С сайта www.malshakov.ru

Московский авиационный институт

(технический исследовательский университет)

Кафедра 403

“Электронно-вычислительные средства и информатика”

Отношения между классами

Корабельников Д.И.

(руководитель: Мальшаков Г.В.)

Москва 2013 г.

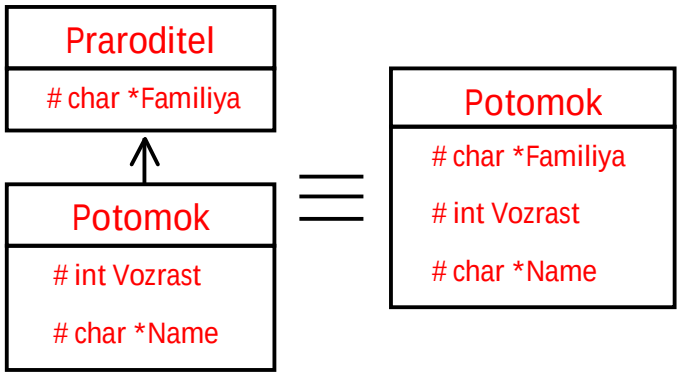
Наследование – отношение, при котором один класс строится на основе другого, уже существующего класса.

Используя наследование, мы создаем ***иерархию классов***, необходимую нам для упорядочивания классов, которое достигается вынесением общих признаков в общий базовый класс.

Производный класс от базового в наследство принимает поля и методы.

```
class Praroditel
{
  protected:
      char *Familiya;
};

class Potomok: public Praroditel
{
  protected:
      int Vozrast;
      char *Name;
};
```



класс Potomok наследует поле *Family* класса Praroditel

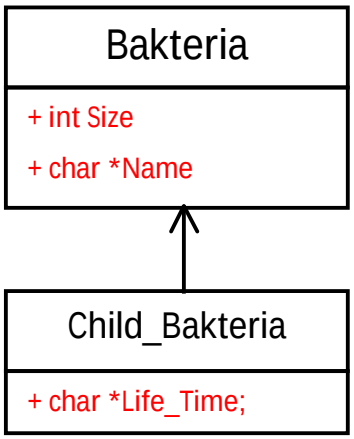
Существуют два вида наследования: простое и множественное.

1. При простом наследовании у класса-ребенка имеется всего лишь один класс-родитель.

Пример:

```
// Бактерия
class Bakteria
{
    public:
       int Size;   \\ размер
       char *Name; \\ имя
};


// Бактерия-Ребенок (отделившаяся)
class Child_Bakteria: public Bakteria
{
    public:
       char *Life_Time; \\ время жизни
};
```



класс Child_Bakteria наследует поля *Size* и *Name* от класса Bakteria

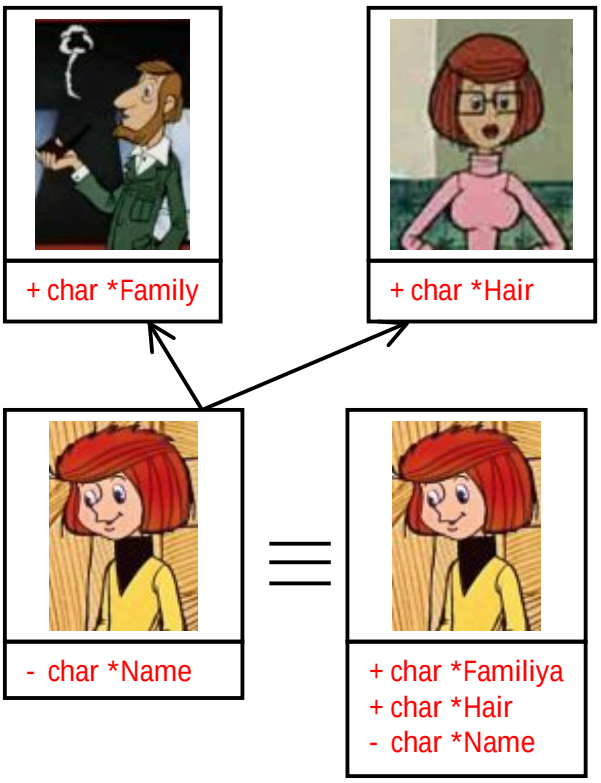
2. При множественном наследовании у класса-ребенка несколько классов-родителей.

Пример:

```
// Папа
class Papa
{
    public:
       char *Family; // фамилия
};


// Мама
class Mama
{
    public:
       char *Hair; // вид и цвет волос
};


// Ребенок наследует от классов Папа и Мама
class Rebenok: public Papa, public Mama
{
       char *Name; // имя
};
```



класс Rebenok наследует поля *Family* от класса Papa и *Hair* от класса Mama

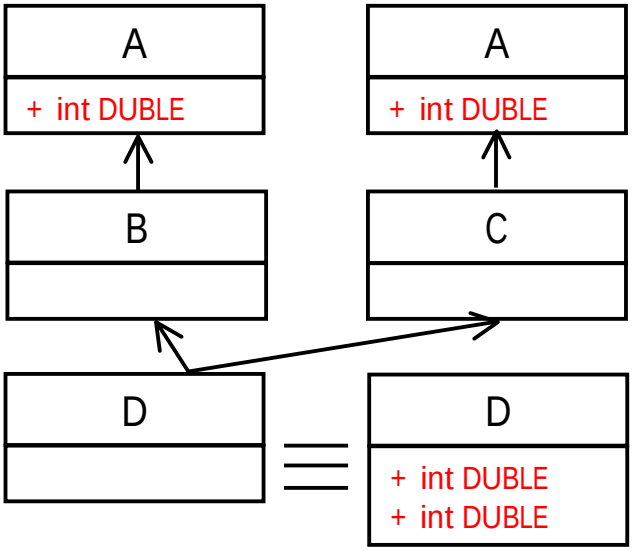
При множественном наследовании может произойти ситуация *дублирования наследования:*

```
class A
{ public: int DUBLE; };

class B: public A
{};

class C: public A
{};

class D: public B, public C
{};
```



класс D наследует два раза поле *DUBLE* класса *A* через производные классы B и C

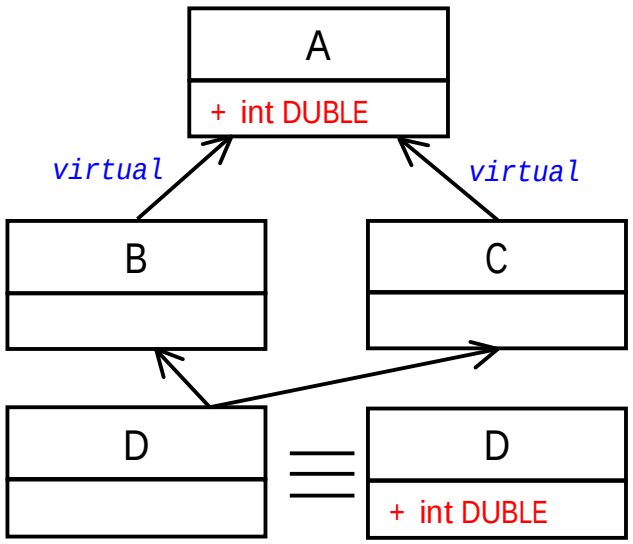
Чтобы этого избежать, используется **виртуальное наследование**:

```
class A
{ public: int DUBLE; };

class B: virtual public A
{};

class C: virtual public A
{};

class D: public B, public C
{};
```



класс D наследует один раз класс *A*, поэтому он будет содержать одну копию поля *DUBLE*

Задания:

1. При транспортировке произошло искажение программы – пропала информация по некоторым классам и наследовании. Восстановите программу и затем нарисуйте диаграмму UML.

```
// животные
class Animal
{
  public:
       char *Name;
       char *Vid; // подразумевается рыба, млекопитающее, птица, др.
};

// птицы
class Bird: public Animal
{
  public:
       int *Razmax krilyev;
};

// рыбы
class Fish:
{
  public:
       int *Dlina_tulovisha;
};

// акула
class Shark: public Fish
{
  public:
    int Kolichestvo_Zubov;
};
// волк
class Wolf:
{
  public:
    char *Color; // цвет
};


// альбатрос
class Albatross:
{
  public:
    char *Mouth; // рот
};
```

2. Объедините представленных ниже животных в единую иерархию наследования (От общего класса «Волчьи» унаследовать «Волк», «Шакал», «Собака»). Нарисуйте диаграмму UML и напишите программный код.

Собака

Шакал

Волк

3. Придумайте свои примеры наследования в виде картинок, диаграмм UML и программного кода.

Наследование бывает трех видов: **закрытое** (*private*), **защищенное** (*protected*), **открытое** (*public*).

От вида наследования зависит видимость элементов класса-родителя в классе-наследнике.

```
class ИМЯ : [вид_наследования] ИМЯ_КЛАССА-РОДИТЕЛЯ
{
      // тело класса;
};
```

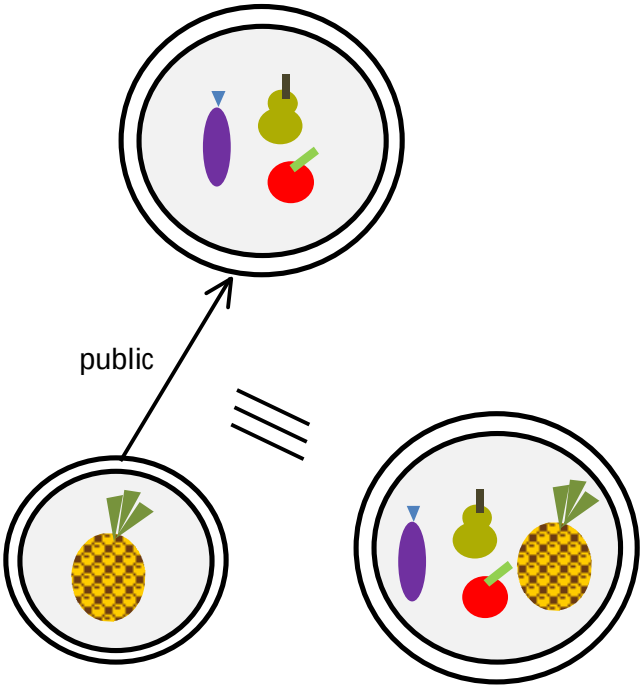
По умолчанию наследование является закрытым (*private*).

Таблица 1. Изменение видимости элементов при различных видах наследования.

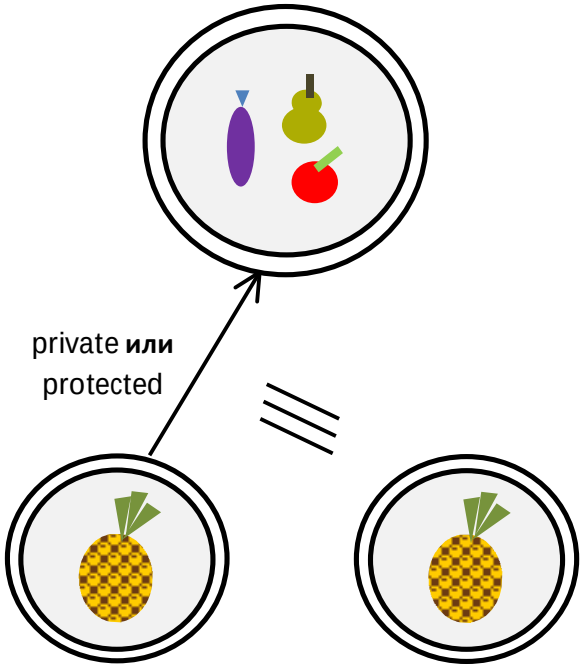
Объявление элементов в ***базовом*** классе	Вид наследования	Доступ к элементам базового класса через производный класс
private	private	private
protected		
public		
private	protected	private
protected		protected
public		
private	public	private
protected		protected
public		public

Правило: *private* - самая сильная защита (доступ закрыт всем), поэтому перебивает *protected* и *public*; *protected* - средняя защита (защищает на уровне данной иерархии, т.е. доступ открыт только наследникам), перебивает только *public*; *public* является самой слабой защитой (доступ открыт всем).

При открытом наследовании



При закрытом наследовании




```
class Big_Tarelka
{
  protected:
   int GrushaSize; //размер груши
   int SlivaLength; //длина сливы
  public:
   char *YablokoColor; //цвет яблока
};

class Small_Tarelka: public Big_Tarelka
{
  public:
   int AnanasWeith; //вес ананаса
};

int main ()
{
   Small_Tarelka Tarelka;

   Tarelka.AnanasWeith=54;

   Tarelka.YablokoColor=”red”;
}
```

```
class Big_Tarelka
{
  protected:
   int GrushaSize; //размер груши
   int SlivaLength; //длина сливы
  public:
   char *YablokoColor; //цвет яблока
};

class Small_Tarelka: private Big_Tarelka
{
  public:
   int AnanasWeith; //вес ананаса
};

int main ()
{
   Small_Tarelka Tarelka;

   Tarelka.AnanasWeith=54;

   Tarelka.YablokoColor=”red”; //ошибка
}
```

Задания:

1. В комментариях для каждого производного класса программного кода напишите вид наследования (простое либо множественное) и список его открытых полей.

```
class Papa
{
  private:
       int Kolichestvo_Eyes;
  protected:
       char *HairColor;
  public:
       int Kolichestvo_Nog;
};

class Mama
{
  public:
       int Kolichestvo_Ruk;
};

//Пример:
class Ivan: private Mama // наследование: простое, открытые поля: *Name
{
  public:
       char *Name;
};

class Petr: Papa, Mama // наследование:            , открытые поля:
{ };

class Roman: protected Mama // наследование:            , открытые поля:
{ };

class Yar: protected Papa // наследование:            , открытые поля:
{ };

class Daniel: public Mama // наследование:            , открытые поля:
{ };

class Alex: public Papa, protected Mama // наследование:            , открытые поля:
{ };
```

2. В соответствии с картинкой определить какие поля (из нижеперечисленных) передаются от родителя потомку. Напишите в соответствии со сделанными выводами класс потомка и наследника.

ąваемые поля: `char *Name, int Kol_vo_Lap, char *Griva_Color, char *Griva_Length, int Razmer_Tela, char *Yazik_Color, char *Virazenie_Lica, int Ushki_Razmer, char *Eye_Color, char *Living_Place,`

Виртуальная функция (метод) – это функция, объявленная в классе-родителе с ключевым словом *virtual* и переопределенная в классах-детях.

```
class A
{
   virtual void Show() { };
};
```

Пример:

Обычный метод Show()	**Виртуальный метод** Show()

`class Praroditel`
`{`
`  public:`
`   void Show()`
`   {`
`        cout << " Метод родителя ";`
`   }`
`};`
 	
`class Potomok: public Praroditel`
`{`
`   public:`
`   void Show()`
`   {`
`        cout << "Метод потомка";`
`   };`
 	
`   void Run() { Show(); };`
`};`
	`class Praroditel`
	`{`
	`  public:`
	`   virtual void Show()`
	`   {`
	`        cout << "Метод родителя";`
	`   }`
	`};`
	
	`class Potomok: public Praroditel`
	`{`
	`   public:`
	`   void Show()`
	`   {`
	`        cout << "Метод потомка";`
	`   };`
	
	`   void Run() { Show(); };`
	`};`

`int main()`
`{`
`   Potomok Petr;`
`   Petr.Run();`
`   return 0;`
`}`	

Вывод на экран: Метод родителя	*Вывод на экран:* Метод потомка

Комментарий: в отличие от обычного метода, виртуальный метод переопределяется в самом базовом классе. После этого методы базового класса будут использовать подмененный метод производного класса. Поэтому во втором случае метод Run() запустит метод Show() потомка и на экране выведется *“Метод потомка”.*

Вывозов *обычного метода* происходит на основе **раннего или статического связывания** (реализуется на этапе компиляции), а вызов *виртуального метода* на основе **позднего (отложенного) связывания** (реализуется на этапе выполнения программы с помощью таблицы виртуальных методов).

Правила описания и использования виртуальных методов:

1. Виртуальными могут быть только нестатические методы.
2. Виртуальные функции могут быть дружественными к другим классам.
3. Виртуальность наследуется.
4. Механизм виртуального вызова перебивается явным использованием полного имени (то есть объекты класса-ребенка могут вызывать методы класса-родителя с помощью оператора видимости **::**).
5. Если в классе вводится описание виртуального метода, то он должен быть определен хотя бы как **чисто виртуальный**:

Чисто виртуальный метод – содержит признак =0 вместо тела, например:

```
virtual тип имя_функции (список_формальных_параметров) = 0;
```

Чисто виртуальная функция ничего не делает и недоступна для вызовов. Её назначение – служить основой для подменяющих её функций в производных классах.

Абстрактный класс – это класс, в котором определена хотя бы одна чисто виртуальная функция. Он может использоваться только в качестве базового для производных классов. Механизм абстрактных классов **разработан для общих понятий, которые в дальнейшем предполагается конкретизировать.**

Абстрактный класс имеет следующие отличия от обычного:

1. Невозможно создать объект абстрактного класса.
2. Если класс-ребенок от абстрактного класса-родителя не определяет все чисто виртуальные функции, то он также является абстрактным.

Задания:

1. Вам, как инженеру космического корабля, было дано задание: «Описать путешествие программой». Необходимо указать общий класс «Путешествие», в котором были бы классы «Корабль» (разнообразные характеристики корабля) и «Экипаж» (члены команды с различными профессиями).

2. В 1834 году Чарльз Дарвин посетил Галапагосские острова. Там он описал множество животных и в частности огромное число разновидностей дрозда. Каждый дрозд обладал неповторимой песней, разной окраской и клювами. После своего путешествия Дарвин, используя эти и другие собранные им данные, построил свою теорию эволюции и приспособления (книга “Происхождение видов”). Вам необходимо описать программой этих дроздов, выделяя их схожие и личные признаки.




Включение – это отношение между классами, при котором объект одного класса строится из объекта (-ов) другого класса.

Включая одни объекты в другие, мы создаем новую иерархию – объектов, которая является альтернативой и дополнением к иерархии классов.

Примеры включения:

Семья состоит из 4 человек

```
class FamilyMember // член семьи
{
  char* Name;
};

class Family
{
  FamilyMember Mem[4]; // члены семьи
  char* Surname;       // фамилия
};
```

Галактика состоит из 1000 звезд

```
class Star
{
  char* Name;
  int Temp; // температура звезды
};

class Galaxy
{
  Star Stars[1000]; // кол-во звезд
  char* GalaxyName; // имя галактики
};
```

Планетарная система состоит из 9 планет

```
class Planet
{
   char* Name;
   int* Life; // наличие жизни
};

class System
{
   Planet Planets[9]; // планеты
   char* System_Name; // имя системы
};
```

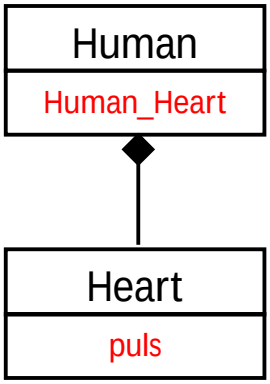
Существует два вида включения: ***композиционное*** **и** ***множественное***.

1. **При композиционном включении объект входит в другой объект как его неотделимая составная часть и время их жизни совпадают.**

Пример: класс Человек содержит поле с типом Сердце.

```
//сердце
class Heart
{
   int puls;
};

//человек
class Human
{
   Heart Human_Heart;
};
```



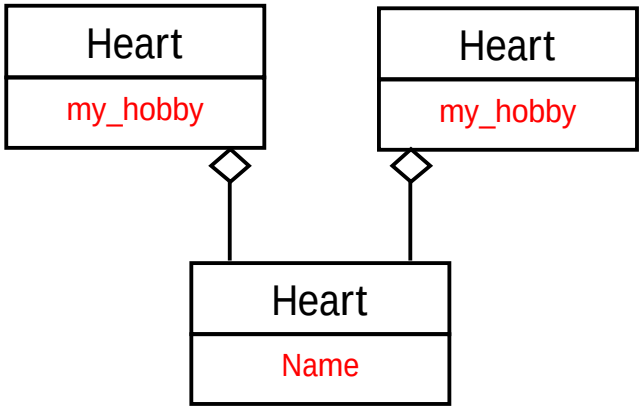
2. **При множественном включении объект может одновременно входить в несколько объектов (объекты хранят адрес вложенного объекта).**

Пример: классы Daniel **и** Alex **имеют одно и то же Хобби.**

```
class Hobby
{
   char Name[100];
};

class Daniel
{
   Hobby* my_hobby;
};

class Alex
{
   Hobby* my_hobby;
};
```



Задания:

1. Нарисуйте диаграмму UML и напишите программу в которой рабочий (имя, фамилия) работает на предприятии, в больнице и в офисе (т.е. работник входит в коллектив различных предприятий и учреждений).

2. Даны беспорядочные элементы некогда целостной диаграммы UML. Восстановите связи, нарисовав диаграмму UML.

3. В соответствии с рисунком нарисуйте диаграмму UML и напишите программу «Велосипед». Не забудьте в ней указать его основные характеристики (детали).

4. По изображениям мультгероев, создайте класс «Мультфильмы», включив туда этих героев и придав им произвольные характеристики, например «имя».





Словарь терминов.

Наследование – это получение нового класса на основе уже существующего.

Иерархия классов – это иерархия, в основу которой заложено порождение из класса-родителя класса детей.

Простое наследование – это наследование, при котором у классов детей не более одного класса-родителя.

Множественное наследование – это наследование, при котором классы-дети имеют более одного класса-родителя.

Наследование бывает трех видов: **закрытое** (*private*), **защищенное** (*protected*), **открытое** (*public*).

Виртуальное наследование – это наследование, объявленное с ключевым словом `virtual`, чтобы избежать дублирования кода при множественном наследовании.

Раннее или **статическое связывание** – это определение вызываемой функции на этапе компиляции.

Позднее (**отложенное**) или **динамическое связывание** – это определение вызываемой функции на этапе выполнения программы.

Виртуальная функция – это функция, объявленная в классе-родителе с ключевым словом `virtual` и переопределенная в классах-детях.

Чисто виртуальный метод – содержит признак =0 вместо тела

Абстрактный класс – это класс, в котором определена хотя бы одна чисто виртуальная функция (метод).

Включение – это такое отношение между классами, при котором объект одного класса строится из объекта (-ов) другого класса.

Иерархия объектов – это иерархия, построенная на основе включения одних объектов в другие.

Композиционное включение – это вхождение одного объекта в другой в качестве его неотделимой составной части части.

Множественное включение – это автономное вхождение объекта в один или несколько объектов по адресу.