Технология программирования

Основы объектно-ориентированного моделирования

Принципы объектно-ориентированного подхода

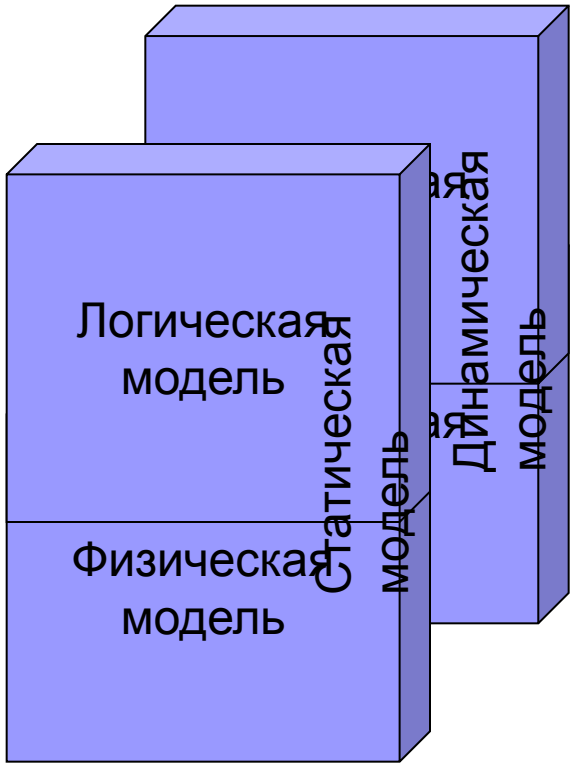
Объектно-ориентированный анализ – это *методология* системного **анализа**, направленная на создание моделей, близких к реальным явлениям. Требования к проектируемой системе формируются на основе понятий (классов и объектов), составляющих *словарь предметной области* (термины предметной области, необходимые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

Логическая модель: структуры классов и структуры объектов

Физическая модель: архитектура модулей и архитектура процессов



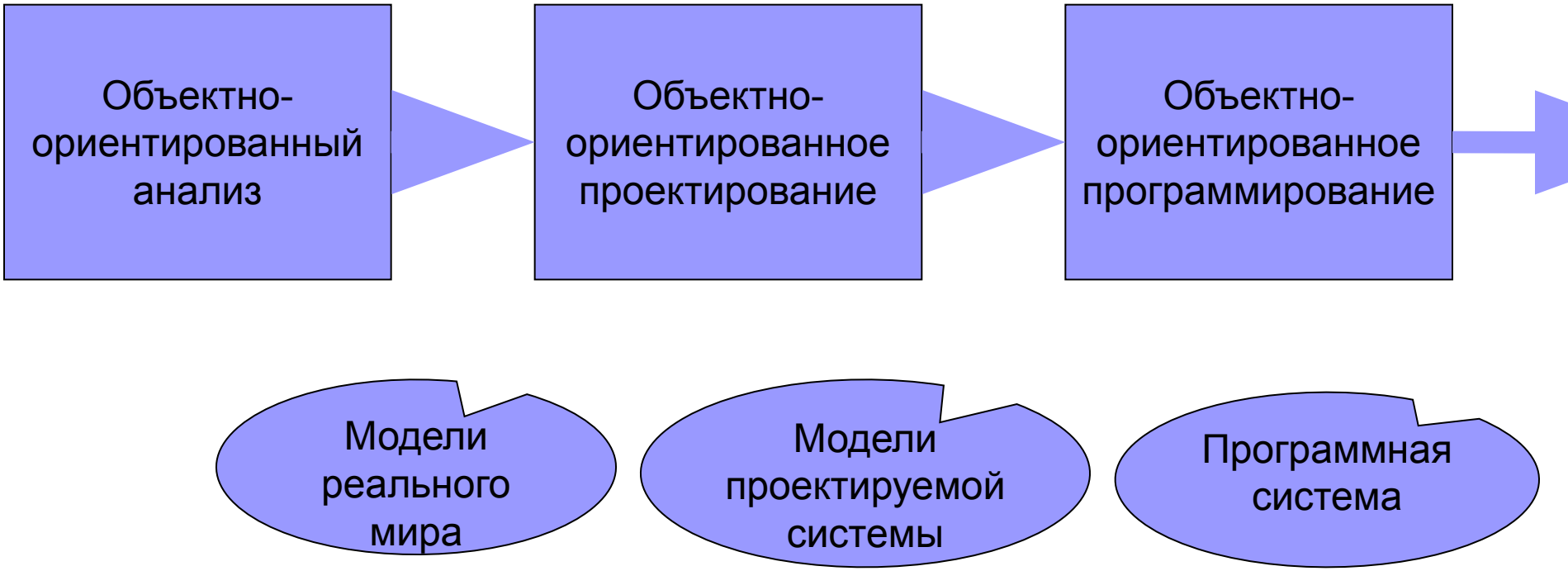
Модели объектно-ориентированного проектирования

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

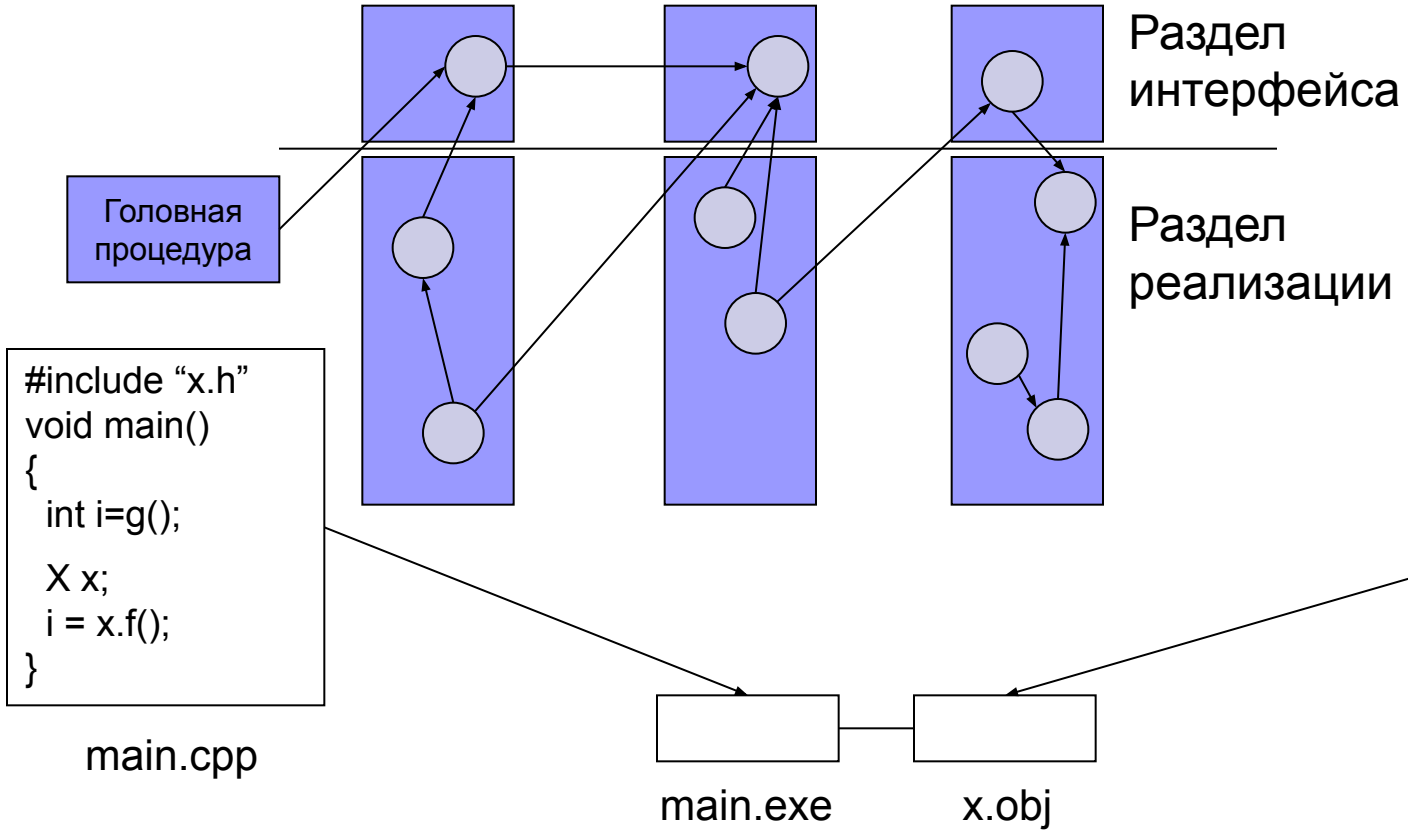
```
struct Point {int x, int y};
class Figure
{
private:
  Point _center;
public:
  Figure();
  void SetCenter(Point center);
  virtual void Draw();
  virtual void Hide();
  Point GetCenter();
}
```

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



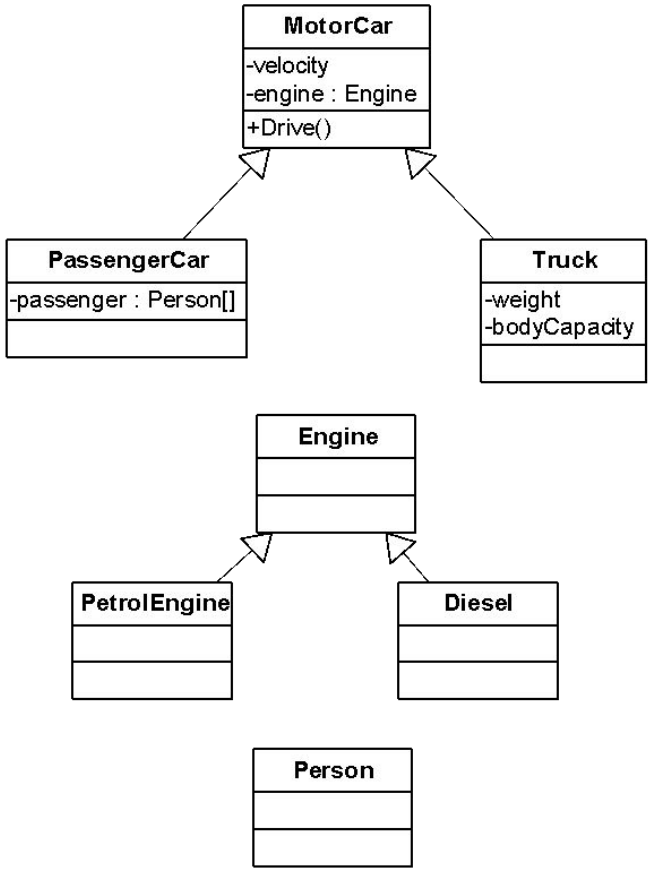
Физические модули: компонент, пакет (физическая группировка)
Логические модули: класс, подсистема (логическая группировка)

Характеристики: связность модуля, сцепление модулей

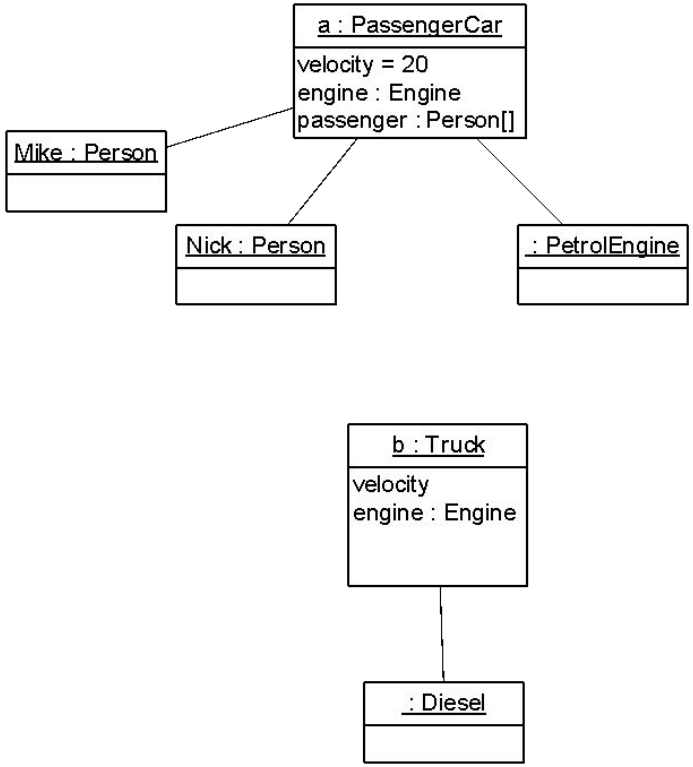
Основные понятия объектного моделирования

4. Иерархия

Иерархия классов



Иерархия объектов



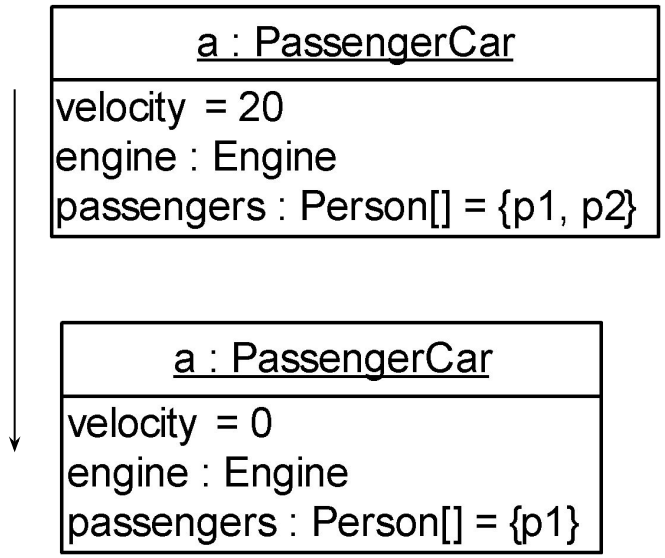
Основные понятия объектного моделирования

```
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;
}
```

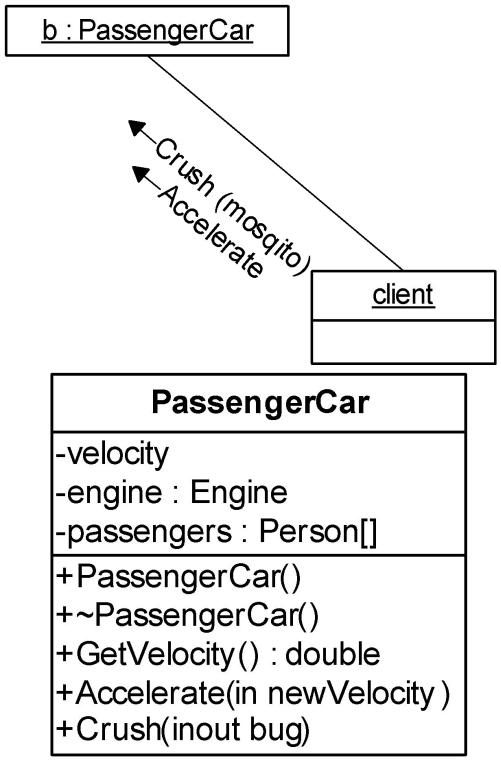
Объекты

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

Изменение **состояния** объекта



Поведение объекта

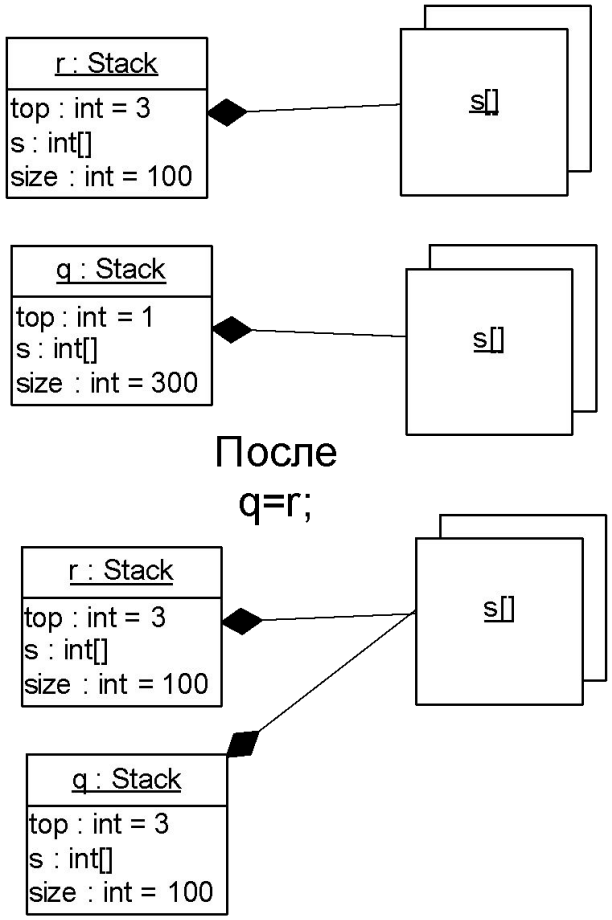



```
class Stack
{
private:
  int top;
  int *s;
  int size;
public:
  void Push(int i);
  int Pop();
  int IsEmpty();
  int IsFull();
  void Copy(Stack *other);
  Stack(int sz);
  ~Stack();
}
```

Объекты

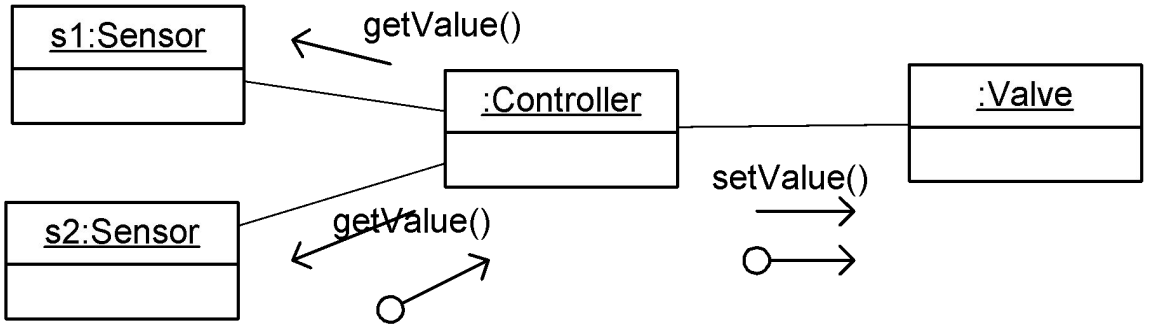
Индивидуальность объекта

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



Объекты

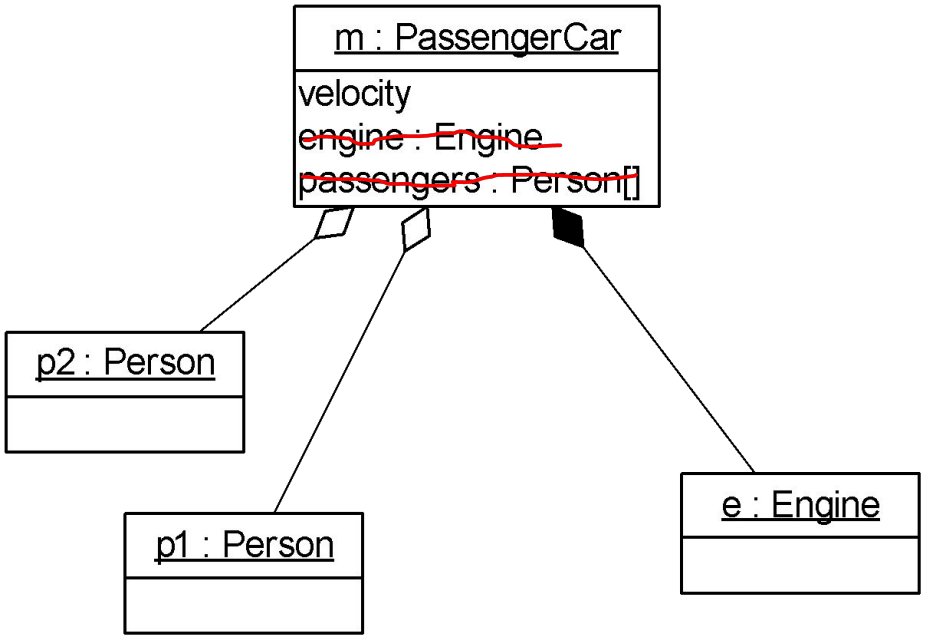
Связь



Отношения:

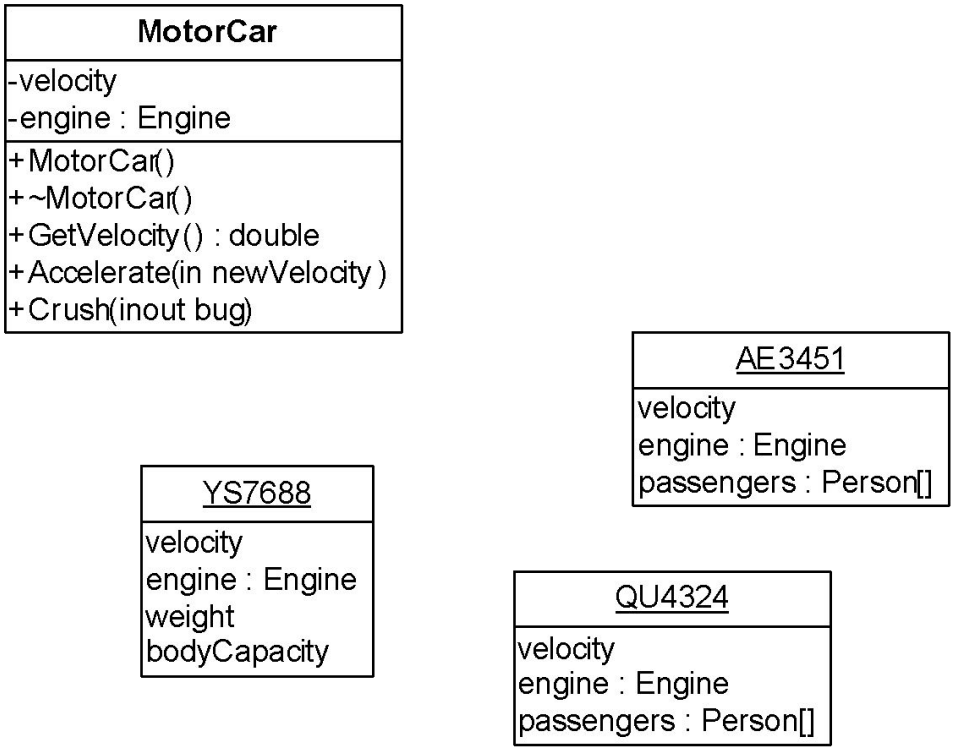
- **Связь** – взаимодействие между экземплярами сущностей
- **Агрегация** (агрегация по ссылке, разделяемая агрегация) - отношение «часть-целое»
- **Композиция** (агрегация по значению) – строгая форма агрегации, агрегируемый объект принадлежит только одному агрегату. Связаны жизненные циклы.

Агрегация



Классы

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



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)

Наследование

При наследовании подкласс может :

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

Принцип подстановки:

Экземпляр подкласса может быть использован при тех же условиях, при которых используется экземпляр суперкласса

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

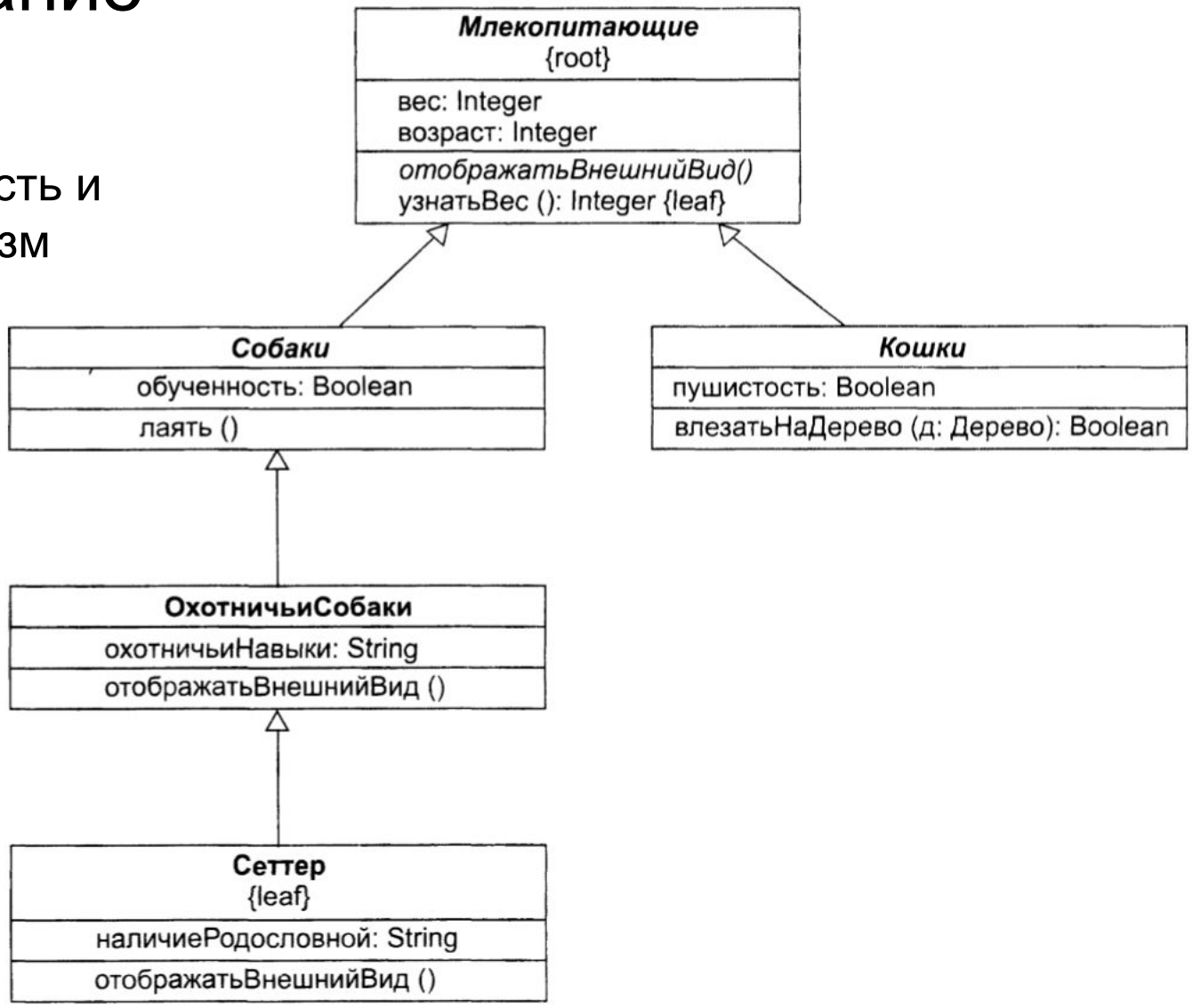
```
class Circle: public Figure
{
virtual void Show()
  {/*рисование окружности*/ };
};
class Face : public Circle
{
virtual void Show()
  {
    /* рисование глаз */
    Circle::Show();
   /* рисование рта и ушей */
  };
};
```

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

В C++ уточнение реализовано для конструкторов и деструкторов

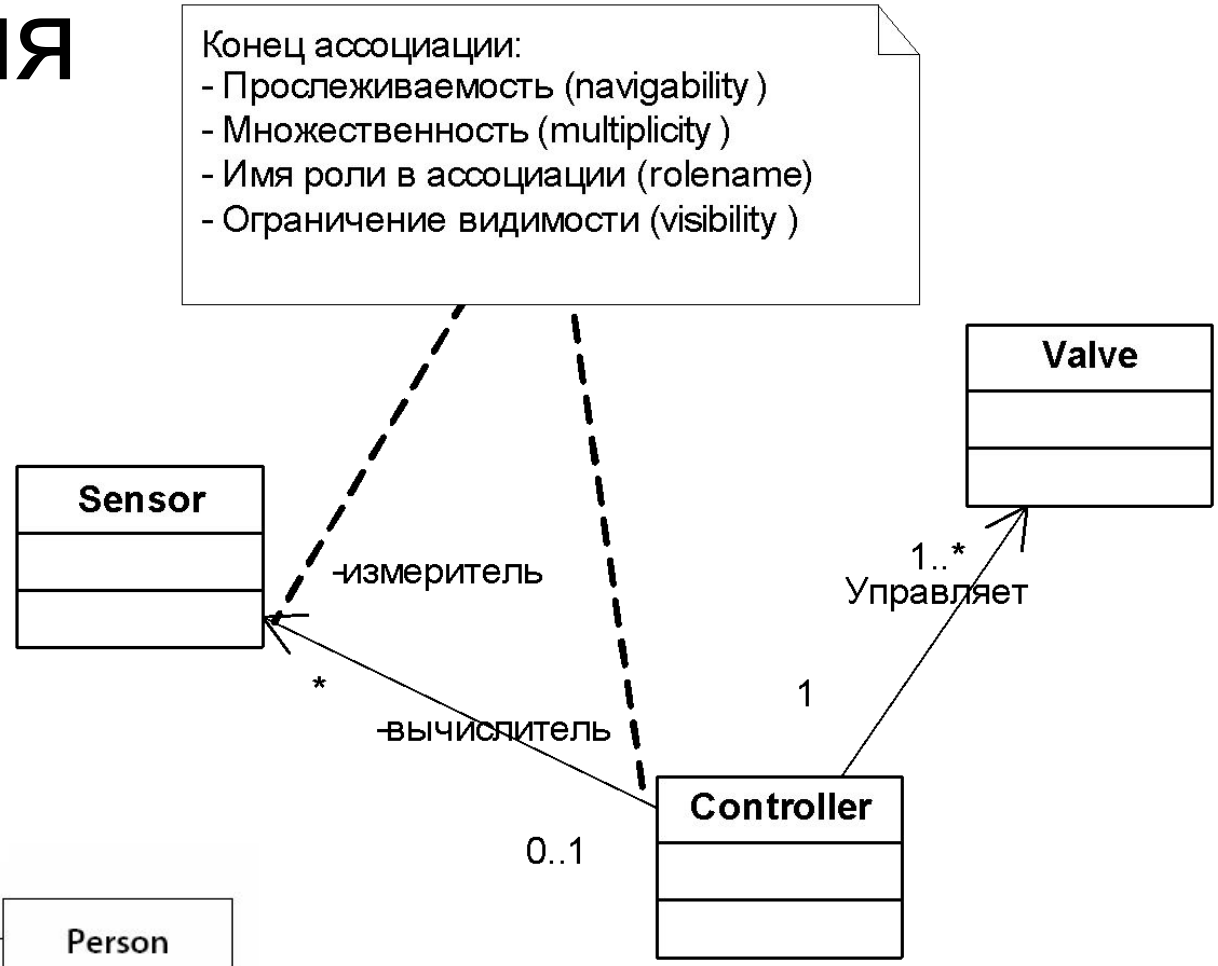
Наследование

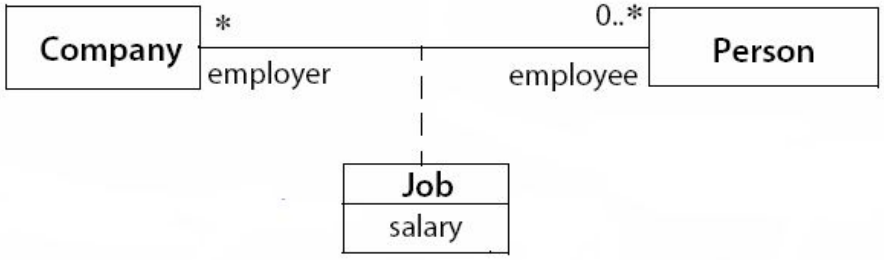
Абстрактность и полиморфизм



Ассоциация

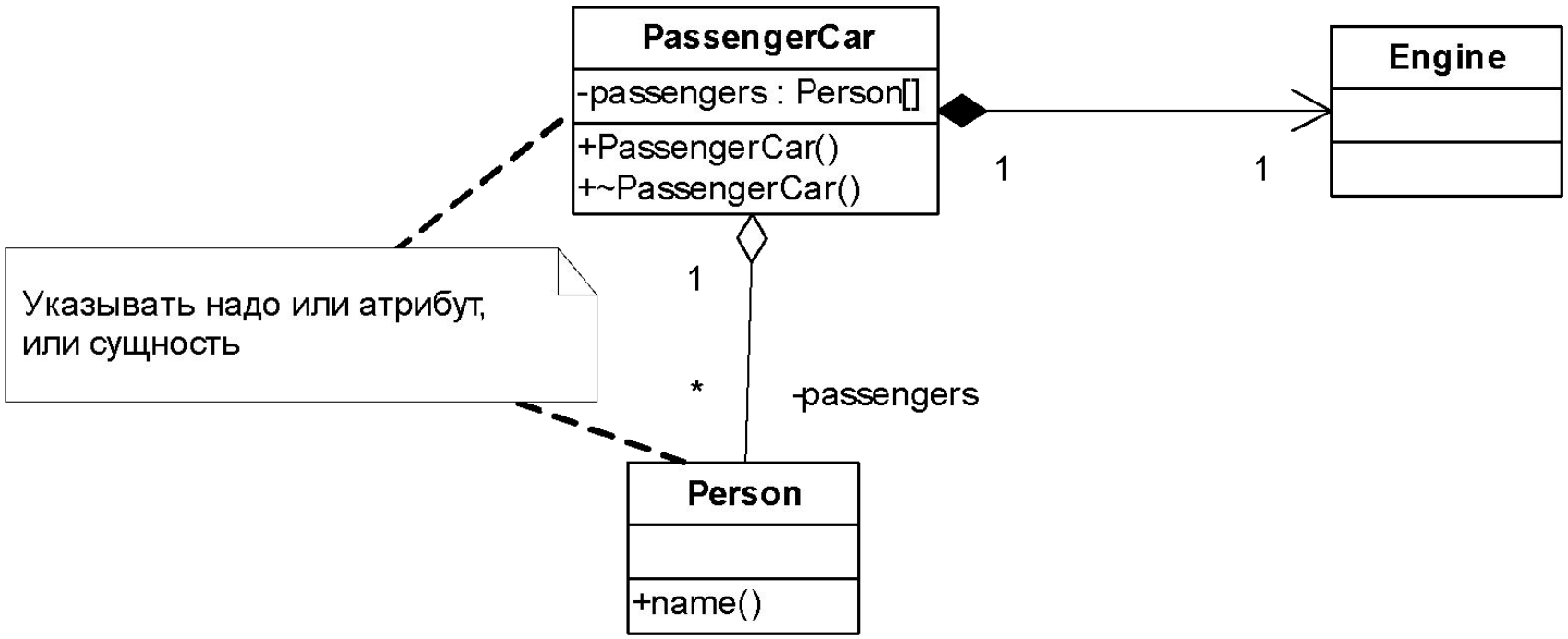
```
class Controller
{
private:
  Sensor _sensor[];
}

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



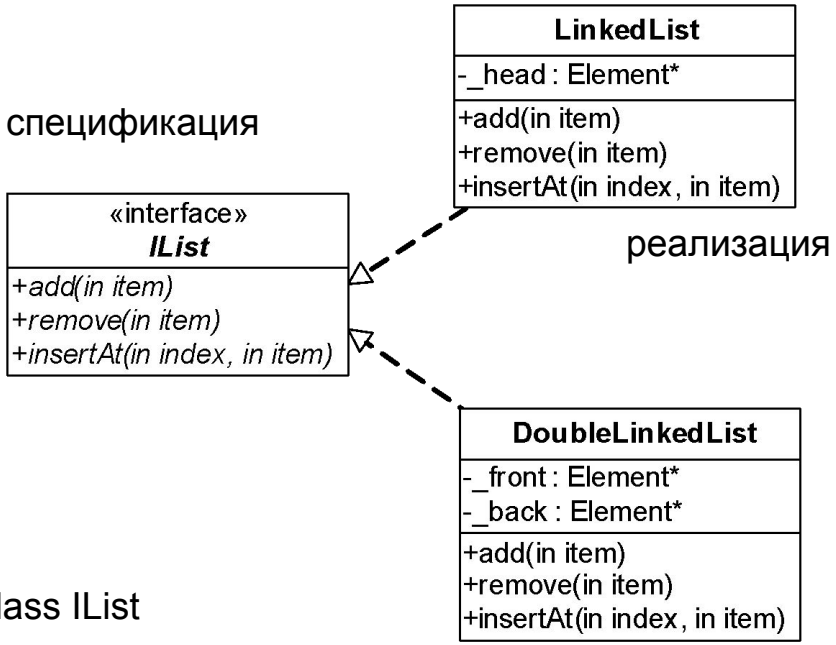


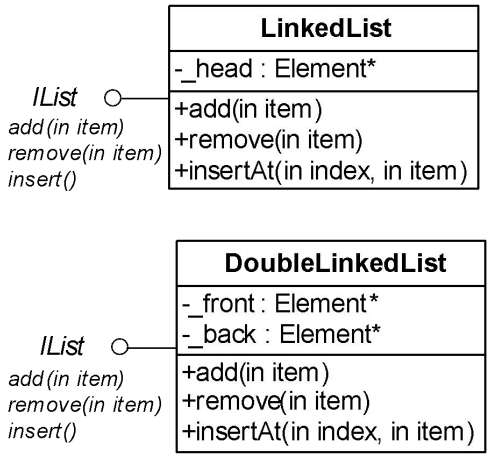

```
class Job
{
  public:
   Company* company;
   Person* person;
   double salary;
}
```

Агрегация


PassengerCar
-passengers : Person[]
+PassengerCar()
+~PassengerCar()
Engine
1
1
1
*
-passengers
Person
+name()
Указывать надо или атрибут,
или сущность

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;
}
```

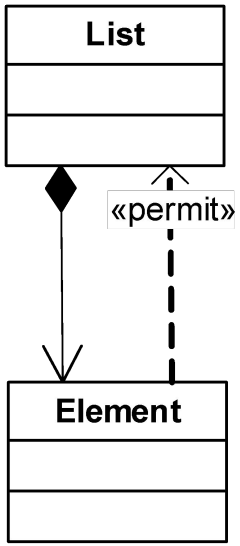
```
class LinkedList : public IList
{
private:
   Element* _head;
public:
  void add(string& item)
     {…  Element * p = new Element(item) …  }
  void remove(string& item)  {… delete p; …}
  void insertAt(int index, string& item) { … }
}
```

Зависимость

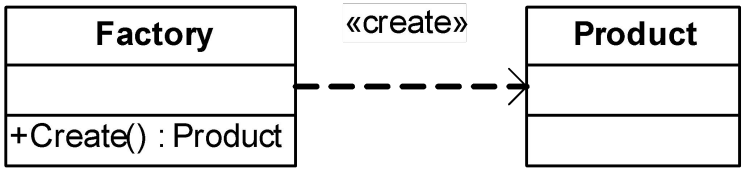
Стереотипы отношения зависимости:

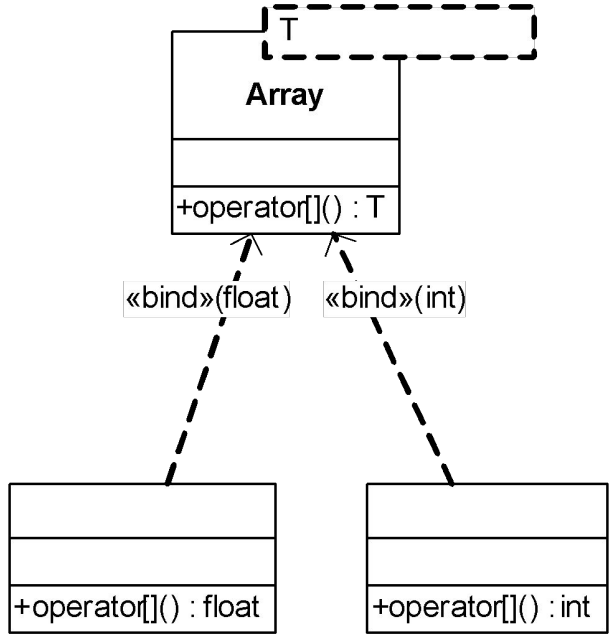
- <<bind>> – назначение параметров шаблонному классу для получения нового конкретного класса
- <<call>> – метод одного класса вызывает операцию другого класса
- <<create>> – один класс создает экземпляр другого класса
- <<permit>> или <<friend>> – разрешение одному классу использовать реализацию другого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

Зависимость




```
class List;

class Element
{
  friend class List;
}

class List
{
   Element* _head;
}
```



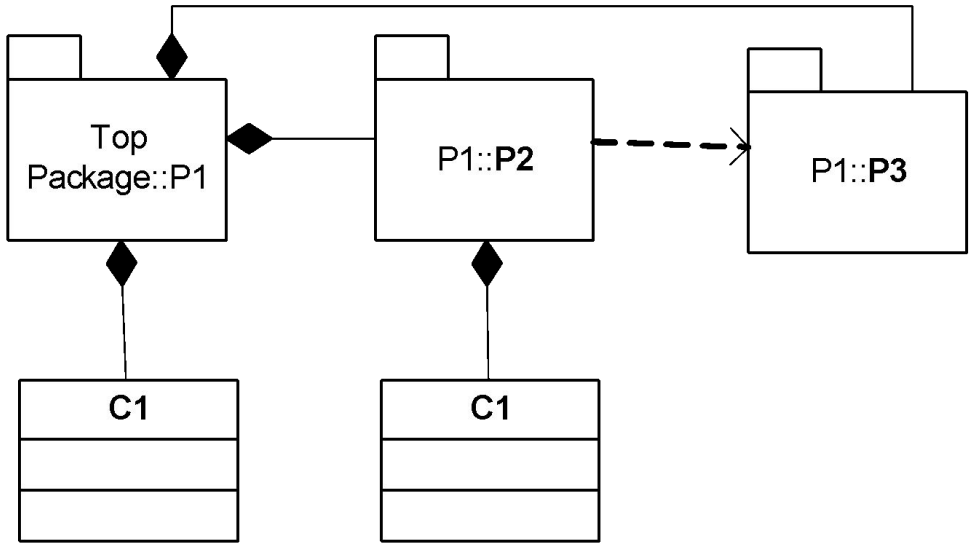



```
template<class T>
class Array
{
public:
  T operator[](int i) {…}
}

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

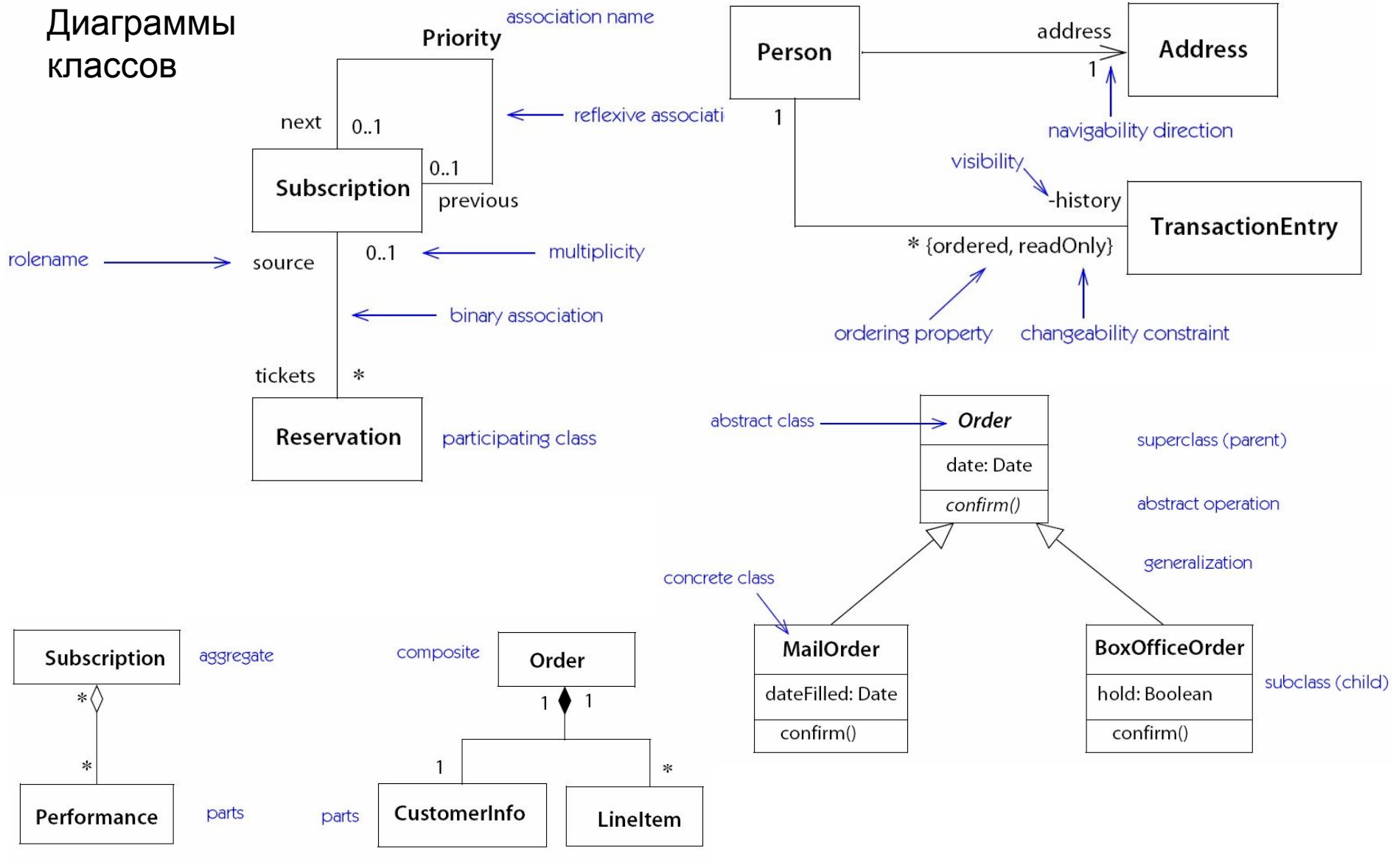
Представление (View) – это подмножество конструкций UML, отражающих один аспект системы.

- 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

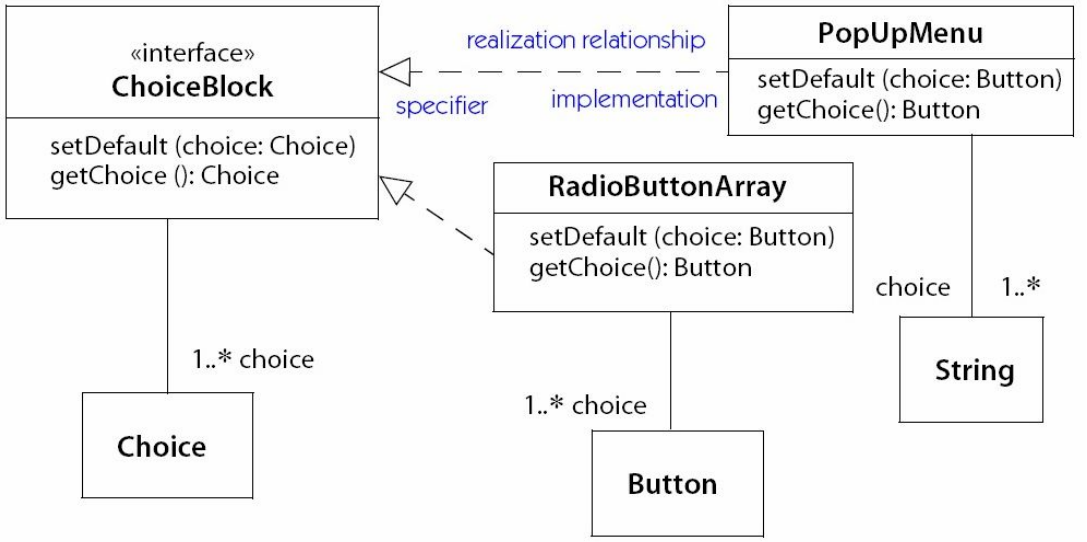
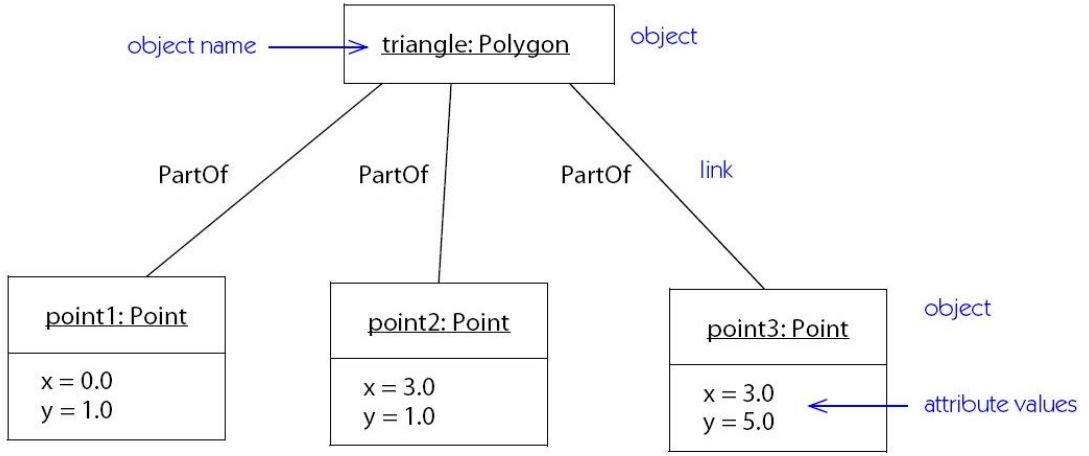


Диаграмма объектов



Диаграммы коммуникации

Обозначение объекта

Синтаксис представления имени имеет вид: имяОбъекта : ИмяКласса

Примеры записи имени.

адам : Человек	Имя объекта и класса
: Пользователь	Только имя класса (анонимный объект)
мойКомпьютер	Только имя объекта (подразумевается, что имя класса известно)
агент :	Объект – сирота (подразумевается, что имя класса неизвестно)

Синтаксис представления атрибута имеет вид: имя : Тип = Значение

Примеры записи атрибута.

номер:Телефон = '7350-120'	Имя, тип, значение
активен = True	Имя и значение

Диаграммы коммуникации

В языке UML моделируются следующие разновидности действий.

вызов (call)	В объекте запускается операция
возврат (return)	Возврат значения в вызывающий объект
посылка (send)	В объект посылается сигнал
создание	Создание объекта, выполняется по стандартному сообщению <<create>>
уничтожение	Уничтожение объекта, выполняется по стандартному сообщению <<destroy>>

Диаграммы коммуникации

Для записи сообщений в языке UML принят следующий синтаксис:

имя Атрибута = имяСообщения (Аргументы): ВозвращаемоеЗначение,

где имяАтрибута задает атрибут, куда помещается возвращаемое значение.

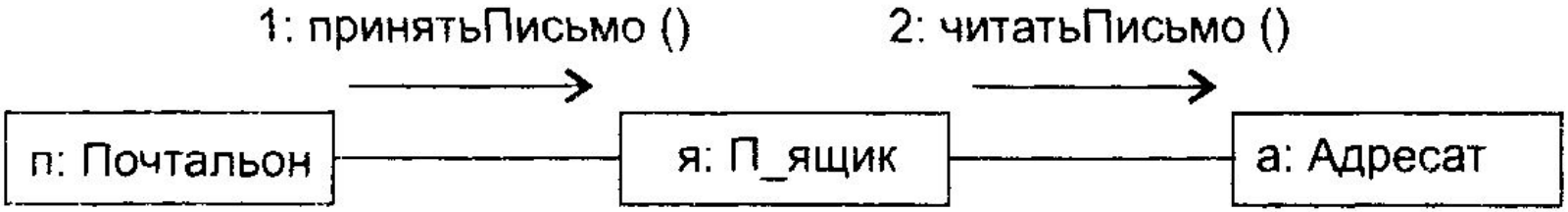
Примеры записи сообщений.

коорд = текущПоложение(самолетТ1)	Вызов операции, возврат значения
оповещение()	Посылка сигнала
установитьМаршрут(х)	Вызов операции с действительным параметром
<<create>>	Стандартное сообщение для создания объекта

Диаграммы коммуникации

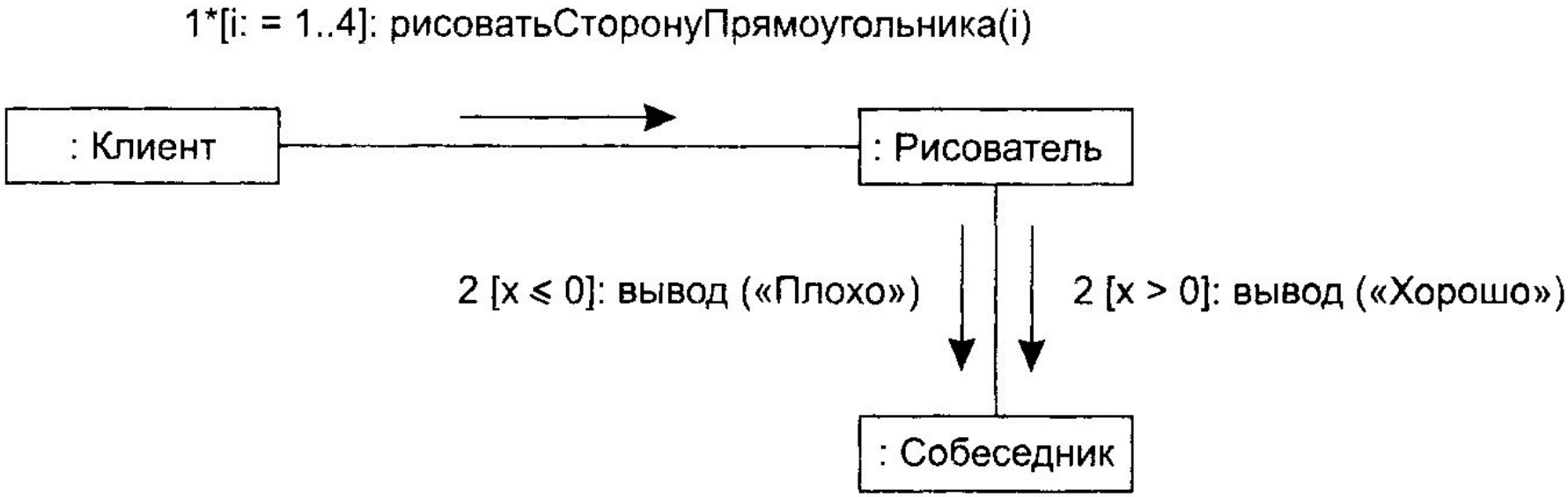
Поток синхронных сообщений.

Поток асинхронных сообщений.



Диаграммы коммуникации

Итерация и ветвление.



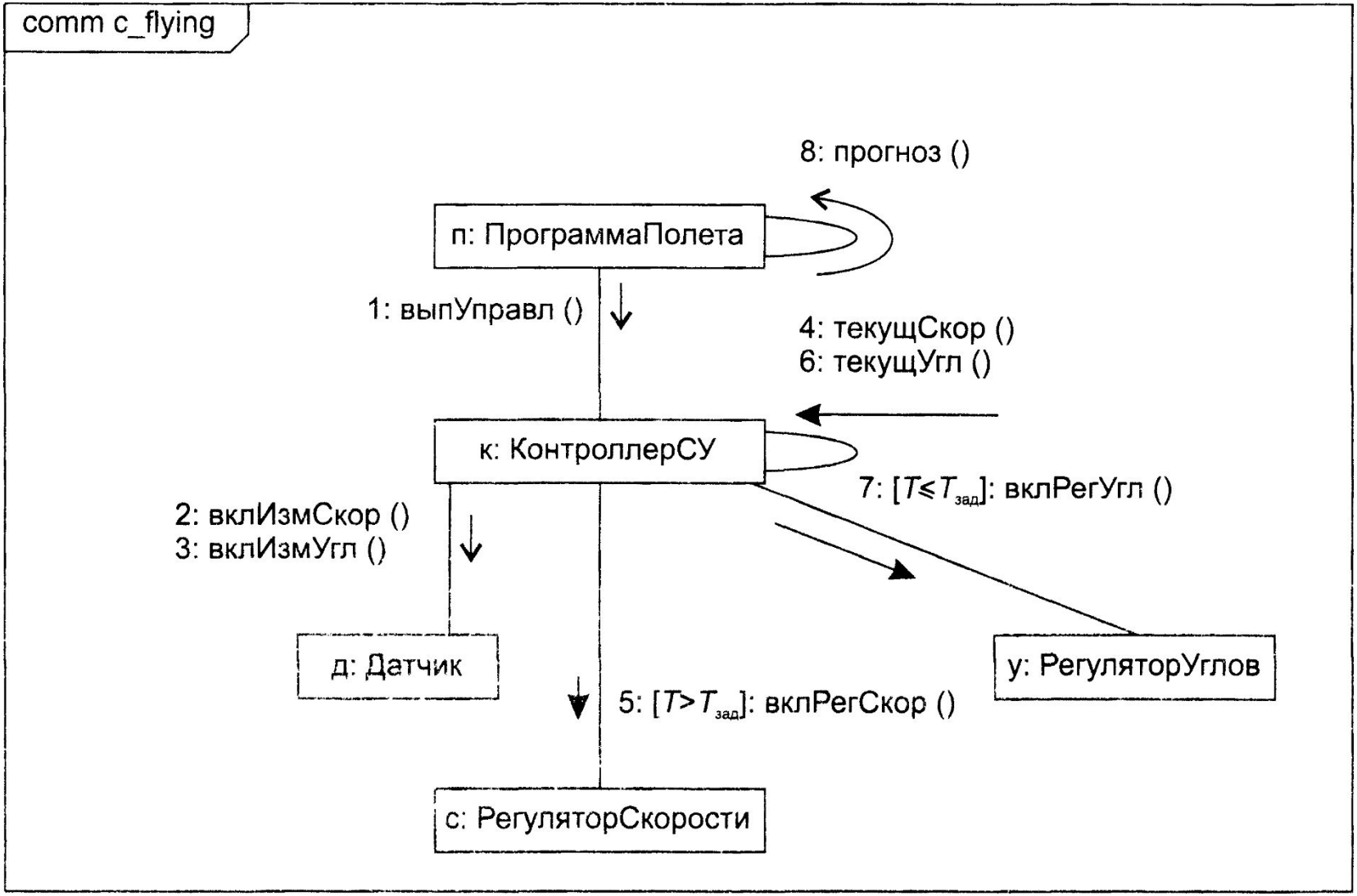
Диаграммы коммуникации

Для формирования диаграммы коммуникации выполняются следующие действия:

1) отображаются участники взаимодействия;
2) рисуются связи, соединяющие этих участников;
3) связи помечаются сообщениями, которые посылают и получают определенные участники.

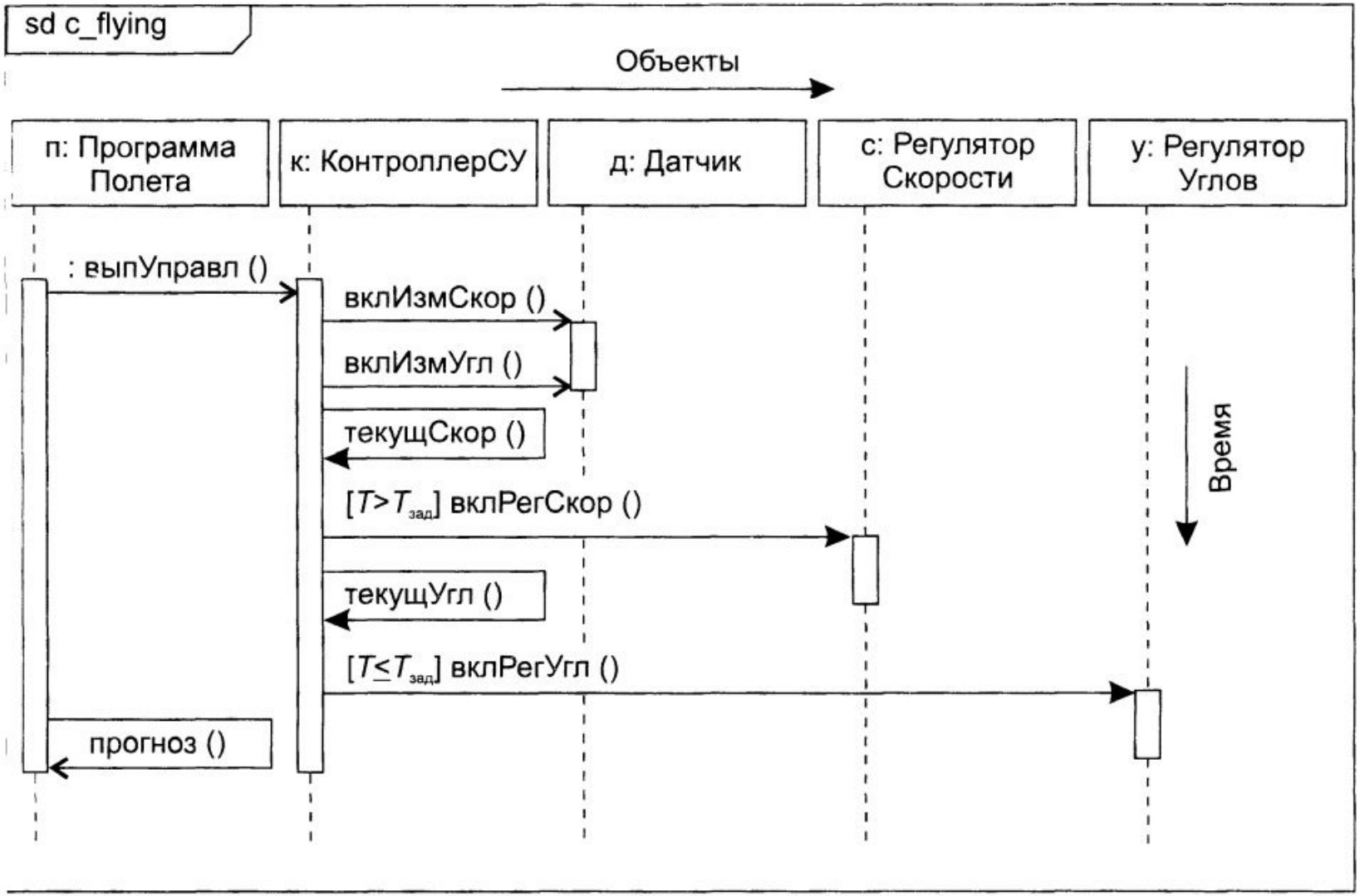
Диаграммы коммуникации

Диаграмма коммуникации системы управления полетом.



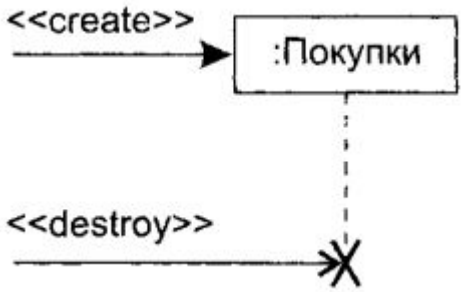
Диаграммы последовательности

Диаграммы последовательности системы управления полетом.

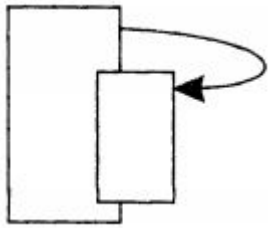


Диаграммы последовательности

Создание и уничтожение обобщенного объекта.



Вложение спецификаций выполнения (активаций)



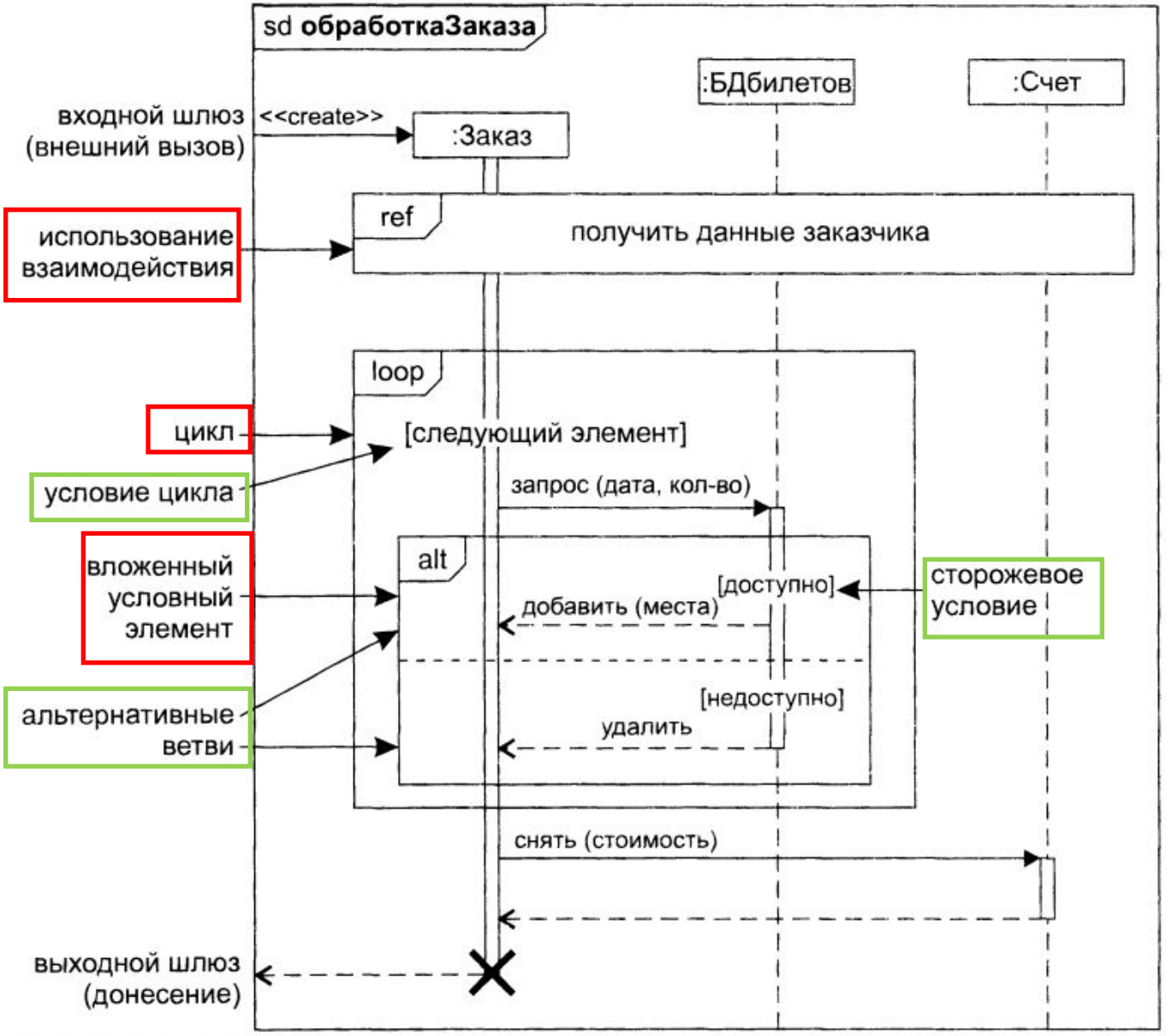
Диаграммы последовательности

Наиболее популярны следующие комбинированные фрагменты:

- Использование взаимодействия (interaction use)
- Цикл (ключевое слово loop).
- Условный фрагмент (ключевое слово alt)
- Необязательный фрагмент (ключевое слово opt)
- Параллельный фрагмент (ключевое слово par)

Диаграммы последовательности

Диаграмма последова-тельности с вложенными фрагментами.



Диаграммы последовательности

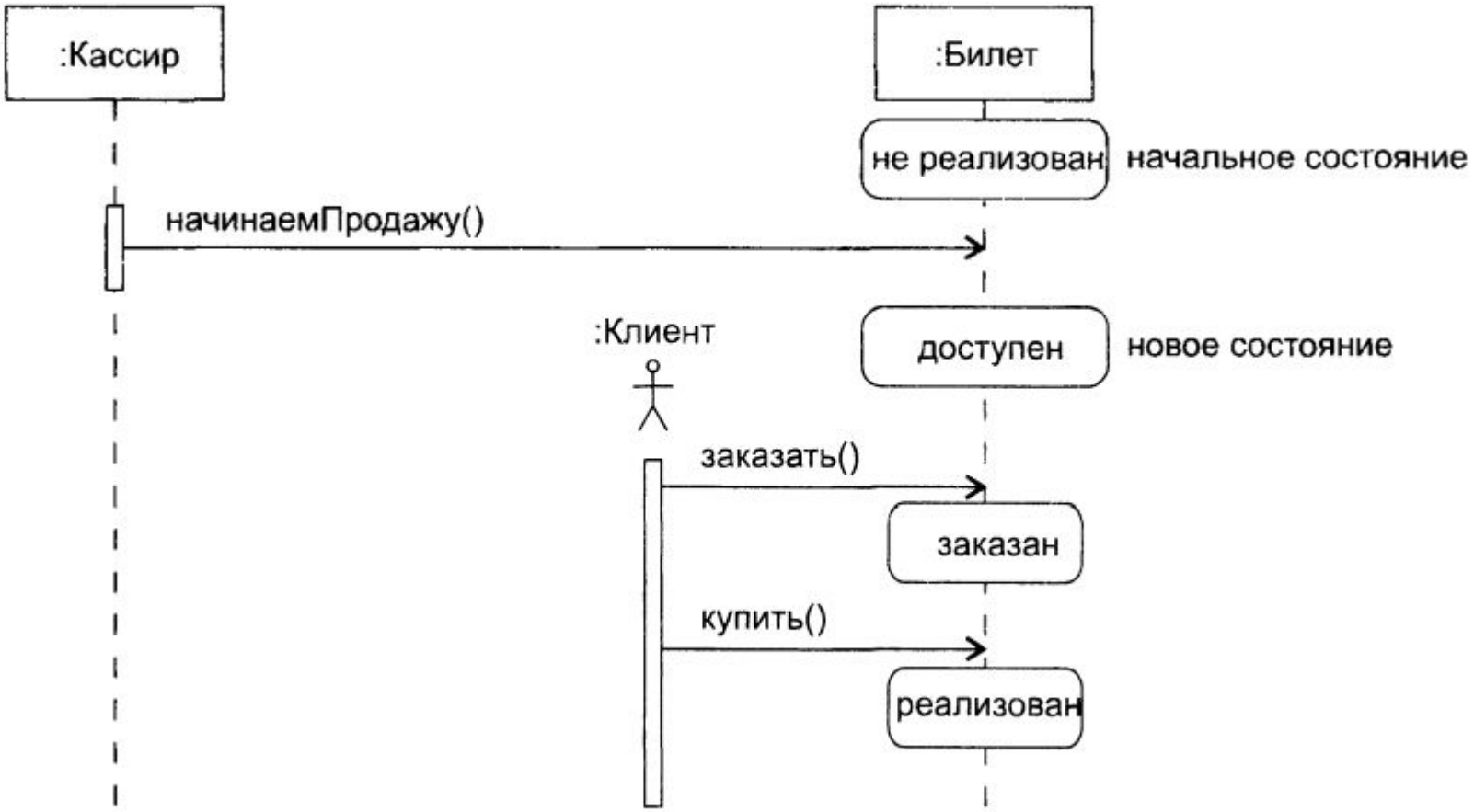
Ограничения на количество итераций фрагмента-цикла указываются в скобках после ключевого слова loop:

```
loop              минимум = 0, максимум неограничен
loop(количество)     минимум = максимум = количество
loop(мин, макс)  явное задание минимальной и максимальной границ.
```

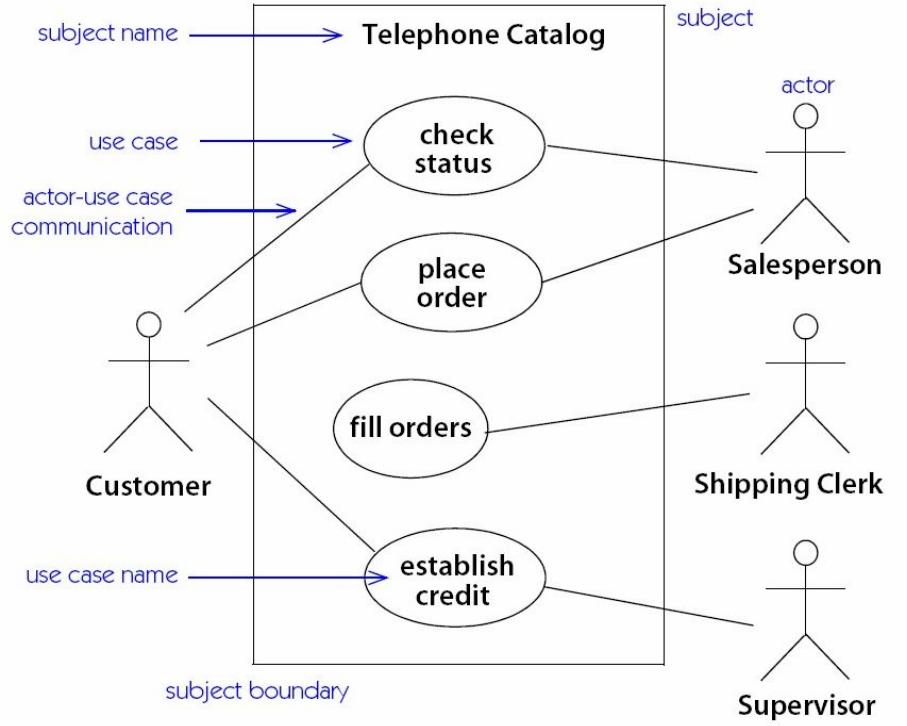
Границы числа итераций

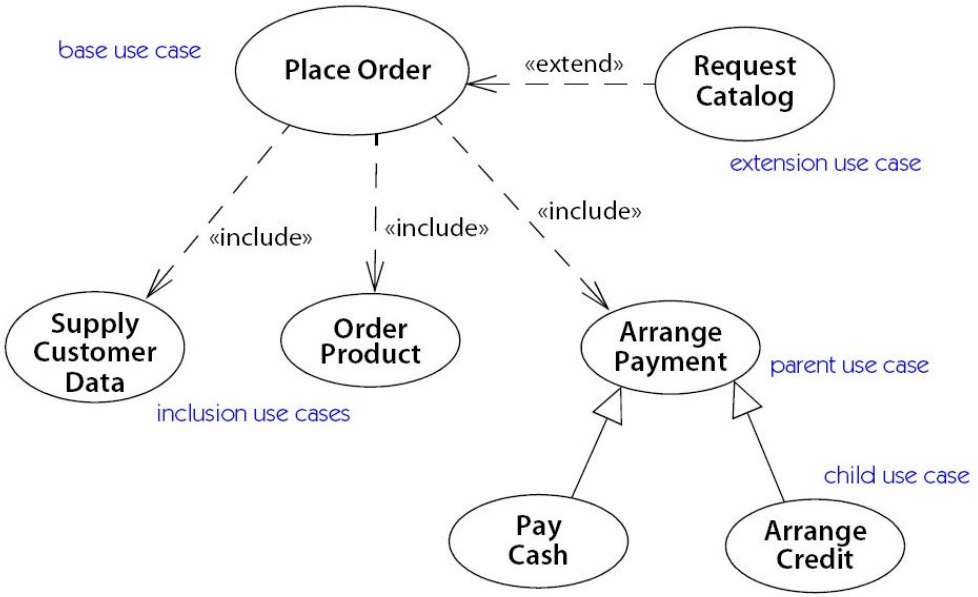
Диаграммы последовательности

Состояние объекта на диаграмме последовательности.



Описание вариантов использования

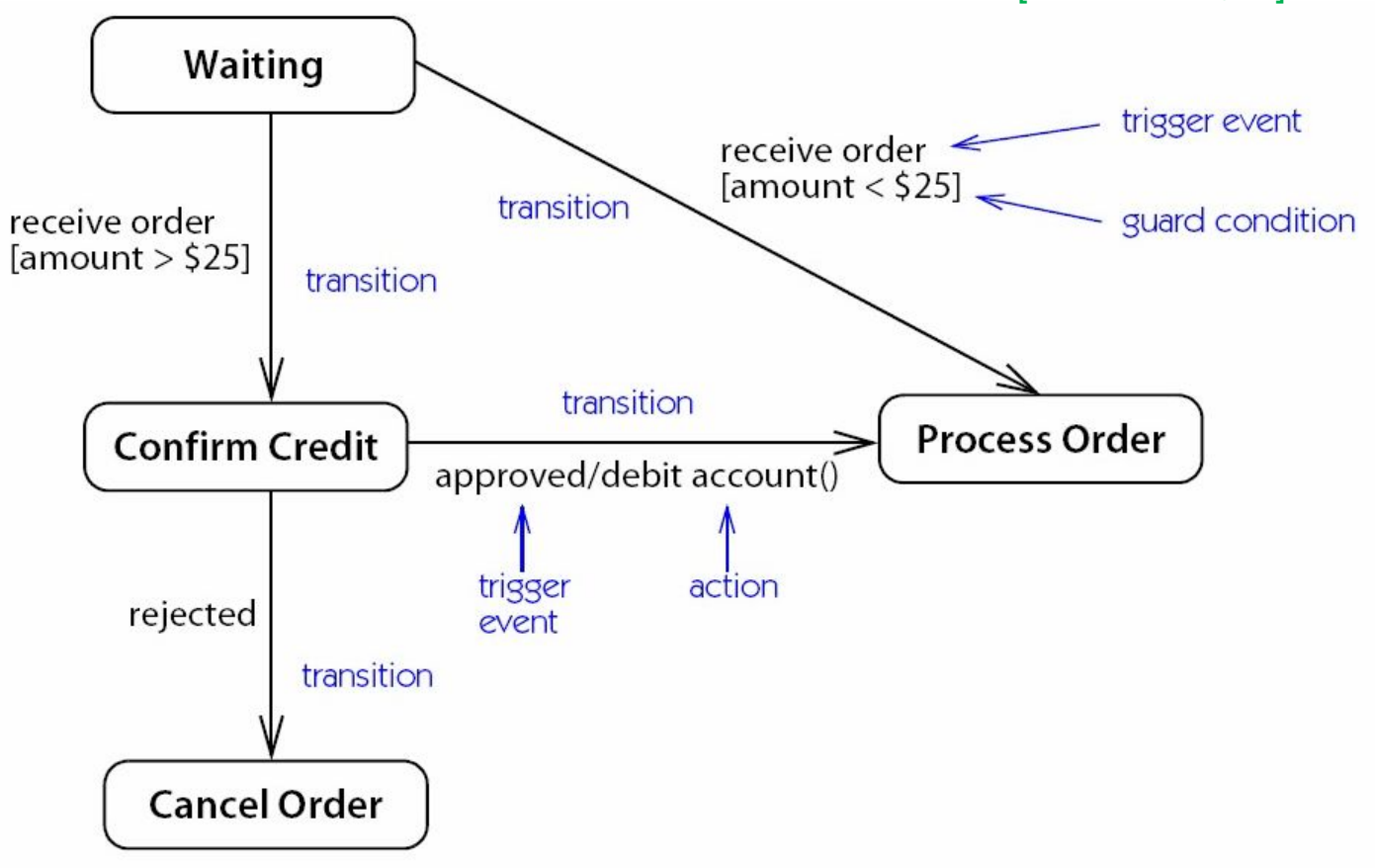




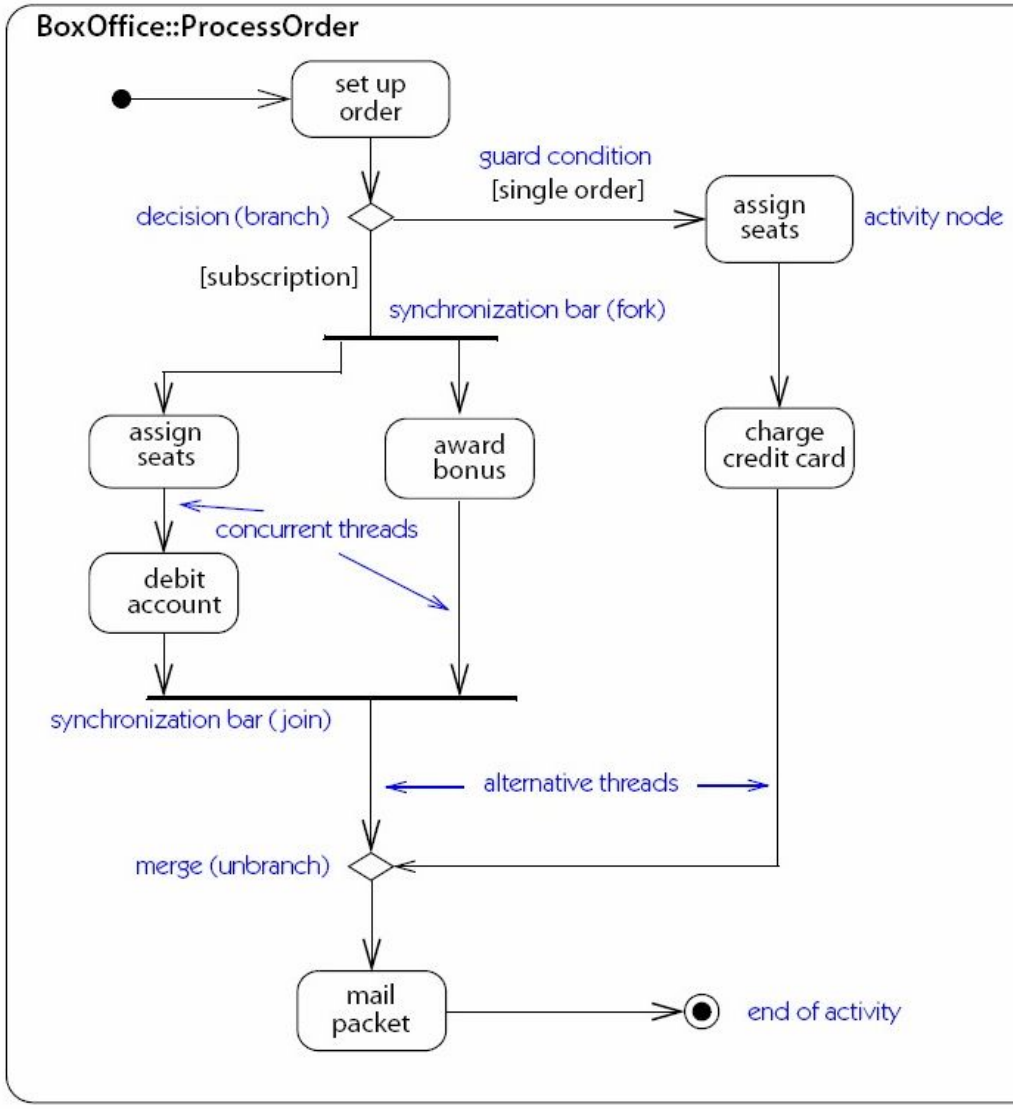
Описание дискретных автоматов

Диаграмма переходов состояний

[amount <= $25] – так правильно



Описание активности



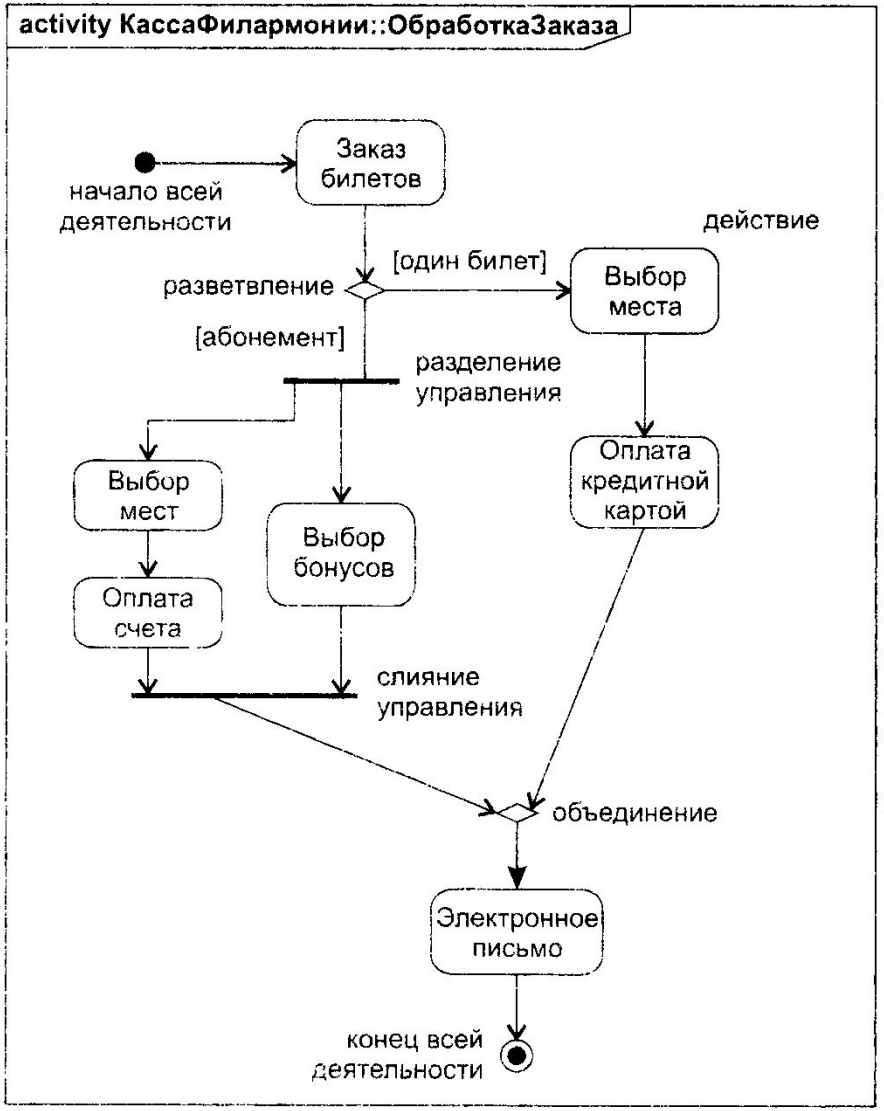
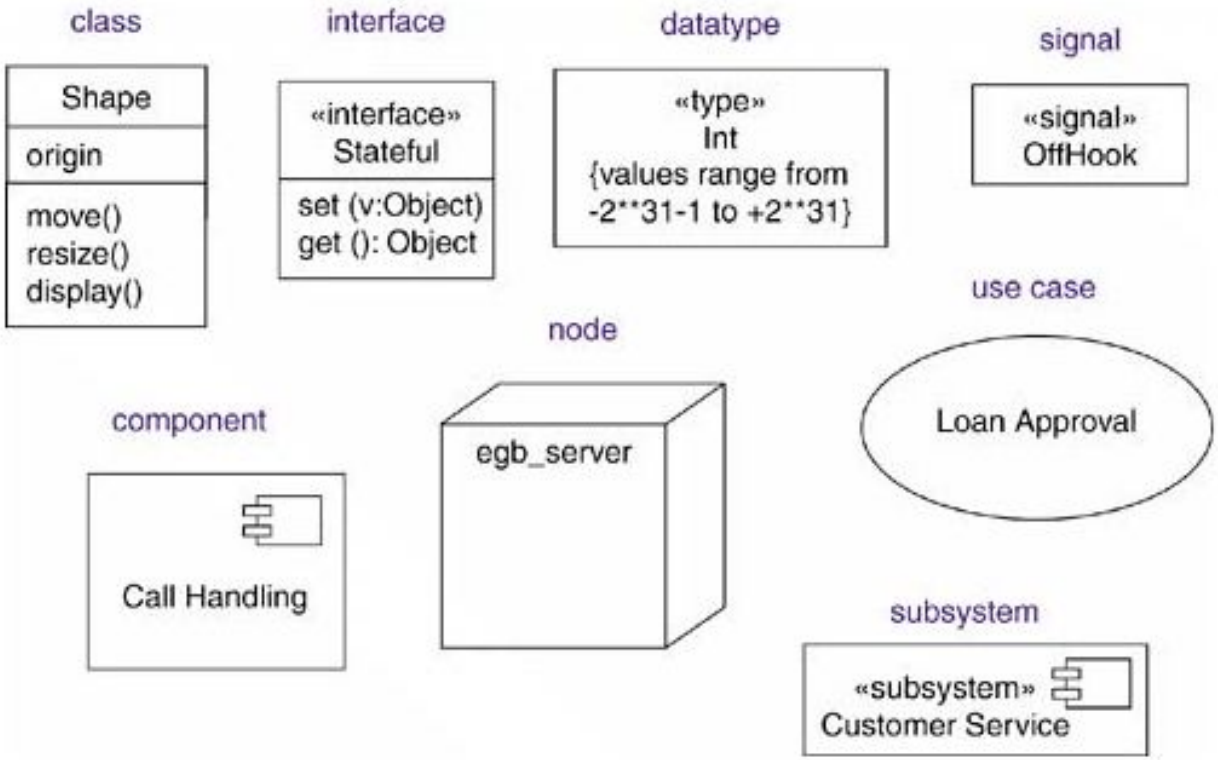


Figure 9-2. Classifiers

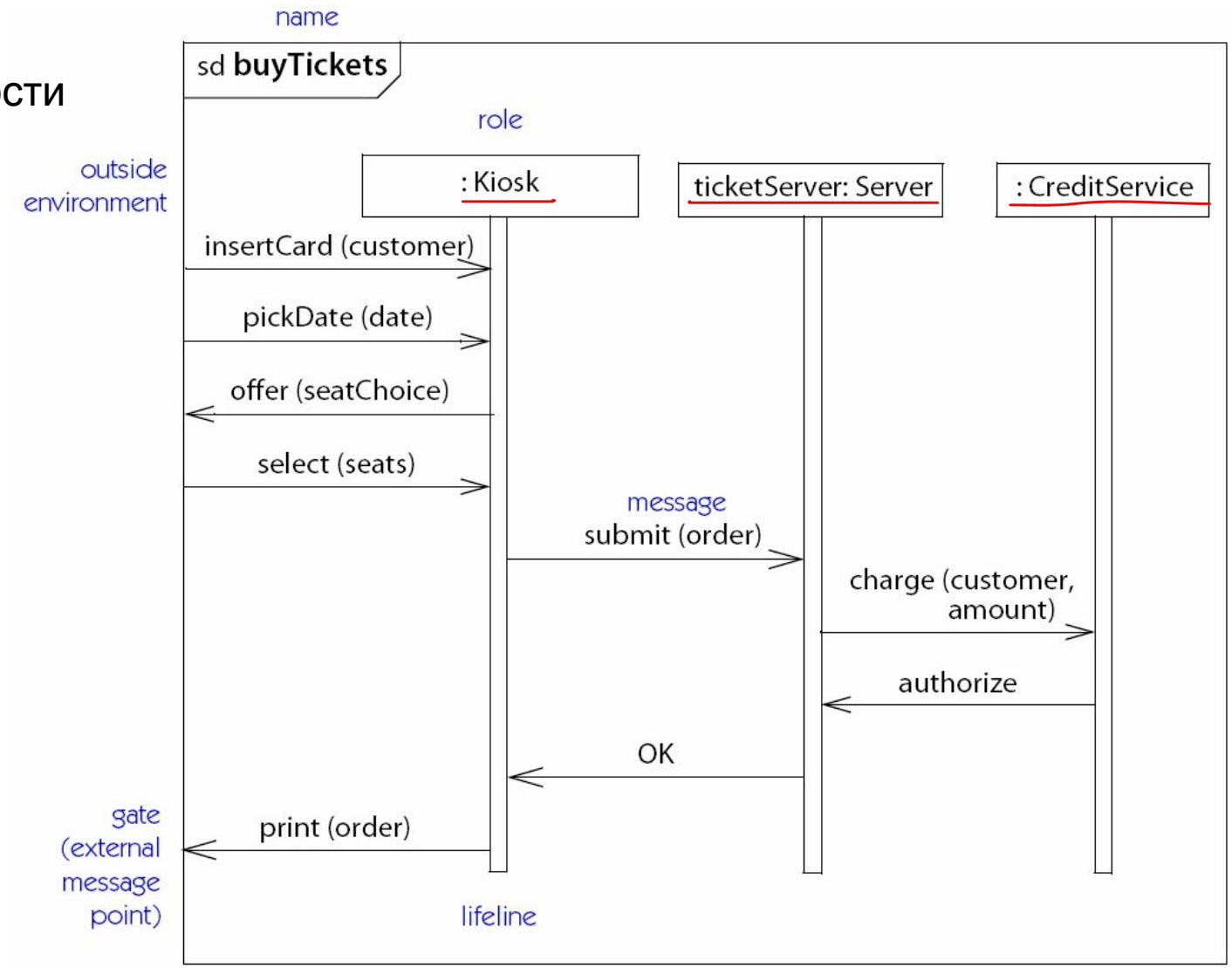
Описание взаимодействия

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.



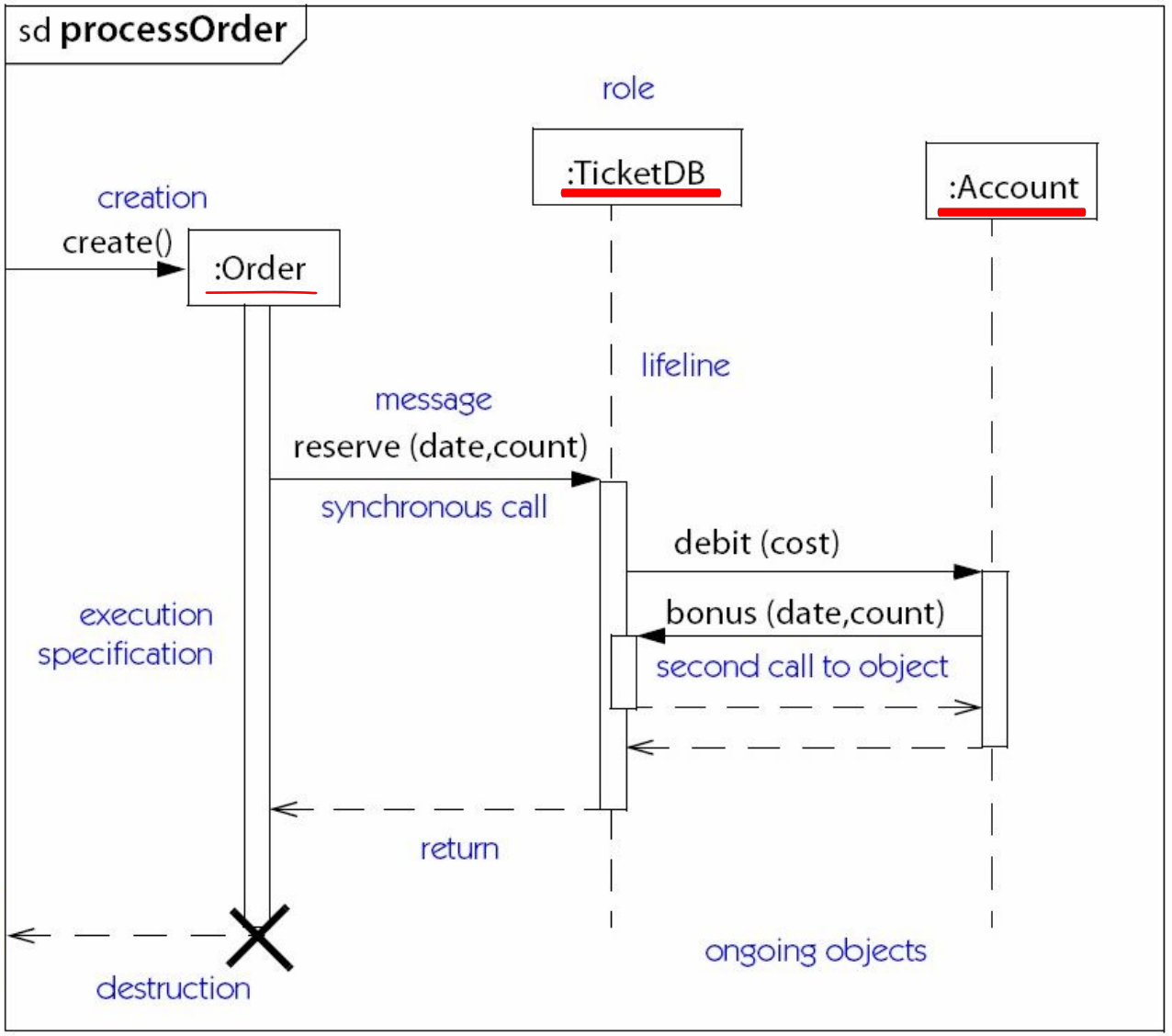
Описание взаимодействия

Диаграмма последовательности



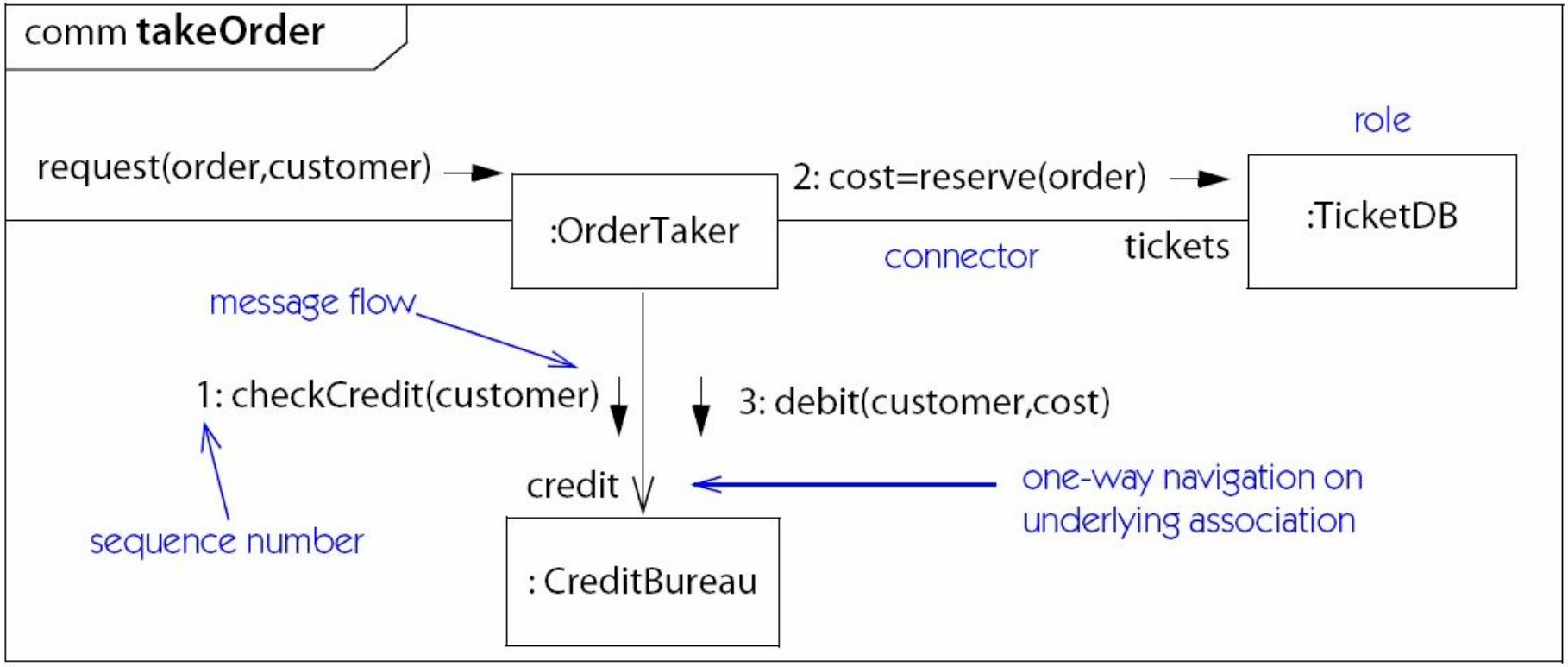
Описание взаимодействия

Диаграмма последовательности с детализацией выполнения



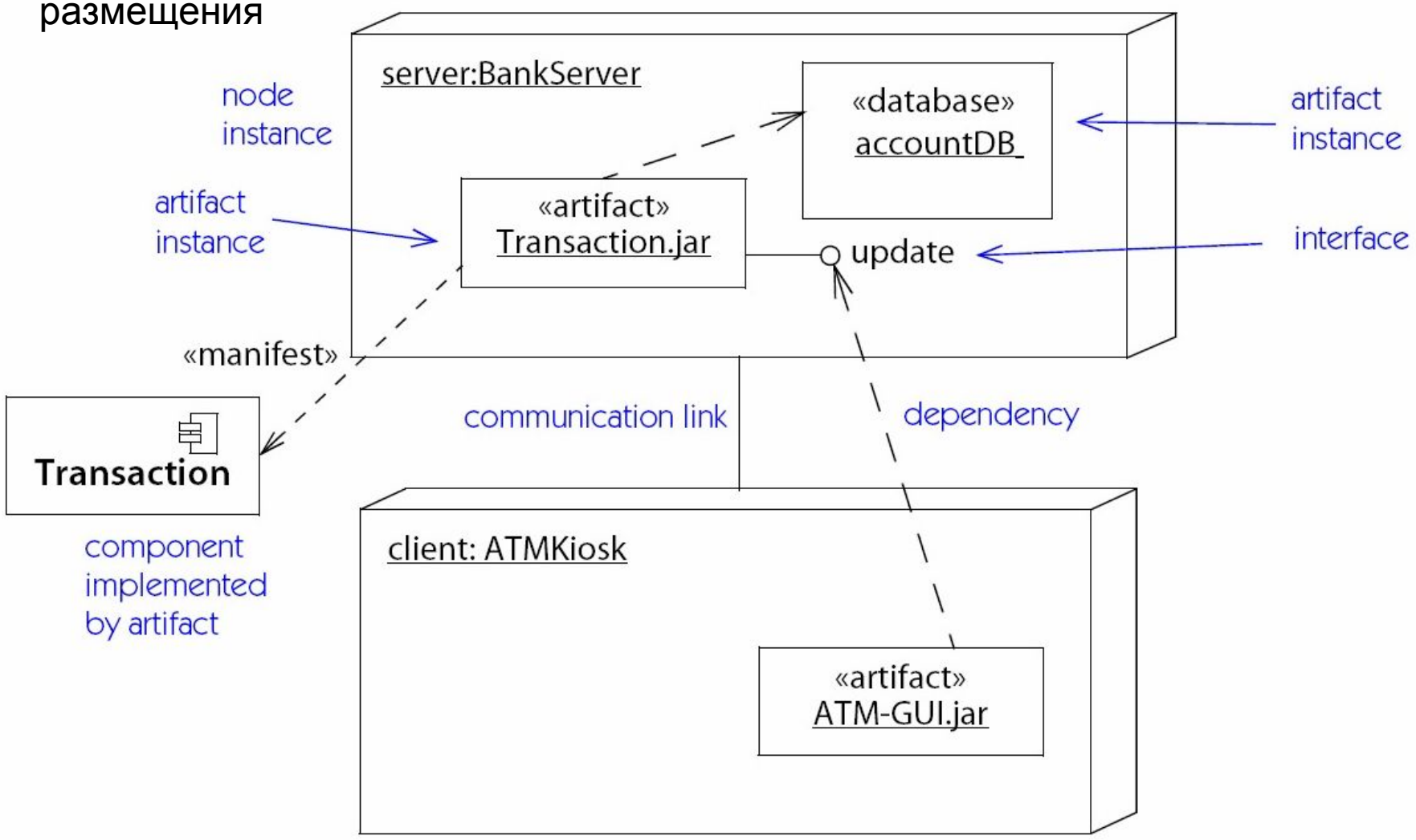
Описание взаимодействия

Коммуникационная диаграмма



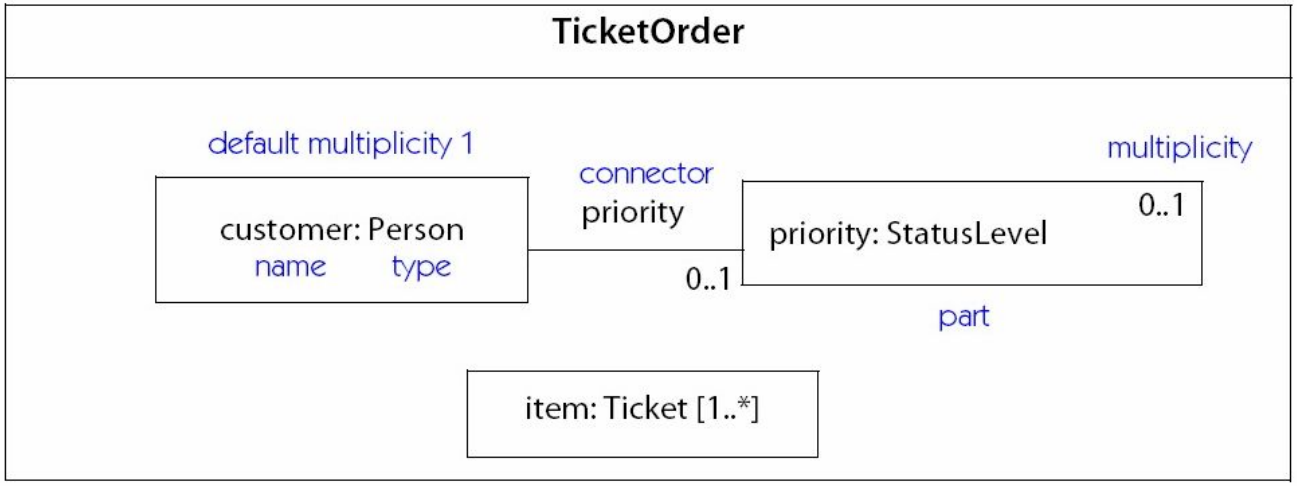
Описание размещения

Диаграмма размещения

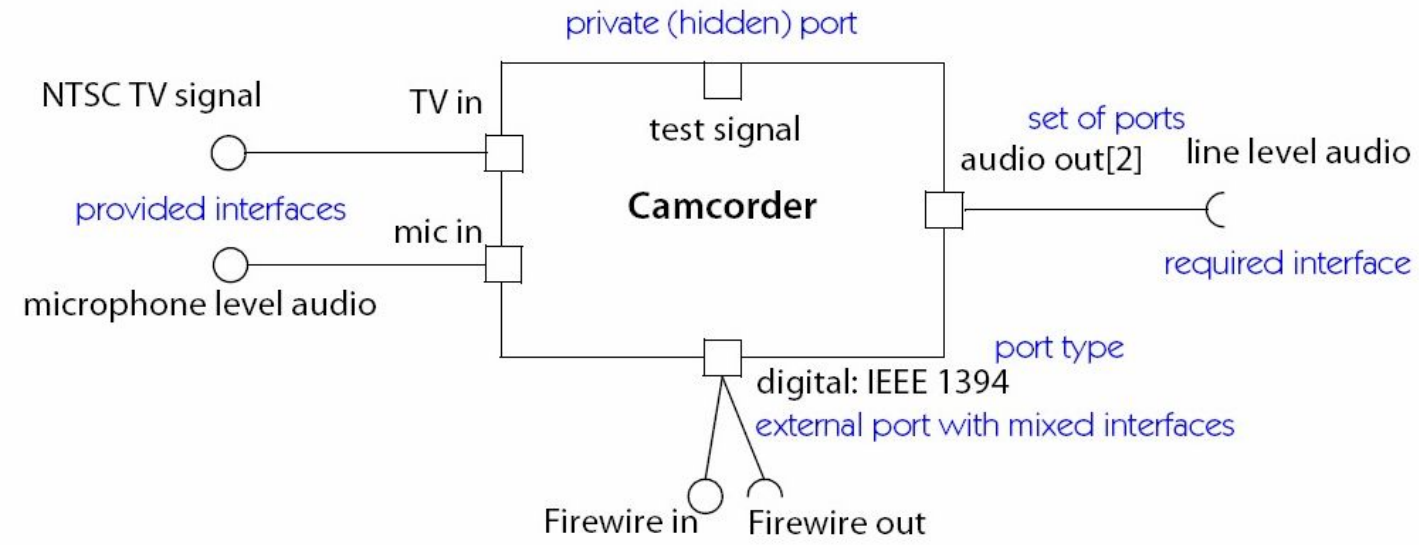


Описание проектных решений

Структурированный класс

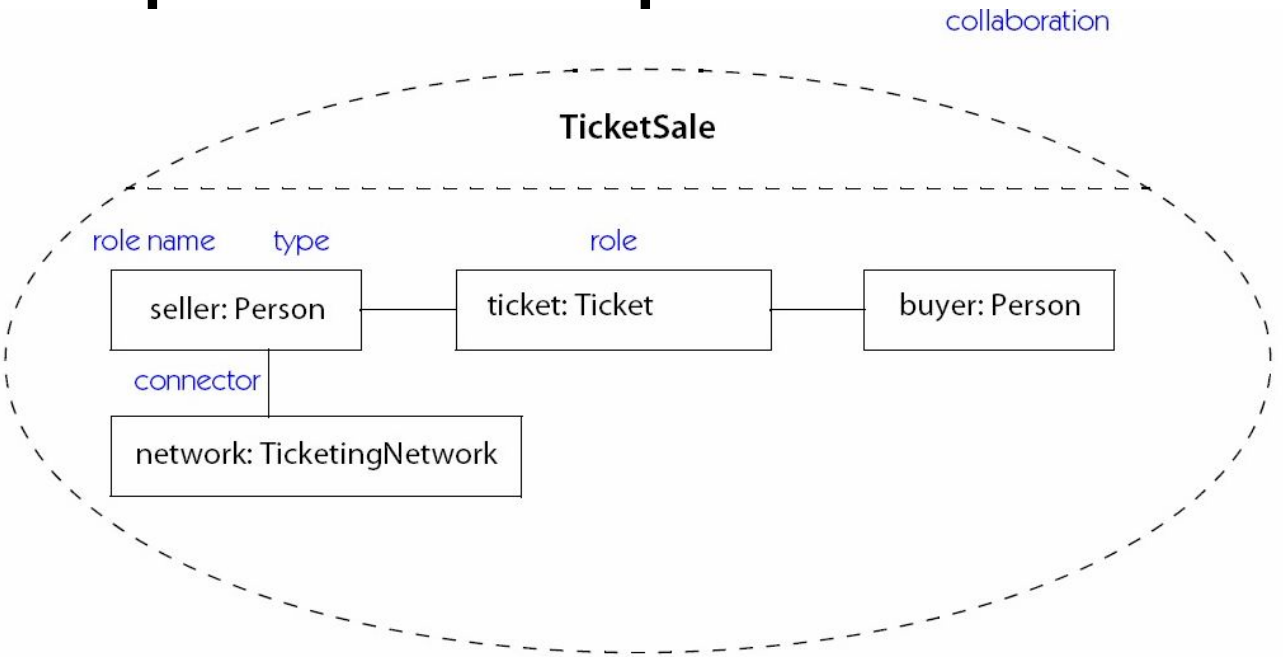


Структурированный класс с портами

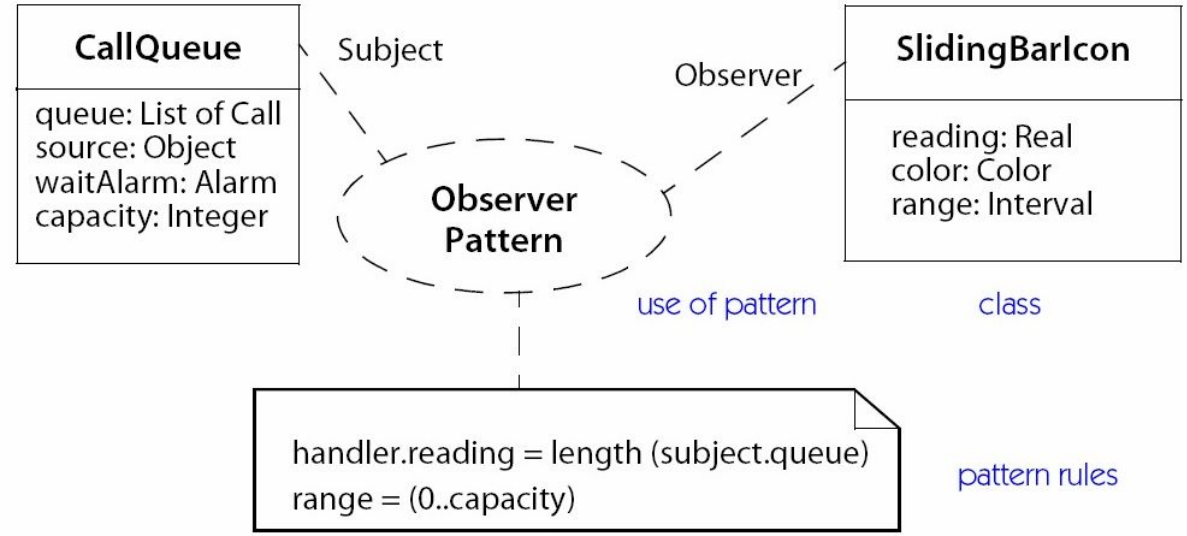


Описание проектных решений

Описание сотрудничества объектов

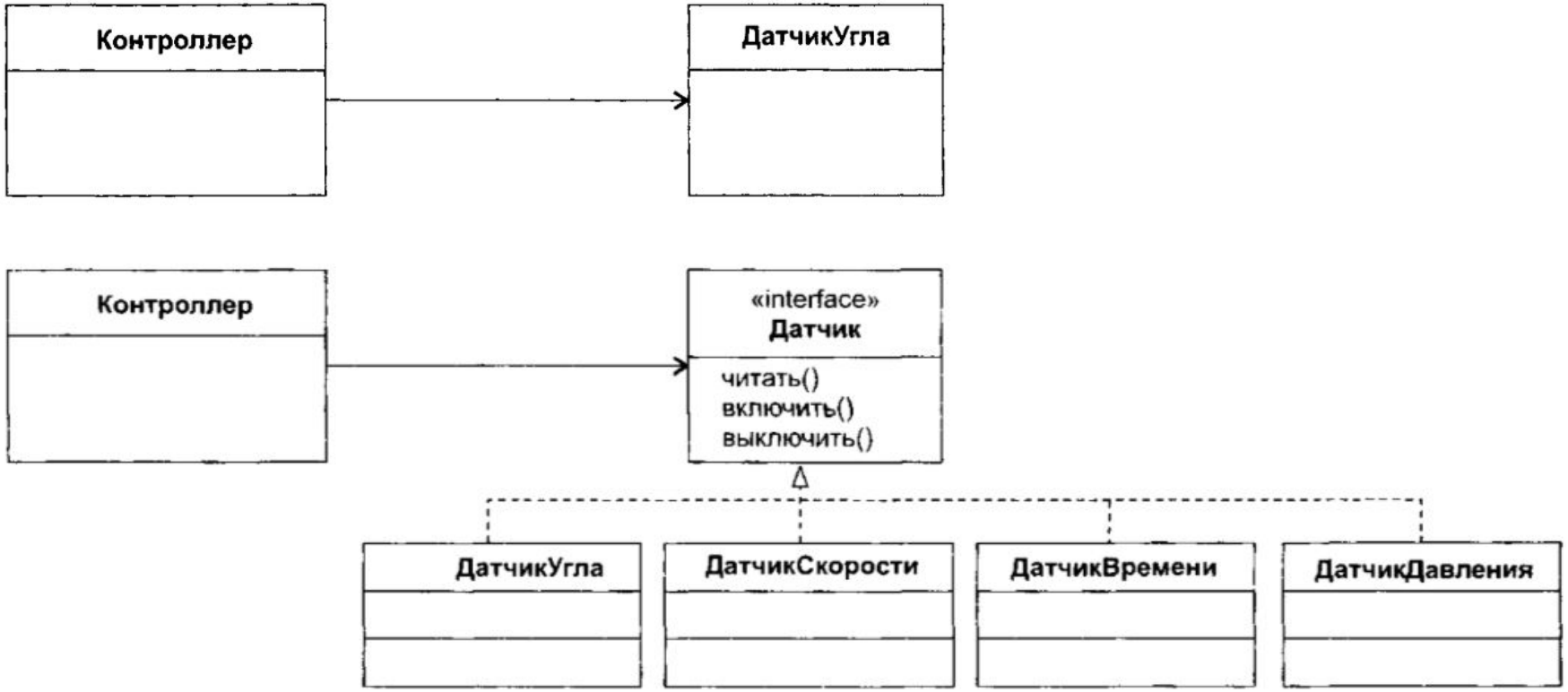


Использование шаблона



Основные принципы детального проектирования

Принцип открытия-закрытия Бертрана Мейера (OCP — The Open-Closed Principle)

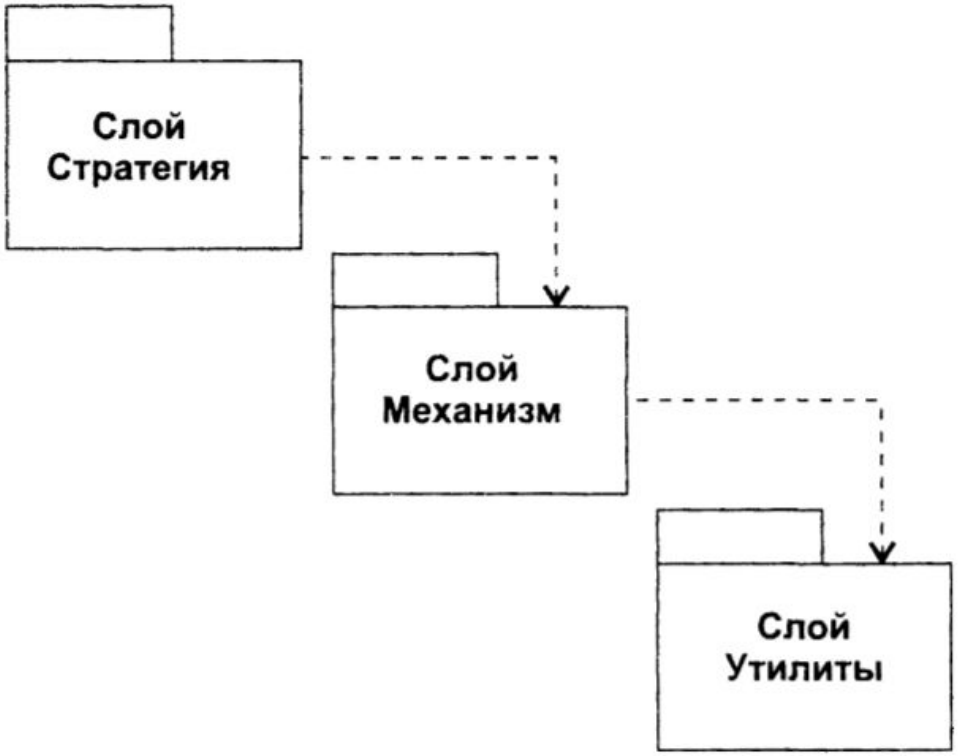


Основные принципы детального проектирования

- Принцип подстановки Барбары Лисков (LSP — Liskov Substitution Principle)
- Принцип инверсии зависимостей Роберта Мартина (DIP — Dependency Inversion Principle)
- Принцип отделения интерфейса (ISP — Interface Segregation Principle)

Основные принципы детального проектирования

- Принцип инверсии зависимостей Роберта Мартина (DIP — Dependency Inversion Principle)



Принципы упаковки классов в архитектурные подсистемы

- Принцип эквивалентности повторного применения (REP - Release Reuse Equivalency Principle). Уровень детализации повторного применения (reuse) должен соответствовать желаемому уровню детализации новой версии.
- Принцип общего закрытия (CCP — Common Closure Principle). Классы, входящие в состав пакета, должны быть закрыты по отношению к одним и тем же изменениям. Изменение, влияющее на пакет, оказывает воздействие на все классы этого пакета, не затрагивая другие пакеты.
- Принцип общего повторного применения (CRP — Common Reuse Principle). Классы, входящие в состав пакета, повторно применяются совместно. Если вы повторно применяете один из классов пакета, вы повторно применяете их все.

Документирование процесса проектирования

Стандарт IEEE Std 1016-2009 «Systems Design — Software Design Descriptions» предлагает документировать весь этап проектирования с помощью описания программного проектирования (SDD — software design description).

Стандарт определяет следующие точки зрения:

□ Контекстная точка зрения. Отражает услуги, оказываемые пользователям и другим заинтересованным лицам, взаимодействующим с системой. Система рассматривается как черный ящик.

□ Композитная точка зрения. Описывает систему как структуру, состоящую из нескольких частей, и определяет роль каждой части.

Документирование процесса проектирования

□ Логическая точка зрения. Выявляет существующие и проектируемые типы, а также их реализацию (классы и интерфейсы, их структурные статические отношения). Могут использоваться экземпляры типов.

□ Точка зрения зависимостей. Определяются отношения взаимодействия и доступа отдельных сущностей. Эти отношения описывают разделяемую информацию, порядок выполнения, параметры интерфейсов.

□ Информационная точка зрения. Точка зрения применима, если ожидается присутствие персистентных (устойчивых) данных.

Документирование процесса проектирования

□ Точка зрения с использованием паттернов. Рассматривается сотрудничество паттернов, их абстрактные роли и соединения.

□ Интерфейсная точка зрения. Обеспечивает проектировщиков, программистов и тестировщиков информацией о правильном использовании сервисов системы. Здесь описываются детали внешних и внутренних интерфейсов, не заданные в спецификации требований. Предлагается набор интерфейсных описаний для каждой сущности.

□ Структурная точка зрения. Используется для документирования внутренних составляющих частей и организации системы в терминах элементов.

Документирование процесса проектирования

□ Точка зрения взаимодействий. Определяет стратегии для взаимодействия сущностей, выделяя почему, где, как и зачем происходят действия. □ Точка зрения динамики состояний. Применима к событийно-управляемым системам. □ Алгоритмическая точка зрения. Детальное описание операций (методов, функций), внутренние детали и логика каждой сущности.

□ Ресурсная точка зрения. Моделируются характеристики и использование ресурсов системы

Документирование процесса проектирования

Оглавление SDD имеет следующий вид.

1. Аннотация
    - Дата создания и статус
    - Выпускающая организация
    - Авторство
    - Перечень изменений
2. Введение
    - Цель
    - Область действия
    - Контекст
    - Выводы
3. Ссылки
4. Словарь

5. Основная часть

Заинтересованные лица и проектные понятия

Проектная точка зрения 1

Проектное представление 1

…

Проектная точка зрения N

Проектное представление N

Обоснование проектирования.