Тема 10

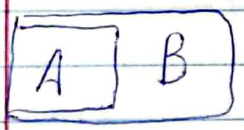
Наследяване. Видове наследяване. Параметри на функции (указатели и референции). Конструктори и деструктори при наследяване.

Наследяване (is-a relationship)
При създаване на нов клас, който има общи компоненти и поведение с вече дефиниран клас, вместо да дефинира повторно тези компоненти и поведение, можем да го обявим за наследник на вече дефиниран клас. При наследяването се наследяват член-данните и методите на основните класове и се получава достъп до всеки от наследените компоненти на основните класове.

Наследяването се извършва по следния начин:
class B: (тип на наследяване) A

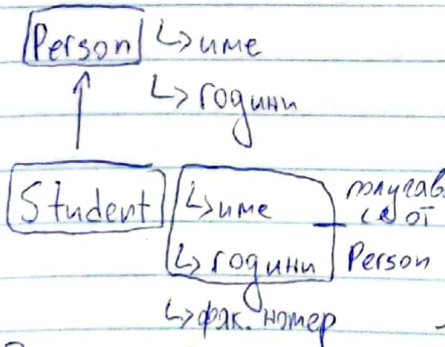
В нашия пример класът B е наследник на клас A. Обектите от класовете изглеждат така:

(Картинките са само за илюстративна цел.)



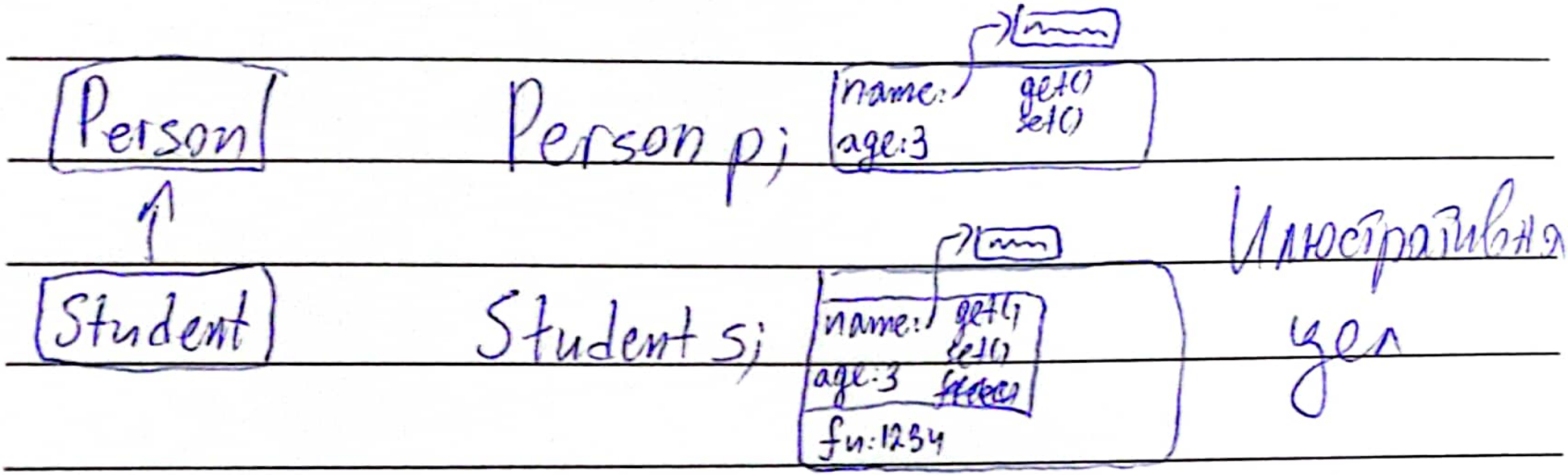
След като B е наследник на клас A, то при създаване на обект от тип B се създава обект от тип A, който е част от обекта от тип B.

Графично пример:



Student наследява Person (чрез наследяването Student "открадва" данните на Person - в случая (години, име и функциите, които се грижат за тях)

За данните, които са приети от Person - за тях се грижи Person
За останалите данни, които са в Student се грижи Student



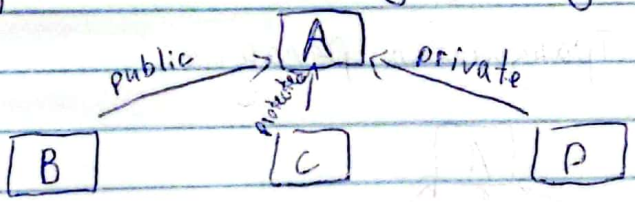
Типове на наследяване:
- private
- public
- protected

Типът на наследяване може да се пропусне в декларацията на класа/структурата, като по default наследяването на клас е private, а наследяването на структура публик.

Забележка: Наследяването на приятелски класове не е транзитивна релация, т.е. ако клас A е приятелски клас за клас B и клас B е приятелски клас за клас C, то нито клас A е приятелски клас за клас C, нито клас C е приятелски клас за клас A.

Нека имаме следния клас: И следните видове наследяване:

```
class A
{
public:
  int x;
protected:
  int y;
private:
  int z;
};
```



Тогава наследените член-данни ще са:

```
class B: public A
{
  x - public
  y - protected
  z - не е достъпен
};
```

```
class C: protected A
{
  x - protected
  y - protected
  z - не е достъпен
};
```

```
class D: (private) A
{
  x - private
  y - private
  z - не е достъпен
};
```

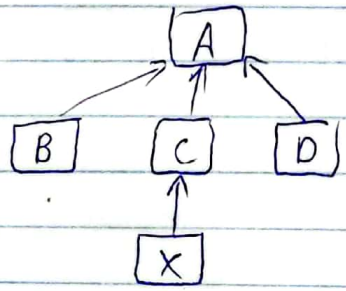
Нека разгледаме следния пример:

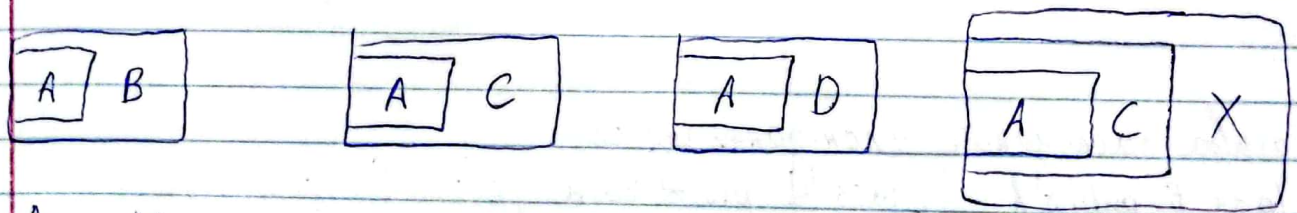
class Dog ⟷ class Perso
↳ име ↳ име
↳ години ↳ години
↳ порода

Тук може да се изкушим да направим наследяване, тъй като Dog и Person имат общи член-данни, по подобие на Student и Person. Но в никакъв случай не бива да го правим, защото формално менд класовете може да се слуги но като дизайн е грешно. Наследяването между класовете се прави, когато между тях има някакво взаимоотношение

Dog is a Person ✗ (куче не е човек)
Student is a Person ✓ (ученикът е човек)

Наследяването е транзитивна релация:





Ако X е наследник на C и C е наследник на A, то X е също наследник на A.

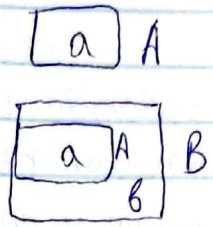
$XRC \;\&\; CRA \Rightarrow XRA$

Подаване като параметри във функции

Класовете-наследници могат да бъдат подавани като параметри на функции, които приемат обекти от базовия клас. Може да се използва функционалността от базовия клас.

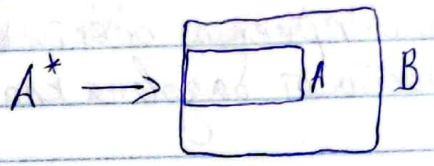
Да разгледаме следния пример:

```
class A
{
 public:
  int a;
};
class B: public A
{
  pubic:
    int b;
};
void f (A& obj)
{
  obj.a++;
}
void g (A* ptr)
{
 (*ptr).a++;
}
void t (B& obj)
{
 obj.b ++;
}
int main()
{
 A obj1; B obj2;
 f(obj1); // Валидно
 f(obj2); // Валидно
 g(&obj1); // Валидно
 g(&obj2); // Валидно
 t(obj2); // Валидно
 t(obj1); // Невалидно - защото обект от тип A няма член-данните на B
}
```



Нека разгледаме следната ситуация:

```
A obj1;
B obj2;
A* ptr1 = &obj1; // Валидно
A* ptr2 = &obj2; // Валидно
B* ptr3 = &obj1; // Невалидно
B* ptr4 = &obj2; // Валидно
```



Можем да насочваме указателите към обекти от ~~типове~~ класовете наследници. A* ptr може да насочим към обект от същия тип (тривиално), както и към обект от тип B, защото в наследника на B имаме обект от тип A, B е негов наследник. B* ptr може да насочим към обект от същия тип, но не и към обект от тип A, защото A не е наследник на B - не съдържа обект от тип B в себе си.

Конструктори и деструктори при наследяване

Във всеки ~~наследник~~ конструктор на класа-наследник трябва да се указва кой конструктор да се извика за базовия клас. Ако не е указано, че се извика default-ния конструктор. Ако базовият клас няма конструктор по подразбиране, то не можем да създадем обект на класа-наследник без да укажем конструктор, който да се извика за базовия клас.

Пример:

```
class B: public A
{
  ....
public:
  B(....) : A(....) // Конструктор на A
  {
    // Грижим се само за данните на B
  }
};
```

Деструктурът на наследения клас извиква деструктура на базовия клас

```
~B()
{
 //грижим се само за B
} -> извиква деструктура на A
```

Копиране при наследяване
При разписване на конструктура за копиране и оператор= на клас-наследник, трябва експлицитно да извикаме копи констр. и оп= и за базовия клас.

```
B(const B& other) : A(other) // к.к. на A
{
  copyFrom(other); //грижи се само за данните на B
}

B& operator=(const B& other)
{
  if (this != &other)
  {
    A::operator=(other); //оператор= на A
    free();              } //грижи се само за данните на B
    copyFrom(other);     }
  }
 return *this;
}
```

Разписване копирането и прието само ако наследникът ползва неозработена динамична памет. Ако не използва, генерираните от компилатора работят коректно.