -> “Car has Tyres”
- Наследяването представя релацията “is a” -> “Cow is a Mammal”

В общия случай когато трябва да избираме дали да използваме композиция или наследяване, използваме композиция. Съществуват ситуации в които обаче композицията не ни дава достатъчно ниво на абстракция (именно при използване на абстракция) и тогава сме длъжни да използваме наследяване.



# Синтаксис

```
class <ClassName> : public <BaseClassName> {  
    //тяло  
}
```



# Модификатори за достъп (Енкапсулация)

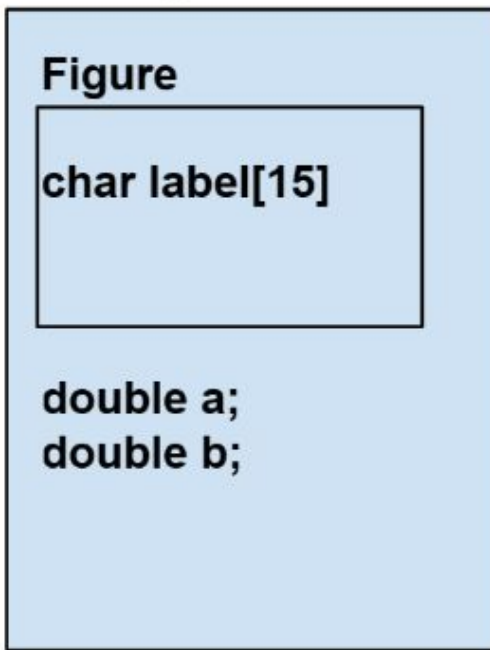
- public - достъп навсякъде в кода
- private - достъп само в тялото на класа
- protected - достъп само в тялото на класа и в тялото на класовете преки наследници



# Наследяване - пример

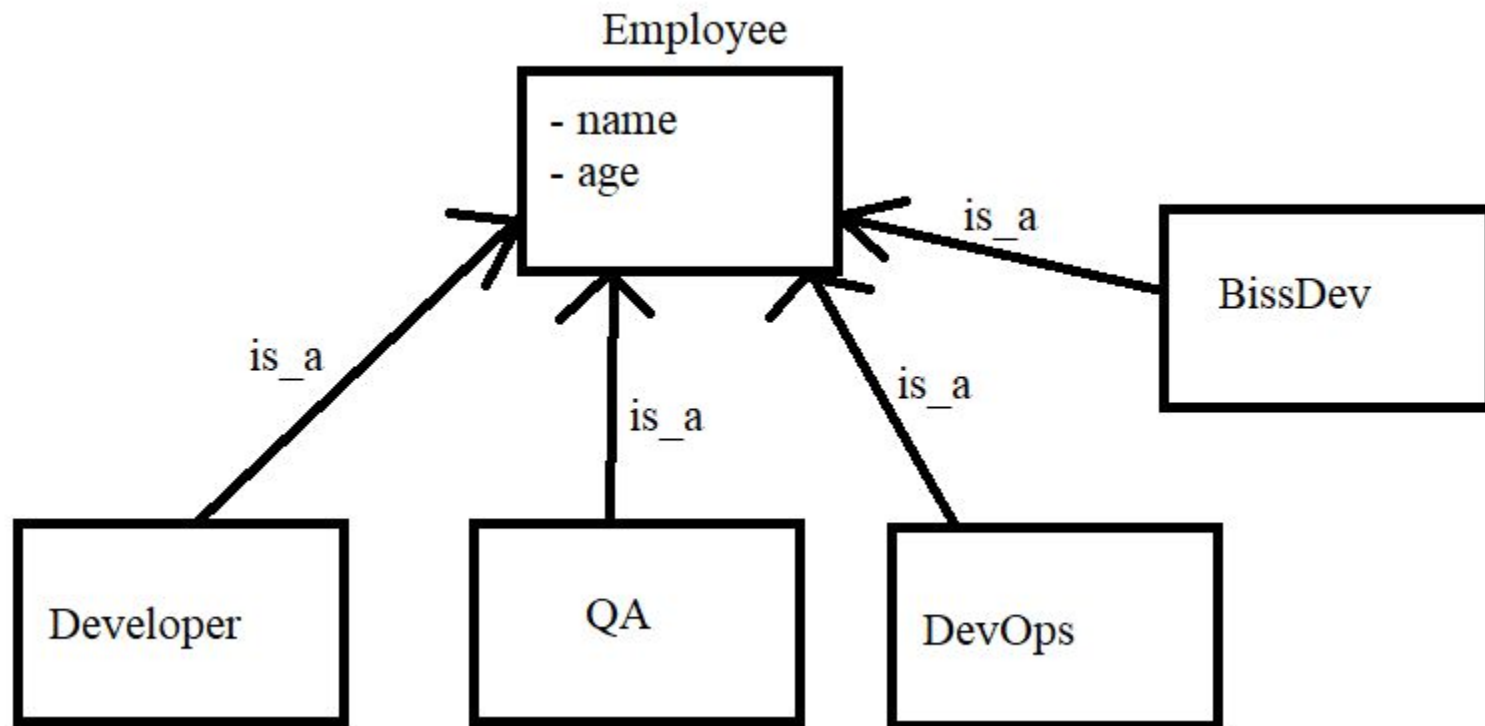
```
class Rectangle : public Figure
{
    public:
        double a,b;
        double surface ()
        {return a*b;}
};
```

## Rectangle





## Решение на проблема



# Множествено наследяване

Множественото наследяване е функционалност на C ++, където един клас може да наследява от повече от един клас. Конструкторите на наследените класове се викат в същия ред, в който са наследени.

```
class C : public A, public B ->
```

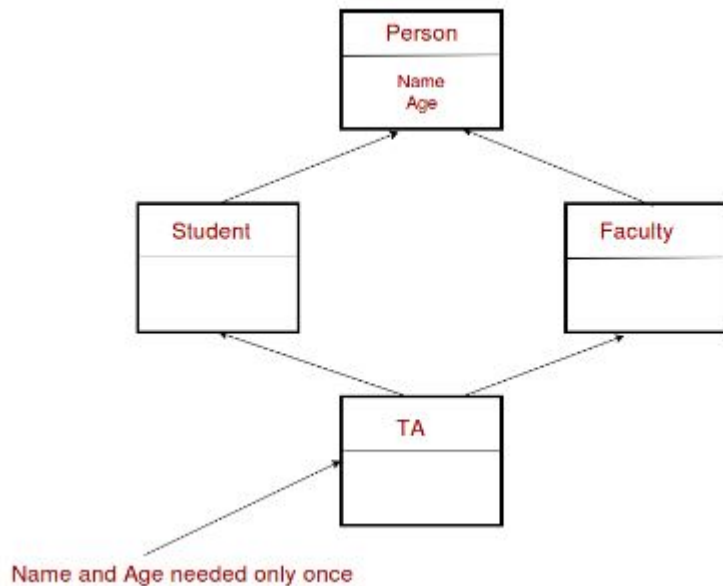
```
C() {
```

```
A(); B();
```

```
}
```



# Diamond problem:

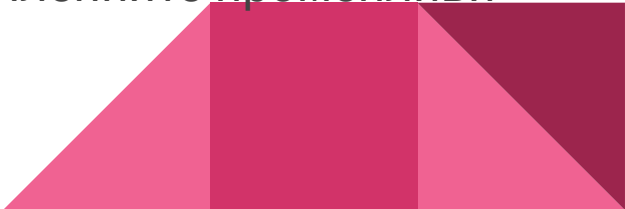


- C++ е един от малкото езици, които позволяват множественото наследяване, именно поради диамантения проблем. По тази причина в други езици се въвеждат интерфейсите.

# Виртуални базови класове (Виртуално наследяване)

Виртуалното наследяване е техника на C++, която гарантира, че само едно копие от променливите на членовете на базовия клас се наследява от производни от второ ниво (известни още като класове, получени от внуци). Без виртуално наследяване, ако два класа B и C наследяват от клас A, а клас D наследява както B, така и C, тогава D ще съдържа две копия на променливите на членовете на A: една чрез B и една чрез C.

Вместо това, ако класовете B и C наследяват виртуално от клас A, тогава обектите от клас D ще съдържат само един набор от членните променливи от клас A.



# Виртуално наследяване - пример

```
1  struct Person {
2      virtual ~Person() = default;
3      virtual void speak() {}
4  };
5
6  // Two classes virtually inheriting Person:
7  struct Student: virtual Person {
8      virtual void learn() {}
9  };
10
11 struct Worker: virtual Person {
12     virtual void work() {}
13 };
14
15 // A teaching assistant is still a student and the worker
16 struct TeachingAssistant: Student, Worker {};
```