Технология программирования

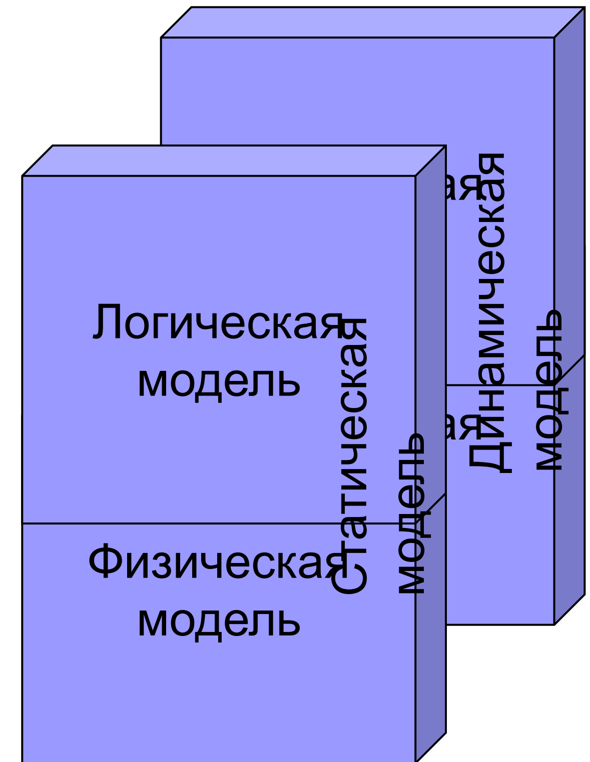
Основы объектно-ориентированного моделирования

Принципы объектно-ориентированного подхода

Объектно-ориентированный анализ – это *методология* системного **анализа**, направленная на создание моделей, близких к реальным явлениям. Требования к проектируемой системе формируются на основе понятий (классов и объектов), составляющих *словарь предметной области* (термины предметной области, необходимые для рассматриваемой задачи)

Принципы объектно-ориентированного подхода

Объектно-ориентированное проектирование – это *методология* **проектирования** на основе объектной *декомпозиции* и объектного *синтеза* логической модели, физической модели, статической модели и динамической модели проектируемой системы



Модели объектно-ориентированного проектирования

Логическая модель: структуры классов и структуры объектов

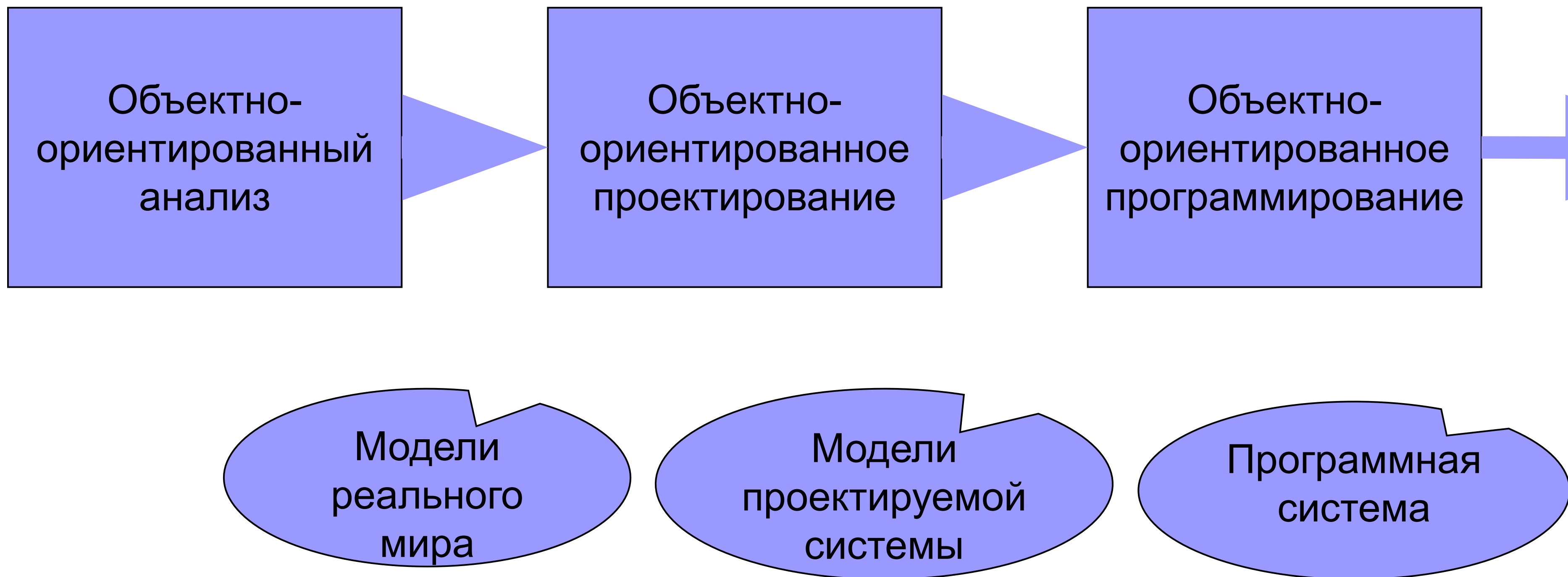
Физическая модель: архитектура модулей и архитектура процессов

Принципы объектно-ориентированного подхода

Объектно-ориентированное программирование – методология *программирования*, которая основана на представлении программы в виде совокупности **объектов**, каждый из которых является реализацией определенного **класса**, а классы образуют иерархию наследования

Принципы объектно-ориентированного подхода

Взаимосвязь анализа, проектирования и программирования



Основные понятия объектного моделирования

1. Абстрагирование

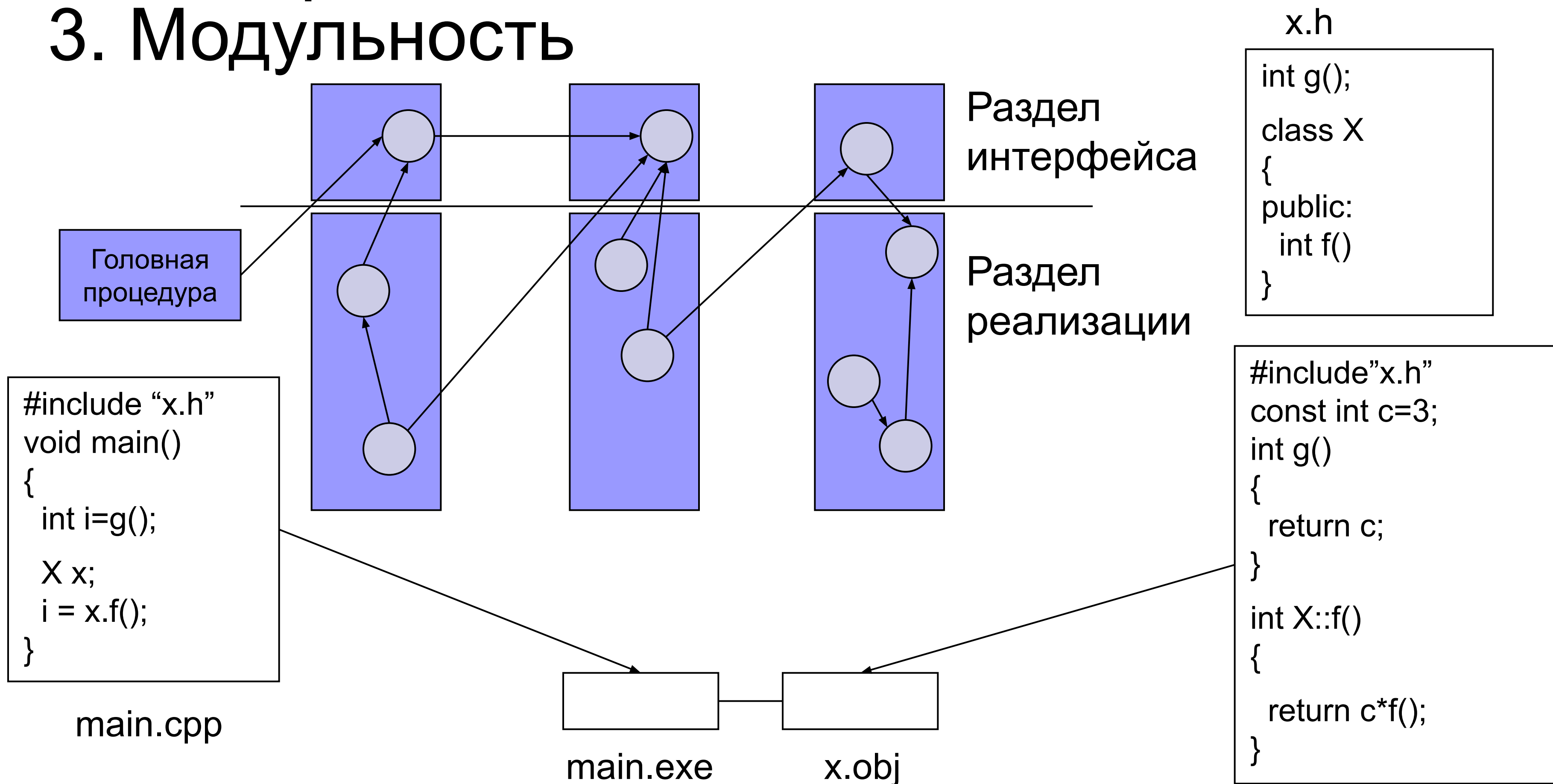
```
struct Point {int x, int y};
class Figure
{
private:
  Point _center;
public:
  Figure();
  void SetCenter(Point center);
  virtual void Draw();
  virtual void Hide();
  Point GetCenter();
}
```

Основные понятия объектного моделирования

2. Инкапсуляция (ограничение доступа)

```
Point point = {1,3};
Figure figure;
figure.SetCenter(point);
figure.Draw();
figure._center = point; // Ошибка, т.к. ограничение доступа
```

Основные понятия объектного моделирования
3. Модульность



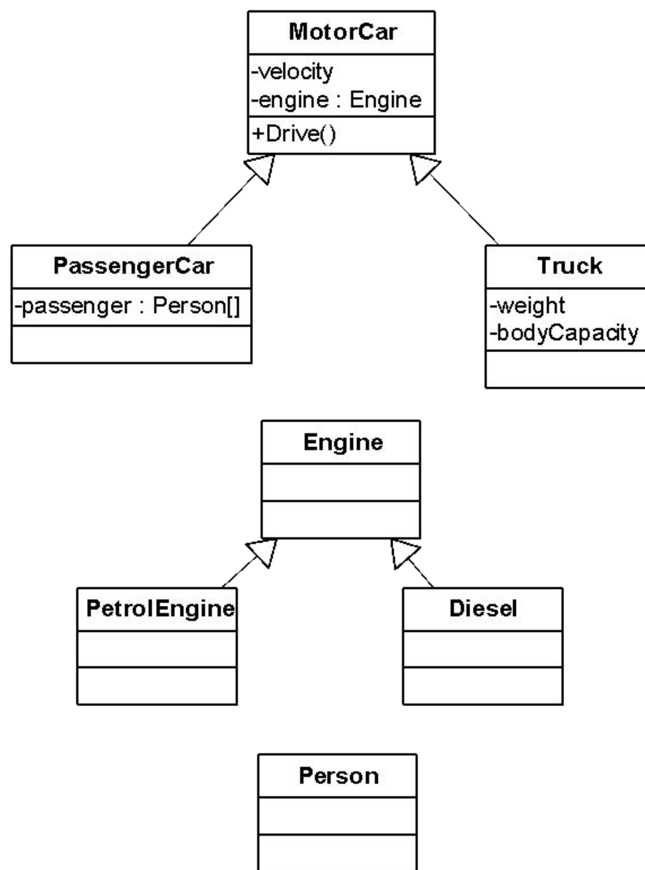
Физические модули: компонент, пакет (физическая группировка)
Логические модули: класс, подсистема (логическая группировка)

Характеристики: связность модуля, сцепление модулей

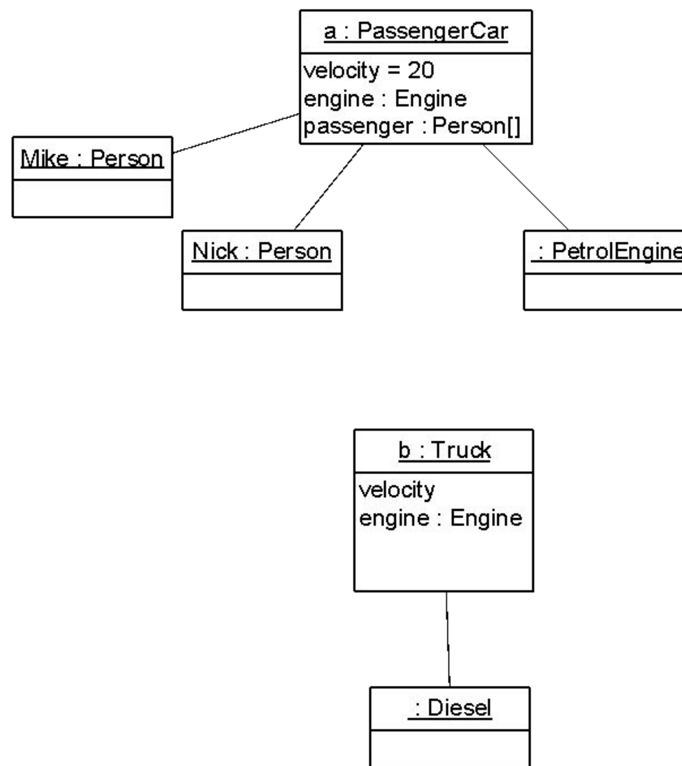
Основные понятия объектного моделирования

4. Иерархия

Иерархия классов



Иерархия объектов



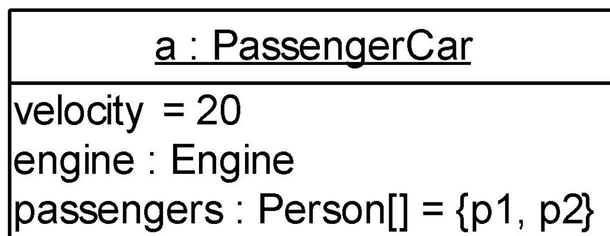
Основные понятия объектного моделирования

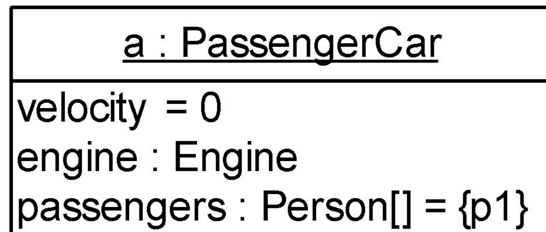
```
class Engine {float power;}
class PetrolEngine : public Engine {}
class DieselEngine : public Engine {}

class Person {}

class MotorCar
{
  double velocity;
  Engine engine;
public:
  void Drive() {}
}
class PassengerCar : public MotorCar
{
   Person passengers[];
}
class Truck : public MotorCar
{
  double weight;
  double bodyCapacity;
}
```

Объекты

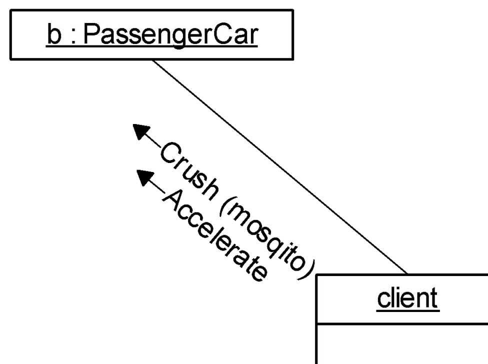
- Объект – это сущность, обладающая индивидуальностью, состоянием и поведением

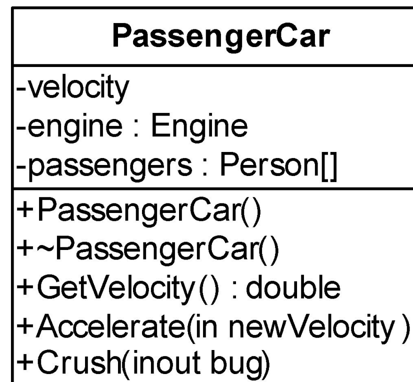
Изменение **состояния** объекта





Поведение объекта



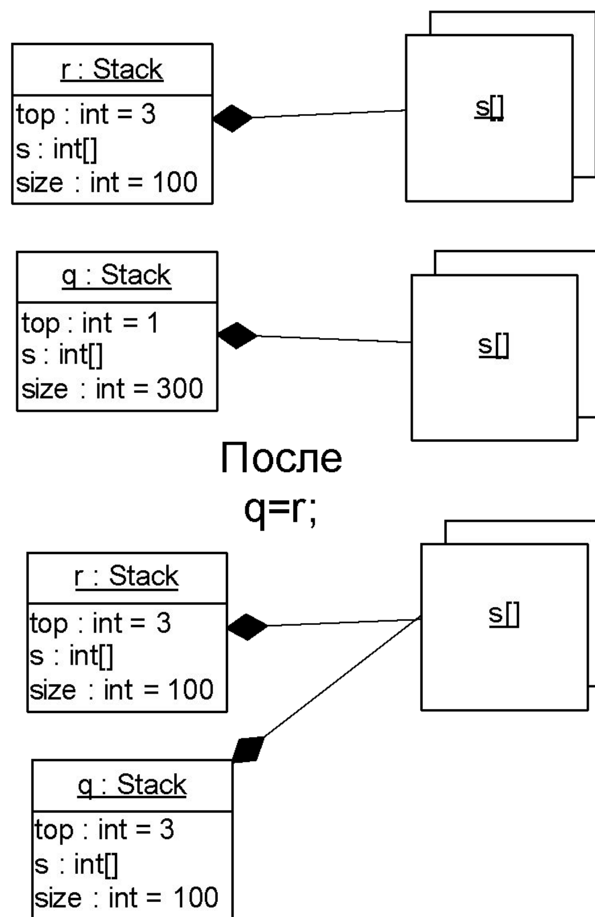



```
class Stack
{
private:
  int top;
  int *s;
  int size;
public:
  void Push(int i);
  int Pop();
  int IsEmpty();
  int IsFull();
  void Copy(Stack *other);
  Stack(int sz);
  ~Stack();
}
```

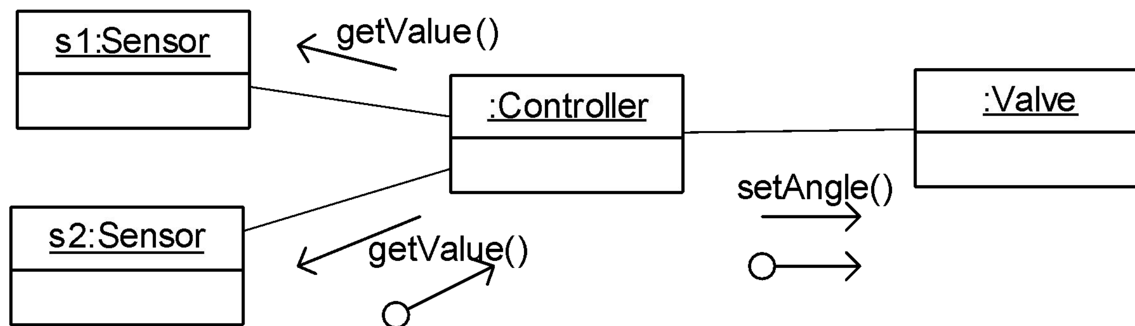
Объекты

Индивидуальность объекта

Stack r(100);
Stack q(300);
// …
q = r;

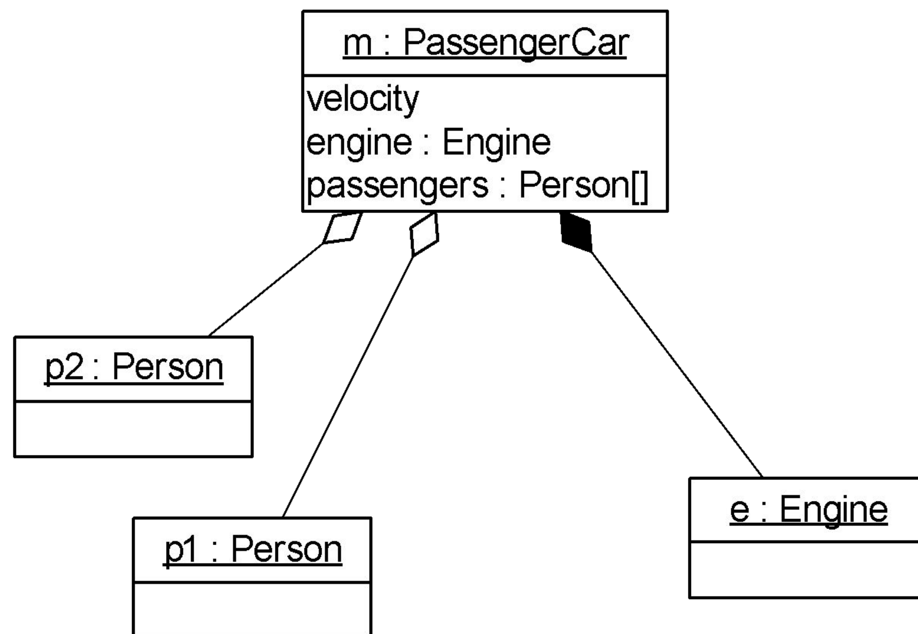


Объекты



Отношения:

- **Связь** – взаимодействие между экземплярами сущностей
- **Агрегация** (агрегация по ссылке, разделяемая агрегация) - отношение «часть-целое»
- **Композиция** (агрегация по значению) – строгая форма агрегации, агрегируемый объект принадлежит только одному агрегату. Связаны жизненные циклы.



Классы

- **Класс** – это описание структуры и поведения объектов, имеющих одинаковые свойства, поведение и семантику

MotorCar
-velocity -engine : Engine
+MotorCar() +~MotorCar() +GetVelocity() : double +Accelerate(in newVelocity) +Crush(inout bug)

AE3451
velocity engine : Engine passengers : Person[]

YS7688
velocity engine : Engine weight bodyCapacity

QU4324
velocity engine : Engine passengers : Person[]

Классы

Отношения (relationship) между классами:

- **Наследование (inheritance, generalization)** – отношение при котором один класс разделяет структуру и поведение другого класса
- **Ассоциация (association)** – описание связей между экземплярами классов
- **Реализация (implementation)** – отношение между интерфейсом и классом, его реализующим
- **Зависимость (dependency)** – отношение между классами, при котором изменения в одном классе приводят к изменениям в другом классе (наследование, ассоциация и реализация – **частные случаи отношения зависимости**, имеющие особое назначение и специальную нотацию)

При наследовании подкласс может :

- добавлять поля
- добавлять методы
- переопределять методы
- замещать методы
- уточнять методы

Наследование

```
class Figure
{
  int _x, _y;
public:
  virtual void Show() = 0;
  virtual void Hide() {/*…*/ };
void Move(int x, int y) {
    Hide();
    _x=x; _y=y;
    Show();
  }
};
class Circle: public Figure
{
// добавление поля
  int _radius;
public:
// замещение метода (реализация)
  virtual void Show() {/*…*/ };
};
```

```
class Face: public Circle
{
  int _eyeColor;
public:
// замещение метода
  void Show();
// переопределение метода
void Move(int x, int y) {/*…*/}
// добавление метода
  virtual void CloseEyes();
};
```

Наследование

```
int main()
{
  Circle *cPtr;
  cPtr=new Face; // фактический объект класса Face
  cPtr->Show(); // вызывается метод Face::Show()
  cPtr->Move(10,20); // вызывается Figure::Move(), так как невиртуальное
                     // замещение, а в нем методы Face::show() и
                     // Face::hide() в соответствии с фактическим
                     // классом объекта *cPtr.
//  cPtr->closeEyes(); // ошибка компиляции
  delete cPtr;

// ...

}
```

Наследование

Уточнение метода

```
class Circle: public Figure
{
virtual void Show()
  {/*рисование окружности*/ };
};
class Face : public Circle
{
virtual void Show()
  {
    /* рисование глаз */
    Circle::Show();
   /* рисование рта и ушей */
  };
};
```

```
int main()
{
  Face face;
  face.Show(); // вызывается также
               // Circle::Show()
}
```

В C++ уточнение реализовано для конструкторов и деструкторов

Наследование

Принцип подстановки: экземпляр класса, порожденного от некоторого класса X, может использоваться без нарушения семантики объявления X везде, где используется экземпляр класса X.

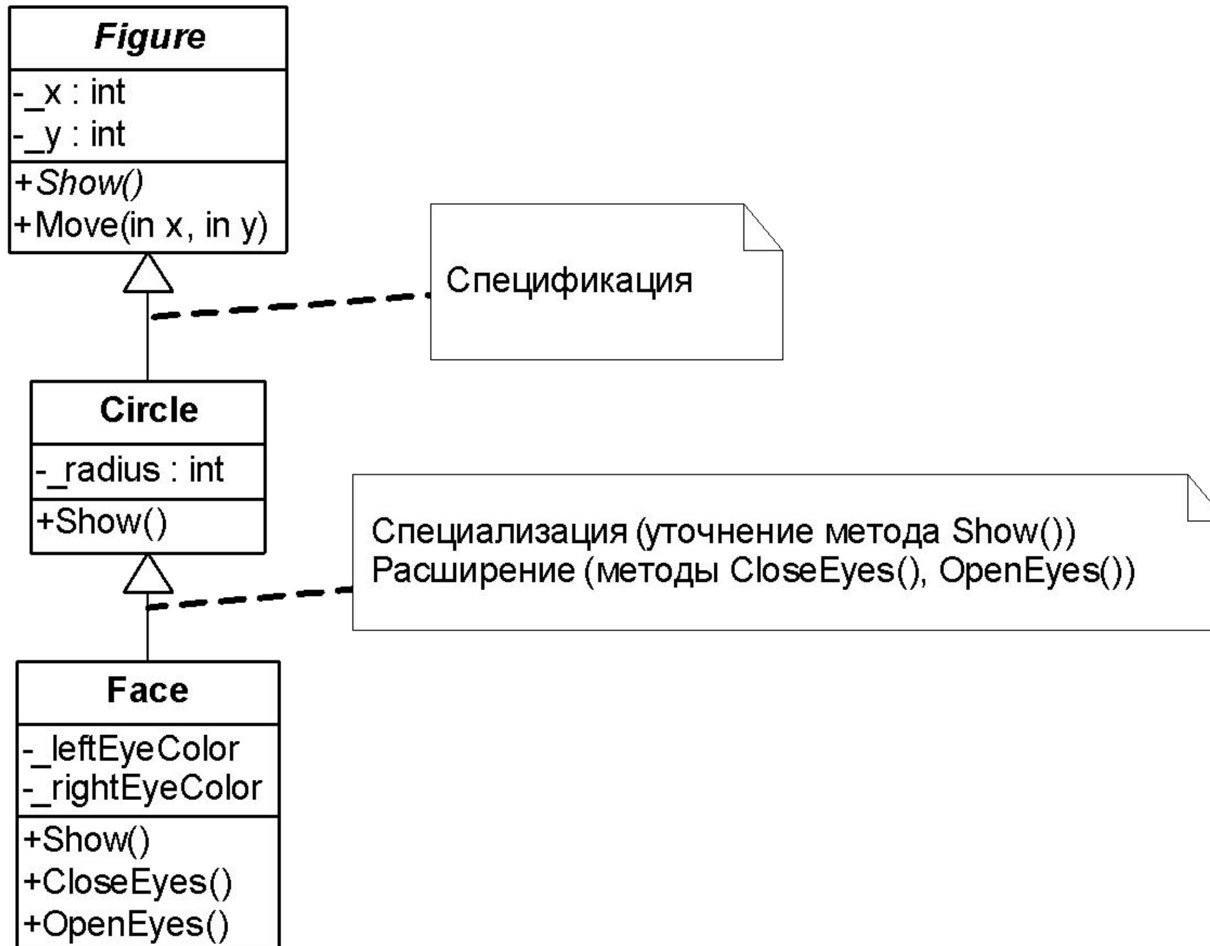
Различие понятий класс и тип: класс описывает структуру объектов, тип описывает протокол объектов. Класс может соответствовать нескольким типам (реализовать различные интерфейсы). Различные классы могут иметь один и тот же тип (реализовать один и тот же интерфейс).

Наследование

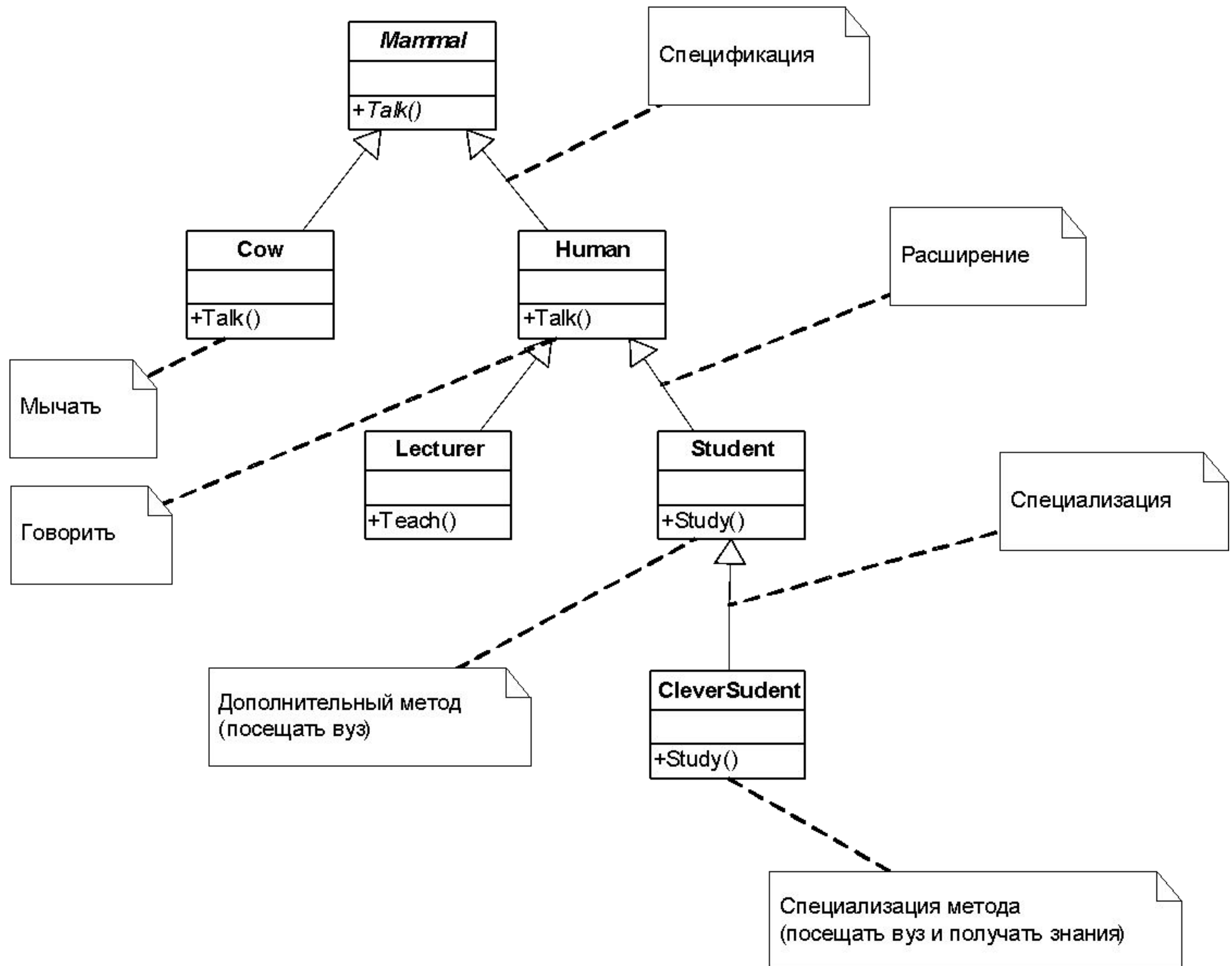
Формы наследования:

1) Порождение класса для **спецификации**. Родительский класс – абстрактный класс, т.е. содержит абстрактные методы. Подкласс должен реализовать поведение (т.е. обеспечить реализацию для абстрактных методов).
2) Порождение подкласса для **специализации**. Подкласс имеет такой же протокол, как и родительский класс, но специальным образом реализует некоторые методы.
3) Порождение класса для **расширения**. Подкласс добавляет к протоколу базового класса дополнительные методы.
4) Порождение класса для **варьирования**. Дочерний и родительский классы имеют одинаковые протоколы и могут занимать место в иерархии классов в произвольном порядке.
5) Порождение для **конструирования**. Подкласс использует структуру и поведение базового класса в своих методах, но при этом подкласс не является подтипом базового класса. Нарушается принцип подстановки.
6) Порождение классов для **комбинирования**. В классе комбинируются свойства двух и более классов (множественное наследование).
7) Порождение класса для **обобщения**. Подкласс является более общим, чем базовый класс. Целесообразно использовать только в особых случаях.
8) Порождение класса для **ограничения**. Подкласс ограничивает использование некоторых методов родительского класса. Нарушается принцип подстановки. Целесообразно использовать только в особых случаях.

Наследование

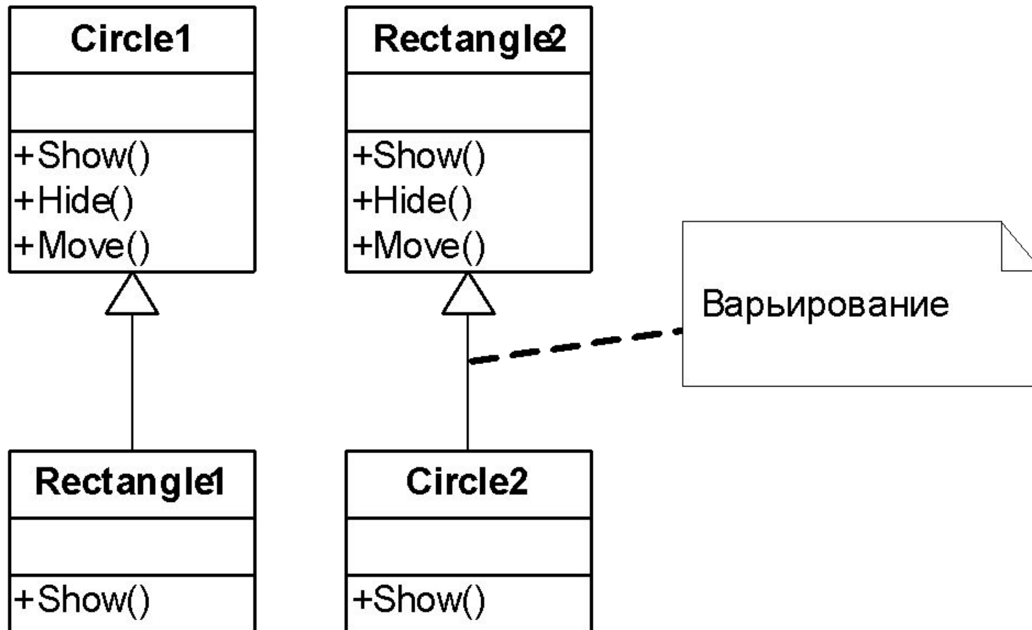


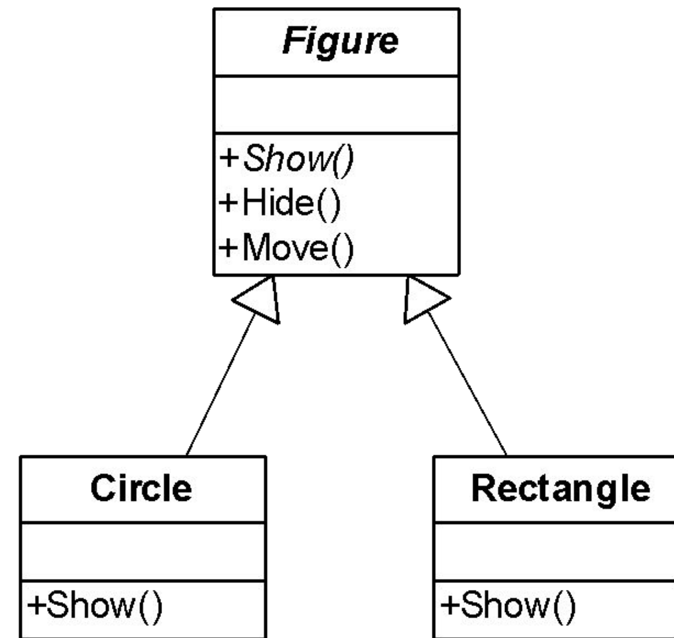
Наследование



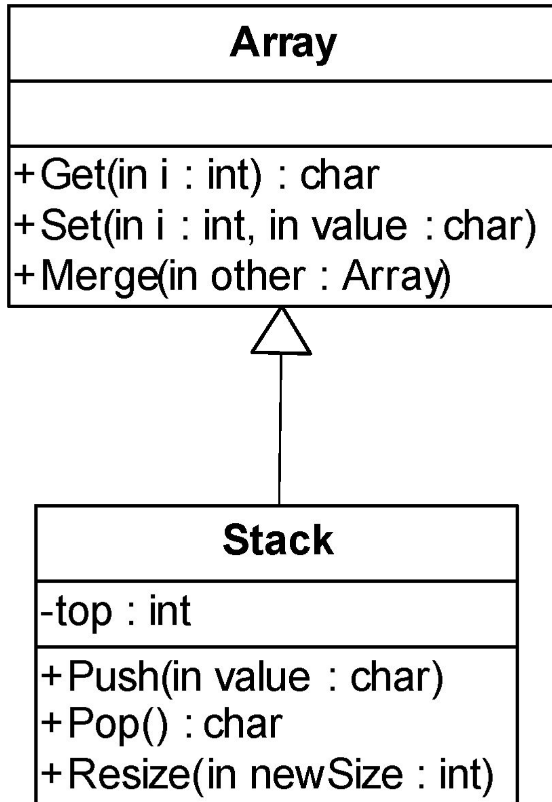
Наследование

Лучше так:

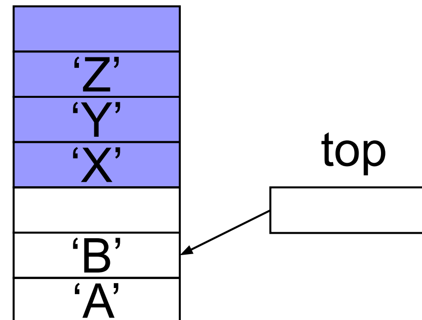




Конструирование



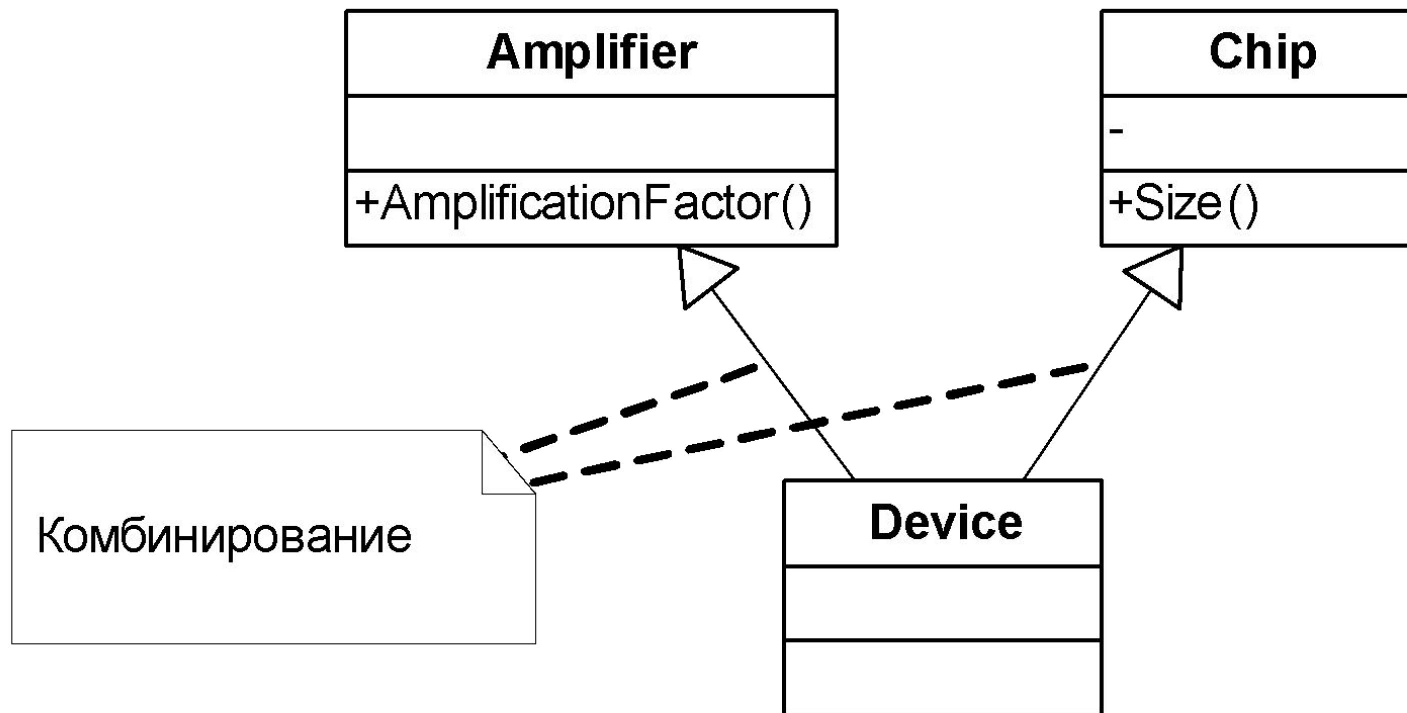
Stack s1, s2;
s1.Merge(s2); // Нельзя, для стека не имеет смысла,
// но в реализации Stack использует Array::Merge()



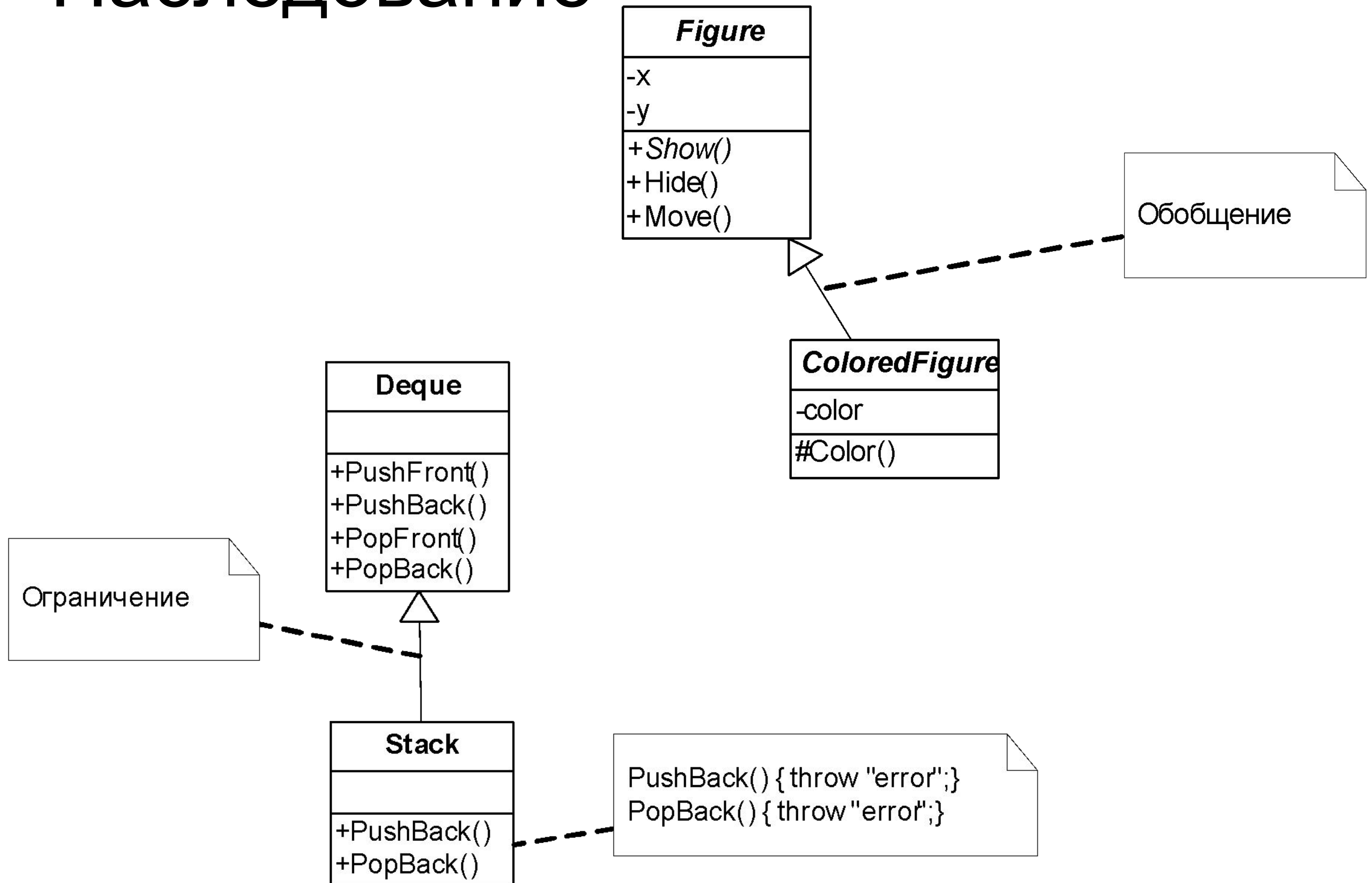
В C++ закрытое наследование

```
class Stack : private Array
{
public:
  void Push(char value) {  /* Set(top) */ }
  int Pop() { /* Get(top) */}
  void Resize(int newSize) { /* Merge(newArray) */ }
}
```

Наследование



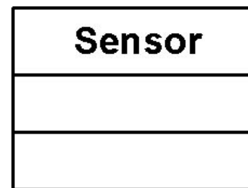
Наследование

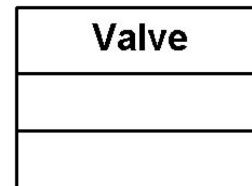


Ассоциация

```
class Controller
{
private:
  Sensor* _sensor[];
}

class Sensor
{
// нет ссылки на Controller
}
```

Конец ассоциации:
- Прослеживаемость (navigability)
- Множественность (multiplicity)
- Имя роли в ассоциации (rolename)
- Ограничение видимости (visibility)





-измеритель

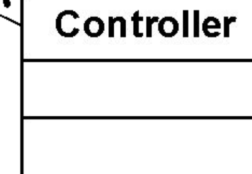
1..*
Управляет

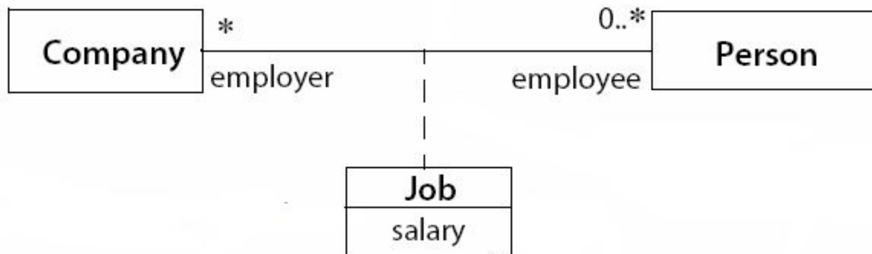
*

1

-вычислитель

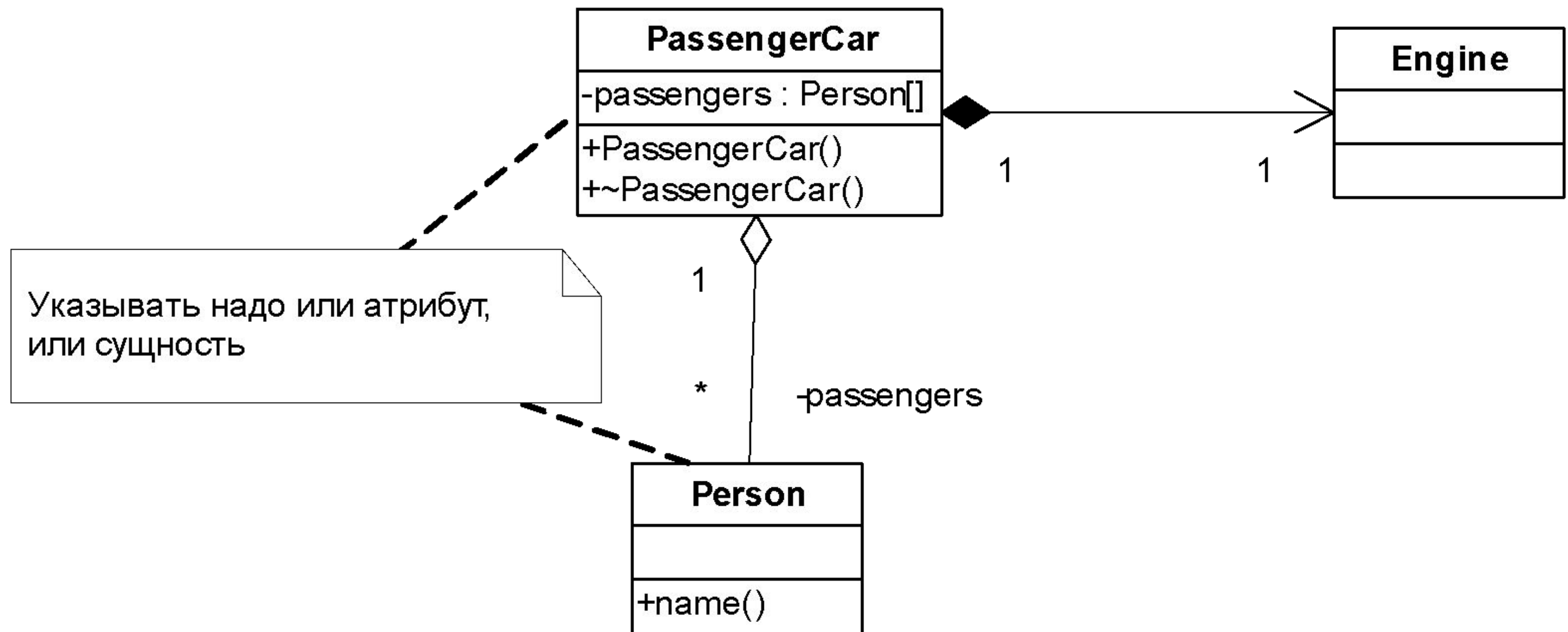
0..1



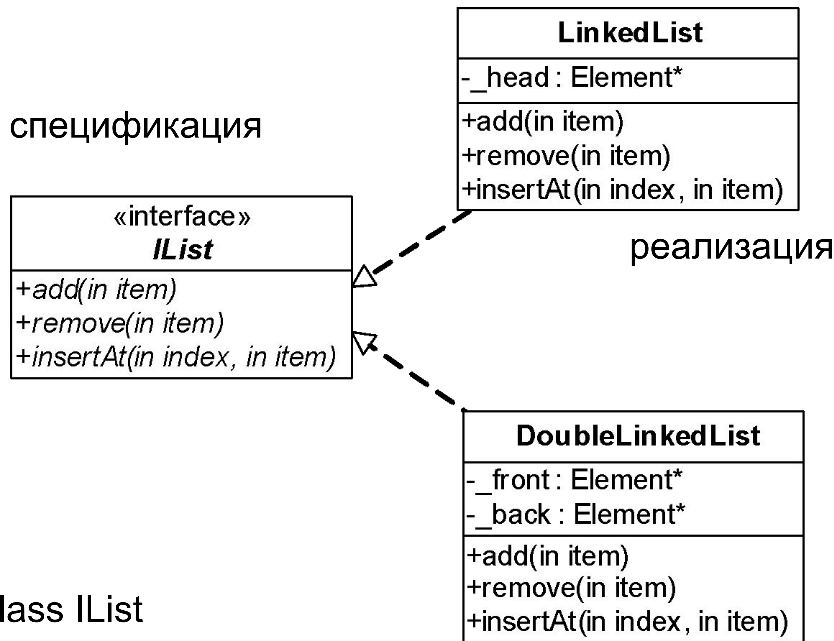


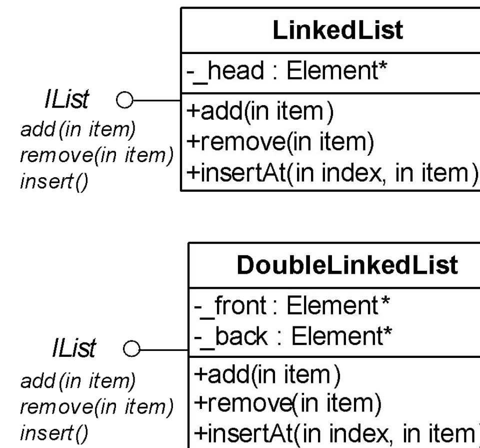

```
class Job
{
  public:
   Company* company;
   Person* person;
   double salary;
}
```

Агрегация



Реализация (realization/implementation)






```
class IList
{
public:
  virtual void add(string& item) = 0;
  virtual void remove(string& item) = 0;
  virtual void insertAt(int index, string& item) = 0;
}
```

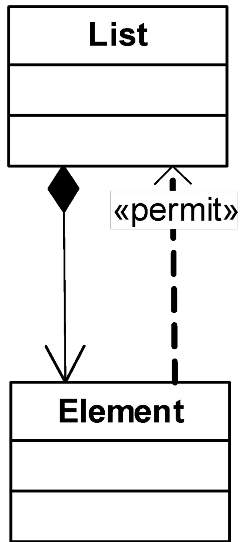
```
class LinkedList : public IList
{
private:
   Element* _head;
public:
  void add(string& item)
     {…  Element * p = new Element(item) …  }
  void remove(string& item)  {… delete p; …}
  void insertAt(int index, string& item) { … }
}
```

Зависимость

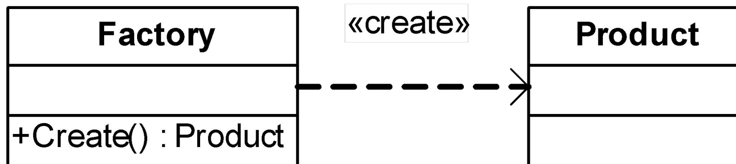
Стереотипы отношения зависимости:

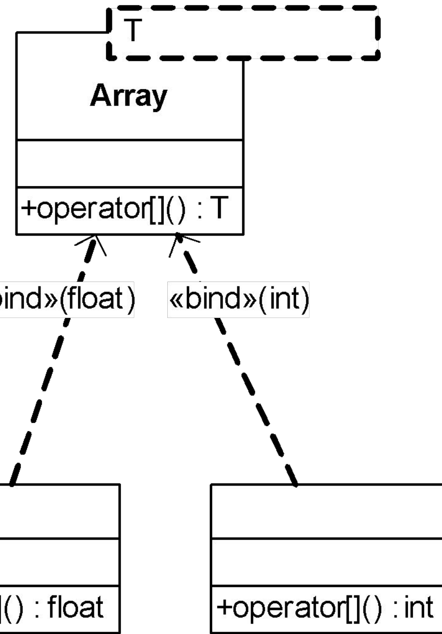
- <<bind>> – назначение параметров шаблонному классу для получения нового конкретного класса
- <<call>> – метод одного класса вызывает операцию другого класса
- <<create>> – один класс создает экземпляр другого класса
- <<permit>> или <<friend>> – разрешение одному классу использовать реализацию другого класса
- <<use>> - общее обозначение

Зависимость




```
class List;

class Element
{
  friend class List;
}

class List
{
   Element* _head;
}
```



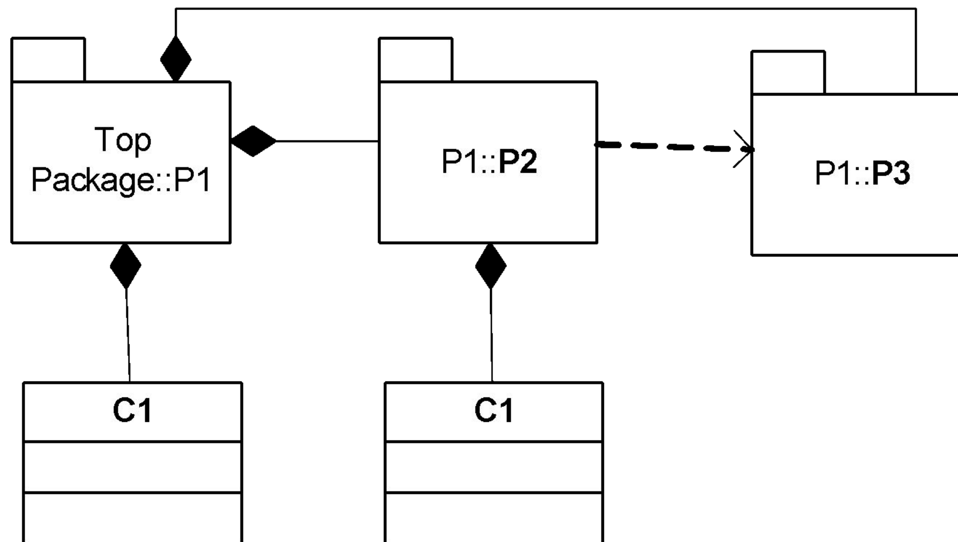



```
template<class T>
class Array
{
public:
  T operator[](int i) {…}
}

void main()
{
  Array<int> intArray;
  Array<float> floatArray;
// …
}
```

```
class Product
{
public:
  Product() {}
}

class Factory
{
public:
  Product Create()
  {
     return Product();
  }
}
```

Пакеты

Пакет – механизм общего назначения для распределения программных элементов по группам с установлением владельца, а также средства для предотвращения конфликтов имен




```
namespace P1
{
  class C1 {}

  namespace P2
  {
    class C1 {}
  }
}

int main()
{
  P1::C1   x1;
  P1::P2::C1 x2;
}
```

Диаграммы UML

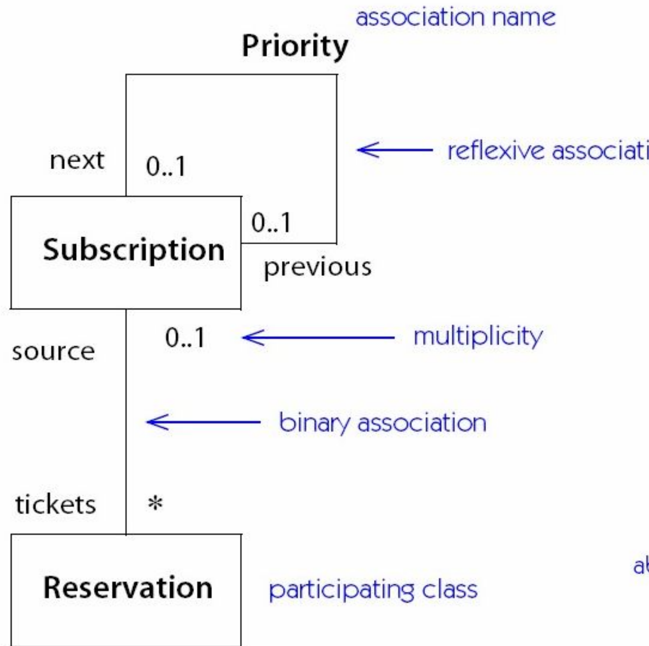
Представление (View) – это подмножество конструкций UML, отражающих один аспект системы.

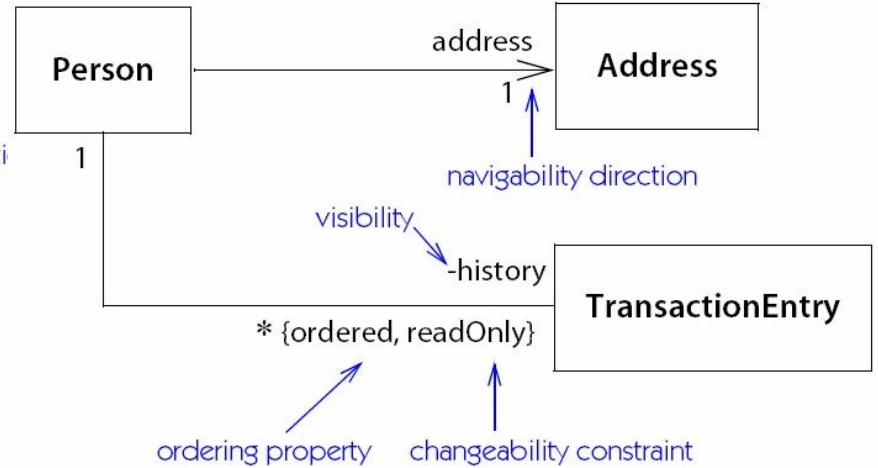
- Описание статической структуры (Static View)
- Описание вариантов использования (Use Case View)
- Описание дискретных автоматов (State Machine View)
- Описание активности (Activity View)
- Описание взаимодействия (Interaction View)
- Описание размещения (Deployment View)
- Описание проектных решений (Design View)

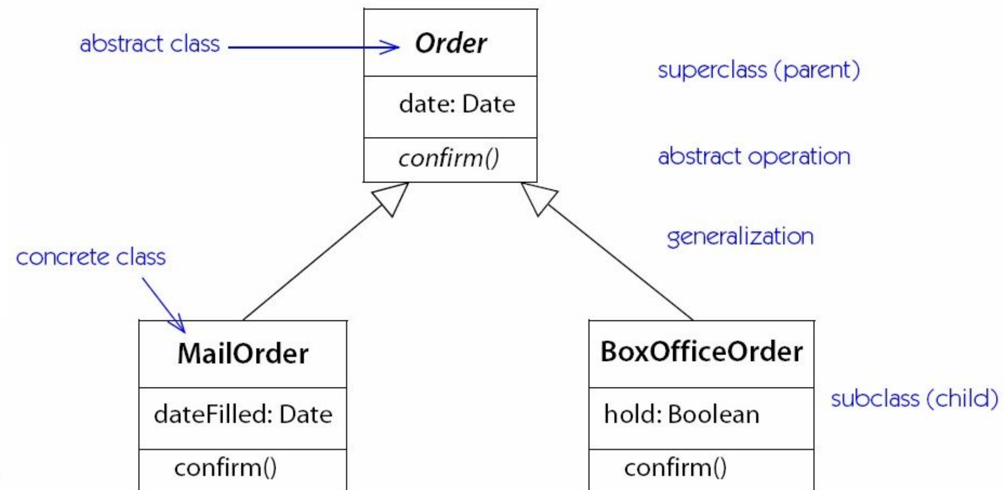
Рисунки из
Rumbaugh J., Jacobson I., Booch G. *The Unified Modeling Language Reference Manual*. – 2nd ed. Addison-Wesley. 2005

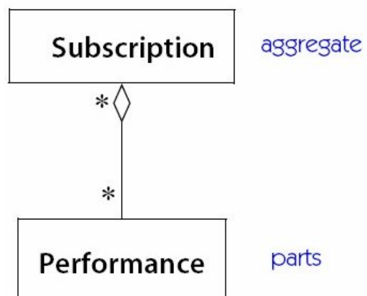
Описание статической структуры

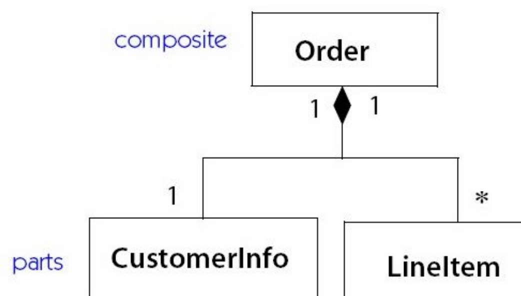
Диаграммы классов











Описание статической структуры

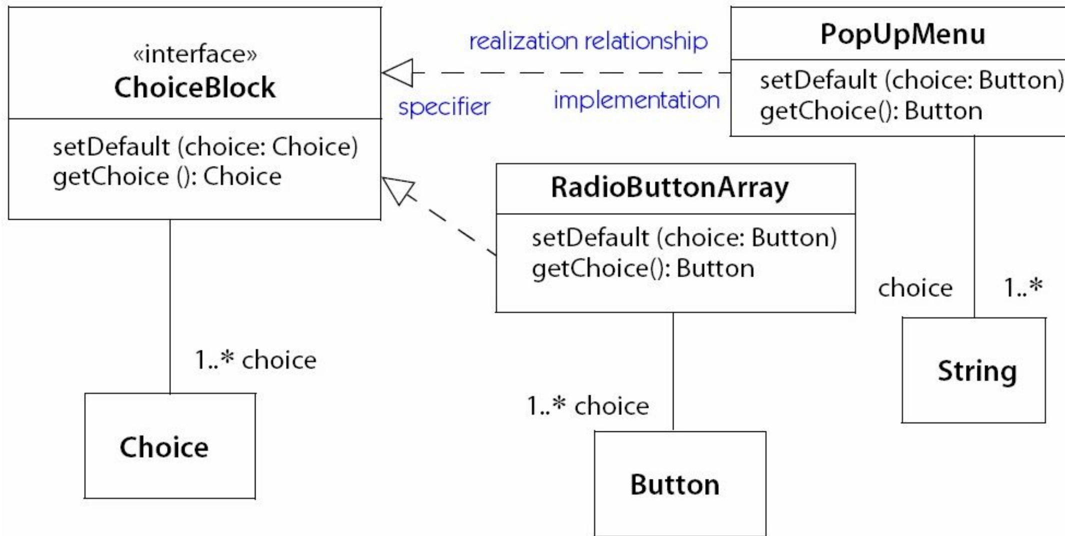
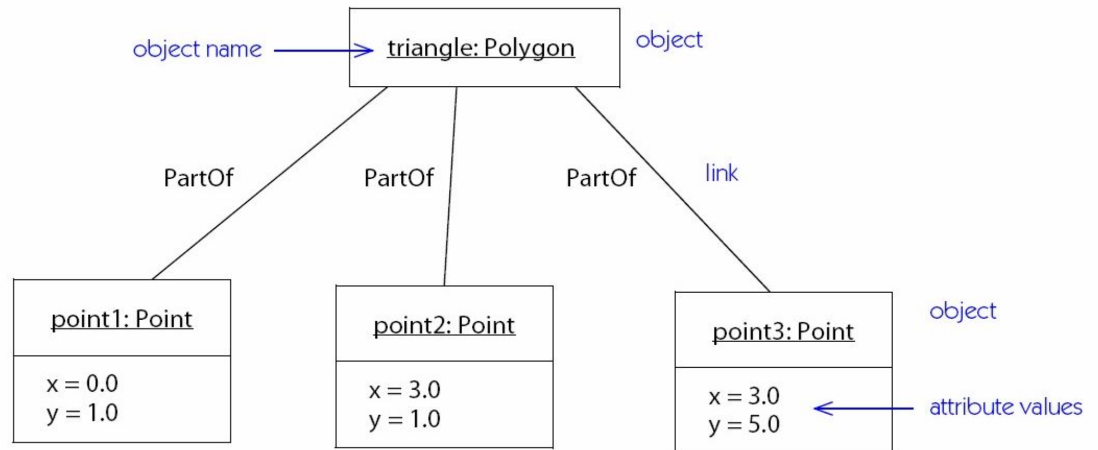
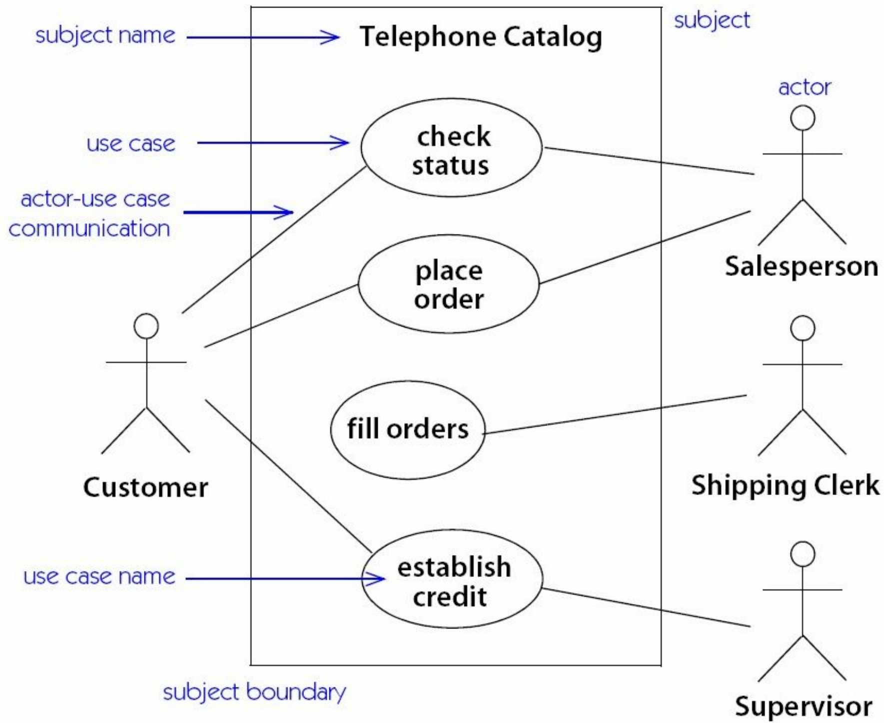
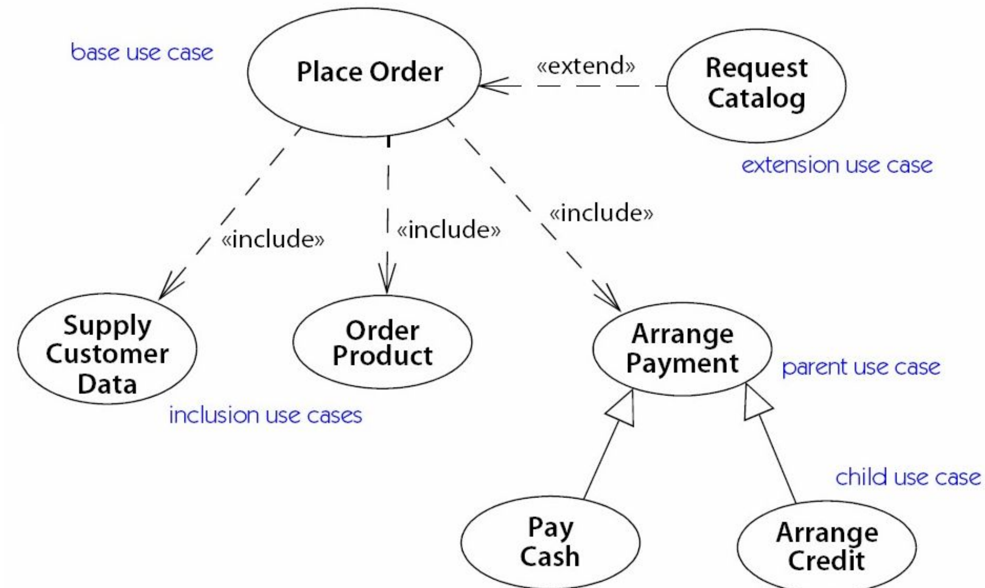


Диаграмма объектов



Описание вариантов использования

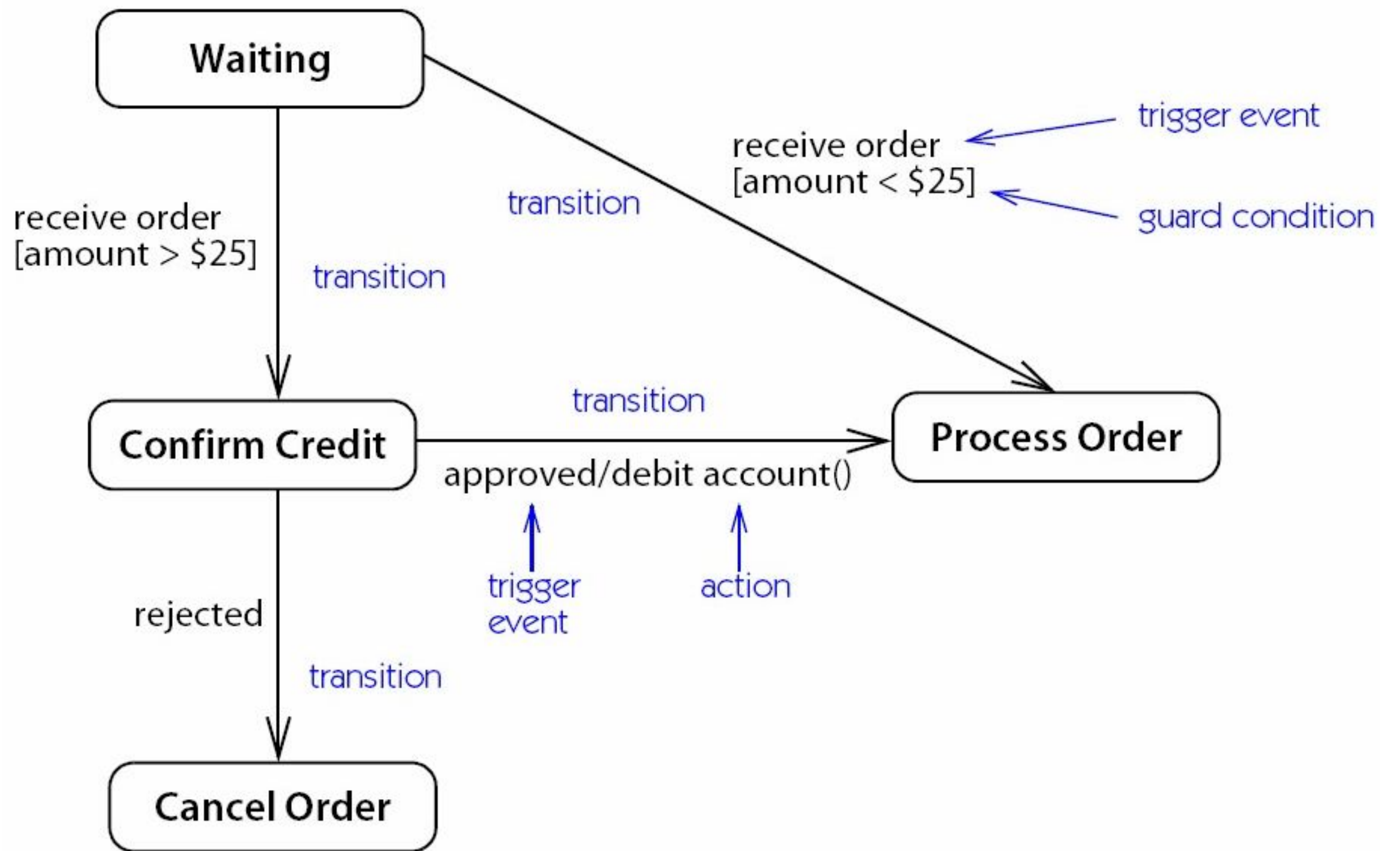




Описание дискретных автоматов

Диаграмма переходов состояний

[amount <= $25] – так правильно



Описание активности

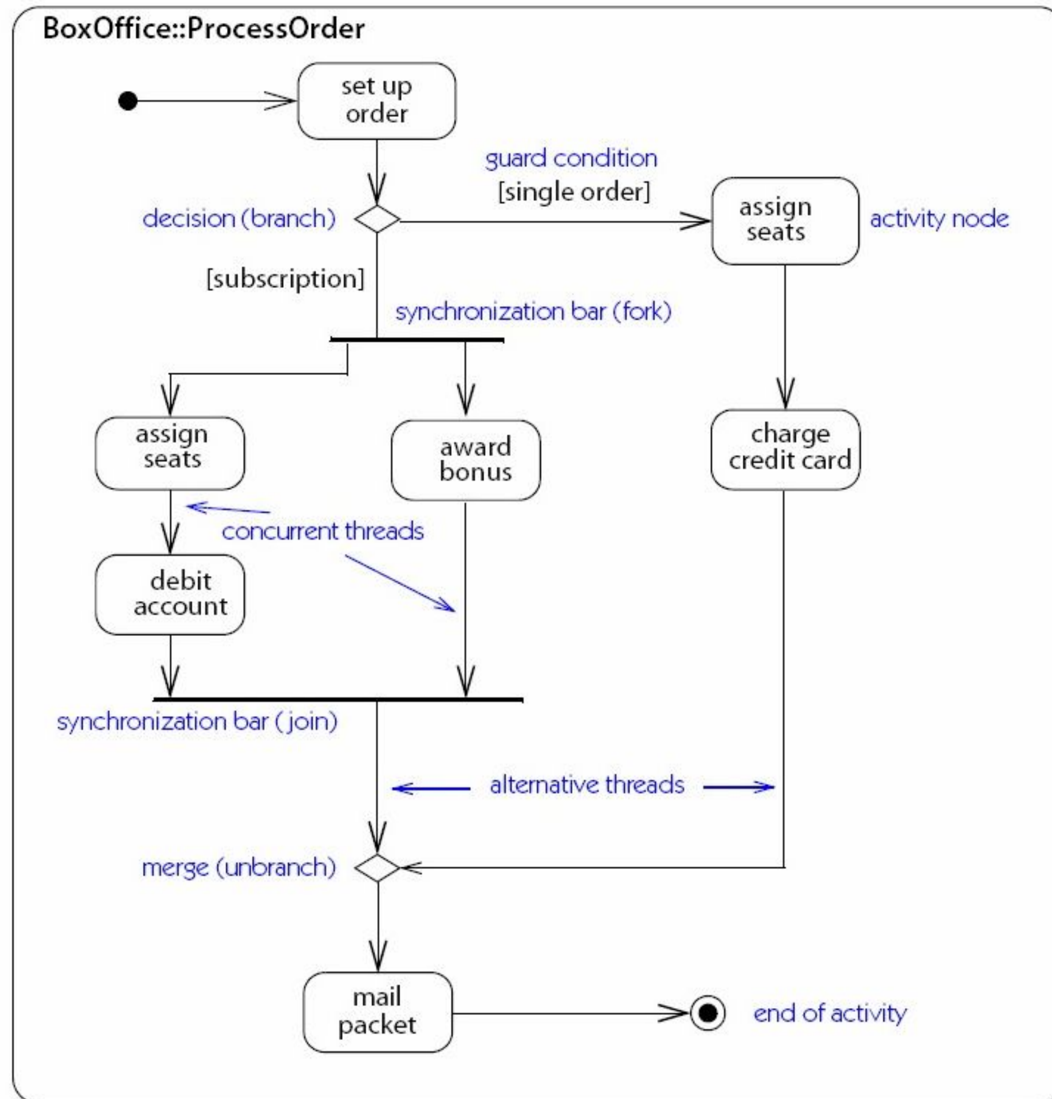
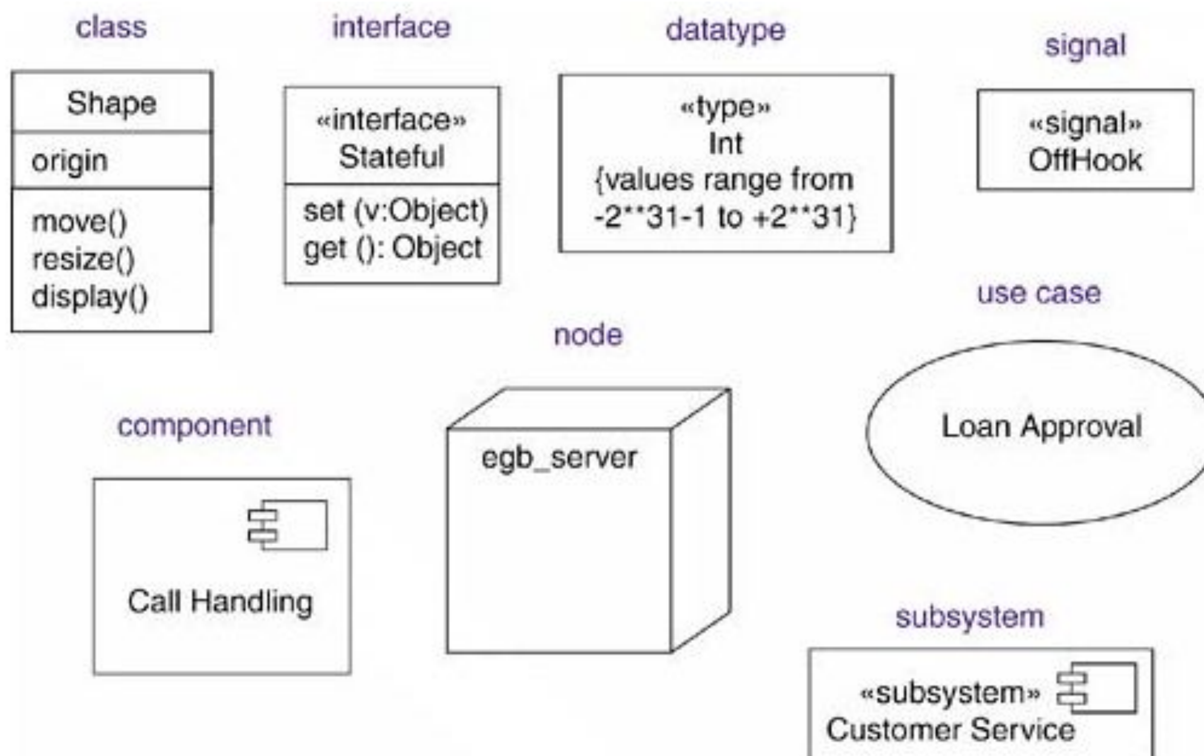


Figure 9-2. Classifiers

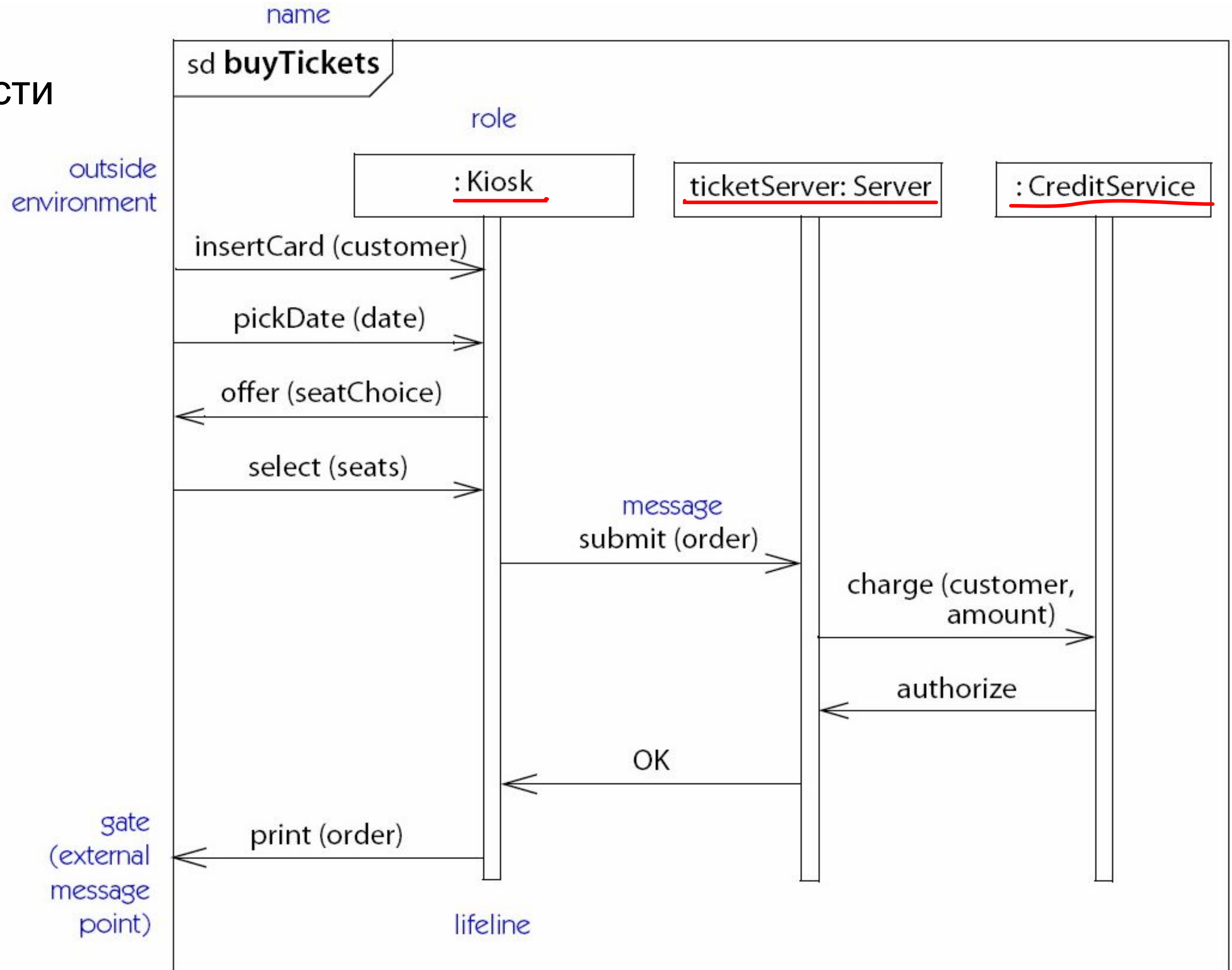
Описание взаимодействия

Классификатор – модельный элемент, который описывает поведенческие свойства (в виде операций) и структурные свойства (в виде атрибутов). Классификаторами являются: класс, интерфейс, компонент, вариант использования, подсистема, узел размещения и т.д.



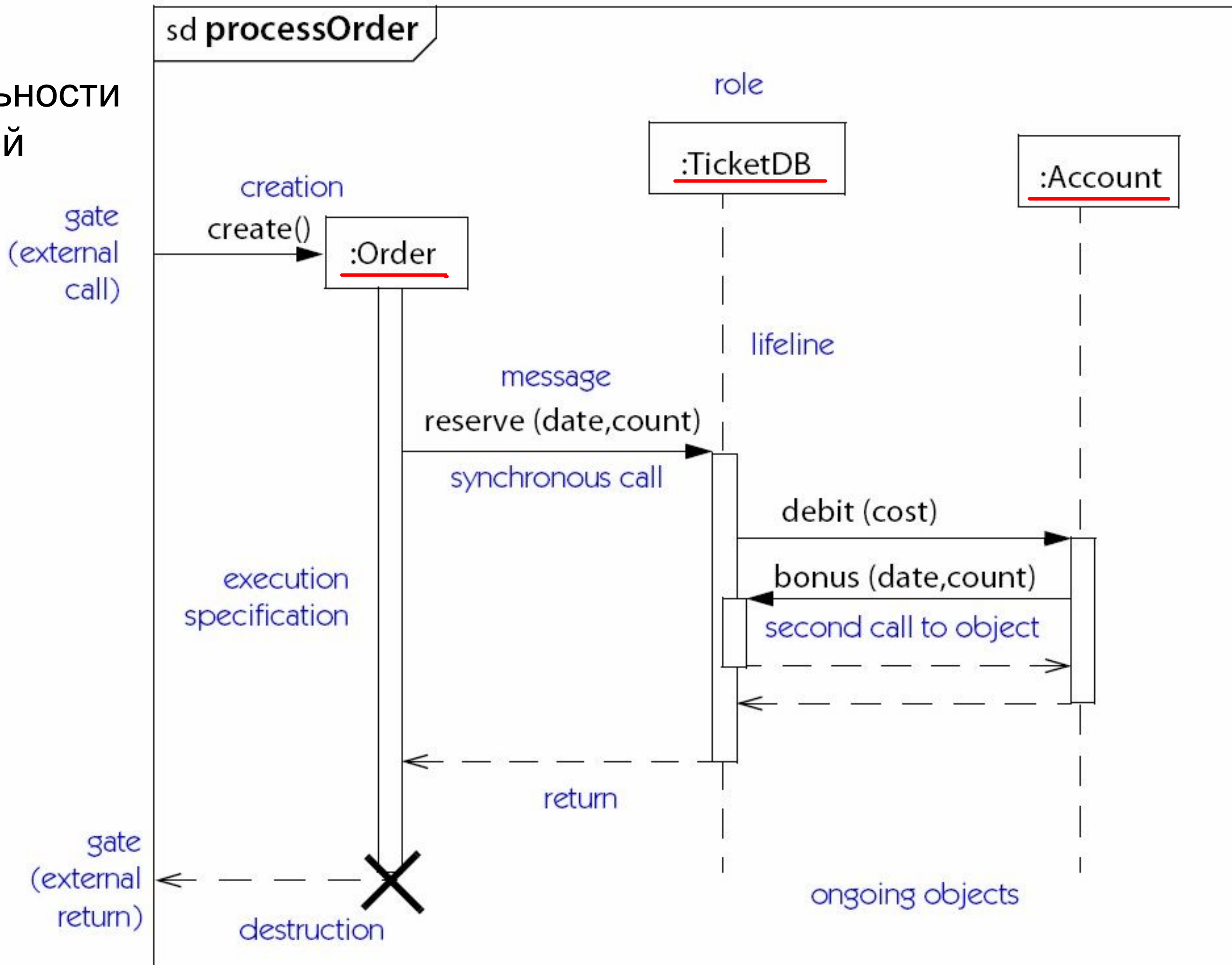
Описание взаимодействия

Диаграмма последовательности



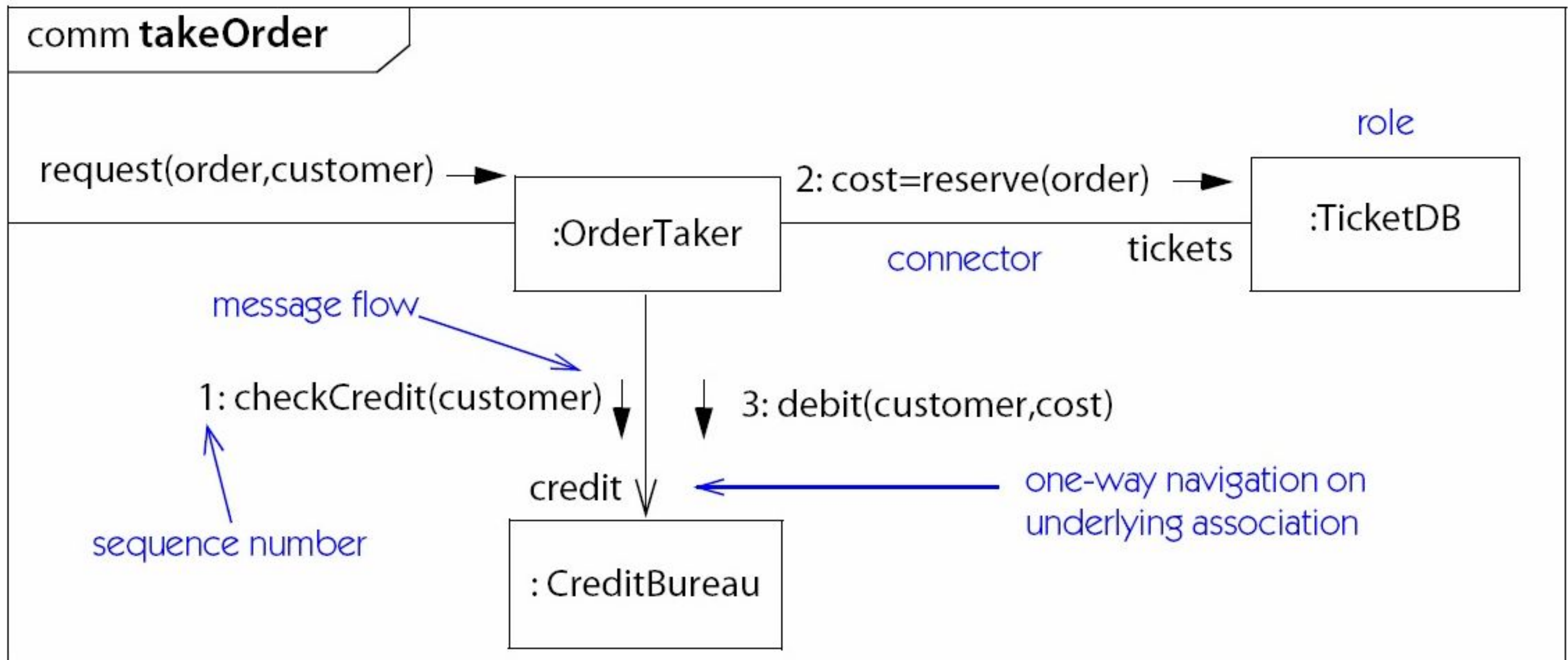
Описание взаимодействия

Диаграмма последовательности с детализацией выполнения



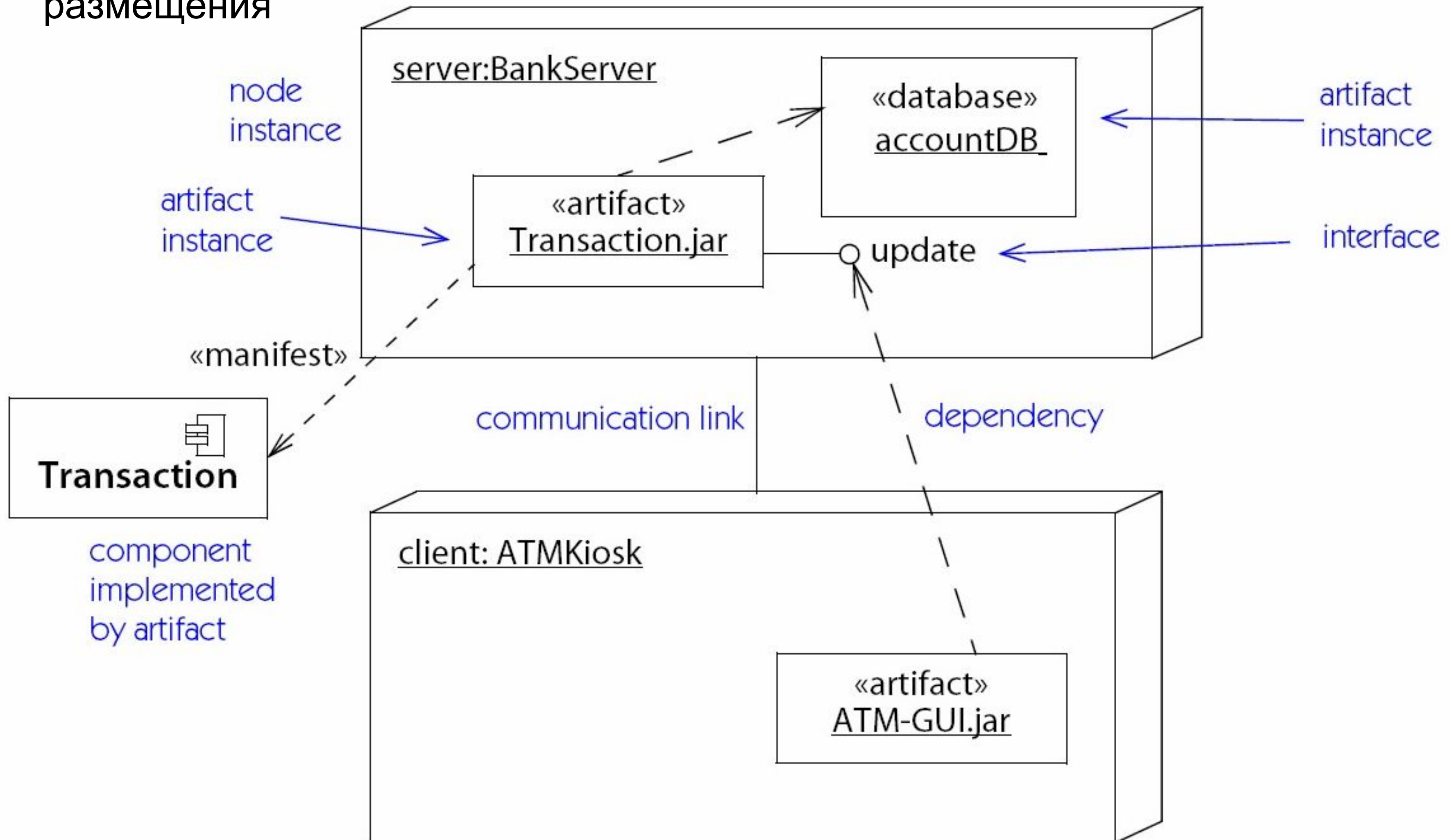
Описание взаимодействия

Коммуникационная диаграмма



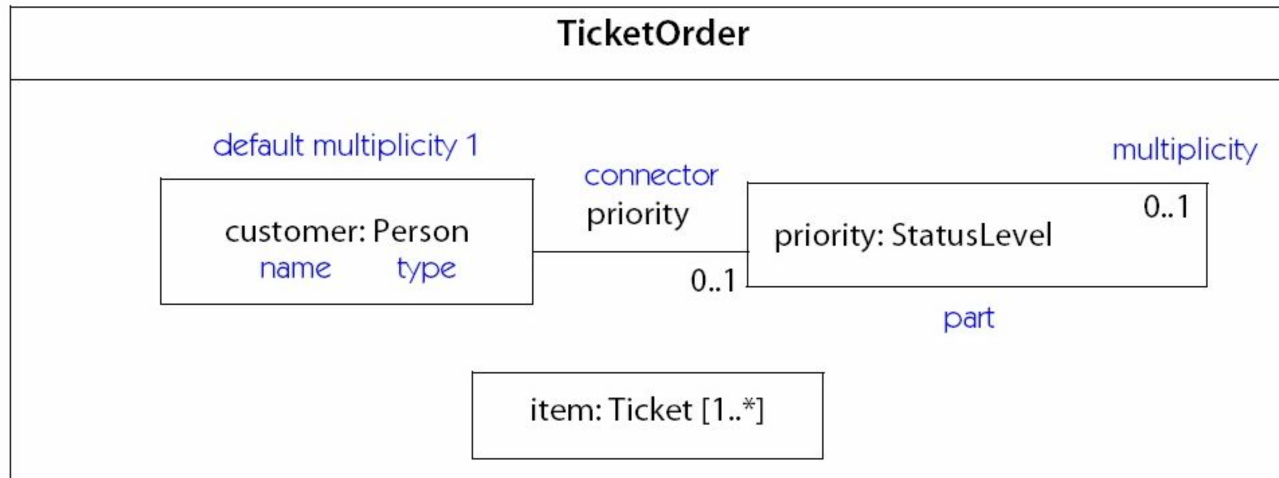
Описание размещения

Диаграмма размещения

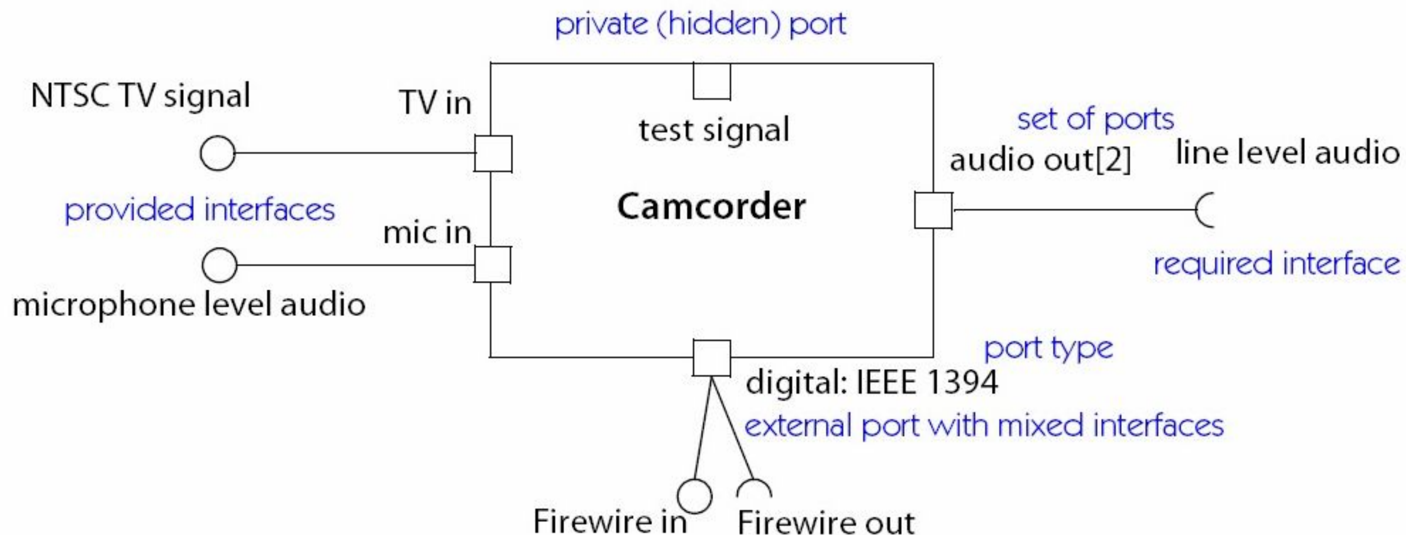


Описание проектных решений

Структурированный класс

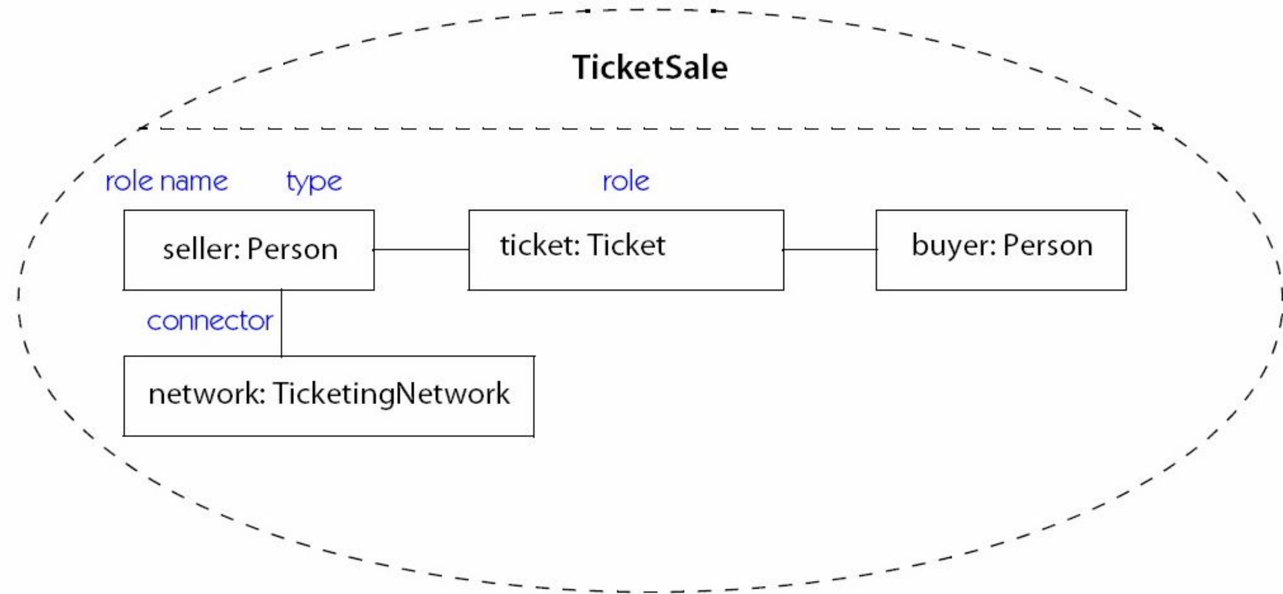


Структурированный класс с портами



Описание проектных решений

Описание сотрудничества объектов



Использование шаблона