Графические нотации, используемые при проектировании ПО

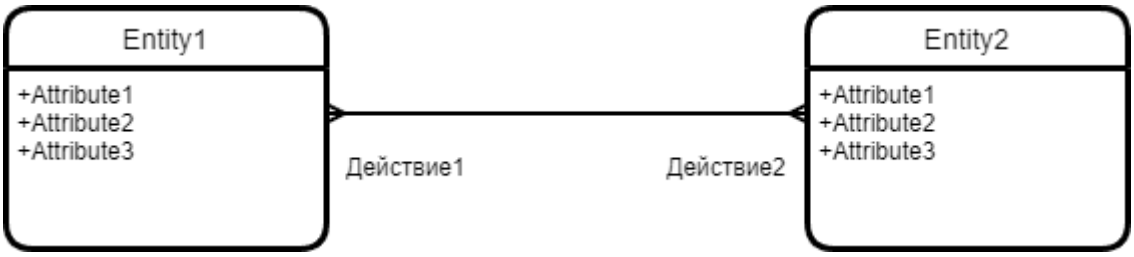
Содержание

ER-диаграммы

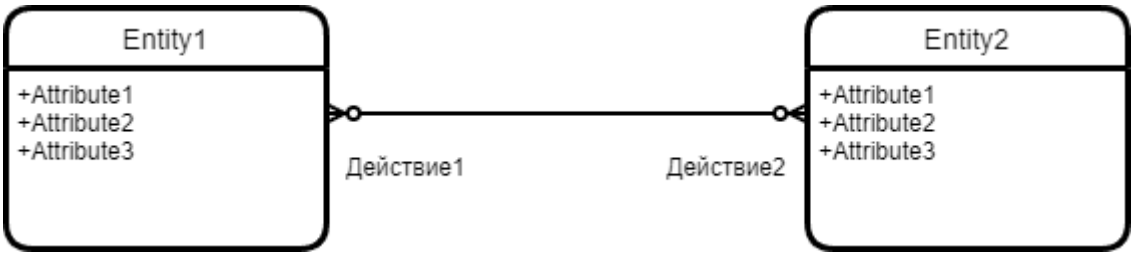
Обозначение:

1. Сущность - множество однотипных объектов предметной области, учавствующих в каких-то процессах в рамках предметной области. Т.е. сущность - это описание какого-то множества объектов. На диаграмме обозначается в виде прямоугольника с названием сущности внутри.
2. Связь - объект на диаграмме, обозначающий взаимодействие объектов сущностей. Связи бывают обязательные и необязательные; множественные и единичные. Связь имеет два конца с именами.
3. Атрибут - свойство объектов, описанных сущностями. Указываются внутри прямоугольника соответствующей сущности.

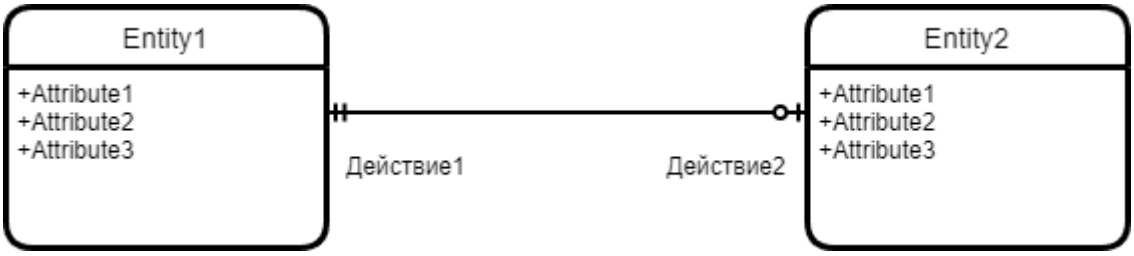
Примеры:



Каждый объект сущности Entity1 выполняет Действие1 как минимум с одним объектом сущности Entity2. Каждый объект сущности Entity2 выполняет Действие2 как минимум с одним объектом сущности Entity1.



Каждый объект сущности Entity1 может выполнять Действие1 с одним или более объектами сущности Entity2. Каждый объект сущности Entity2 может выполнять Действие2 с одним или более объектами сущности Entity1.



Каждый объект сущности Entity1 может выполнять Действие1 с одним объектом сущности Entity2. Каждый объект сущности Entity2 выполняет Действие2 с одним объектом сущности Entity1.

Диаграмма классов

Класс

Класс изображается в виде прямоугольника, разделённого на четыре секции. В верхней секции указаны имя класса и стереотип <>, если класс является интерфейсом. Названия абстрактных классов записываются курсивом; статических подчёркиваются. В секции ниже описаны атрибуты класса. В третьей секции сверху указаны методы класса. В последней секции указываются ограничения класса (инварианты). Если в классе отсутствует содержимое какой-то секции, то она всё равно очерчивается, но остаётся пустой.

Атрибуты

Атрибуты записываются в следующем формате:

```
<квантор видимости> <имя атрибута> [кратность] : <тип атрибута> = <исходное
значение> {строка-свойство}.
```

Кванторы видимости:

1. Символ '+' - public
2. Символ '#' - protected
3. Символ '-' - private
4. Символ '~' - package (атрибут виден везде, но только внутри пакета / сборки)

Кратность - мощность для атрибутов, представляющих множества. Можно указывать диапазон.

Чтобы указать, что атрибут является статическим, его определение выделяется знаком подчёркивания.

Символ '/' перед атрибутом указывает, что он может быть рассчитан на основе значений других атрибутов.

Строка-свойство указывает какие значения должны присваиваться атрибутам в момент создания объекта класса. Такие атрибуты не должны изменяться.

Методы

Методы класса описываются в следующем формате:

```
<квантор видимости> <имя операции>(список параметров): <выражение типа возвращаемого значения> {строка-свойство}
```

Параметры указываются так:

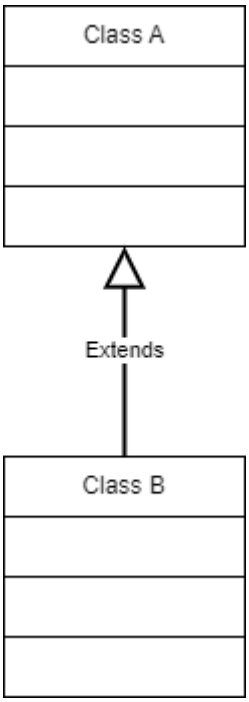
```
<направление параметра> <имя параметра>: <выражение типа> = <значение параметра по умолчанию>.
```

Направления параметров:

1. in - параметр для передачи значений.
2. out - параметр для возврата значений; во время вызова метода может быть неопределён.
3. inout - параметр для передачи и возврата значений.

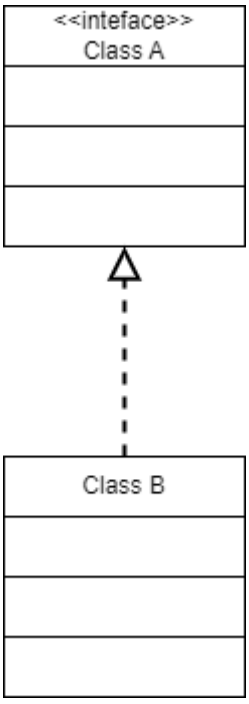
Связи между классами:

1. Наследование.



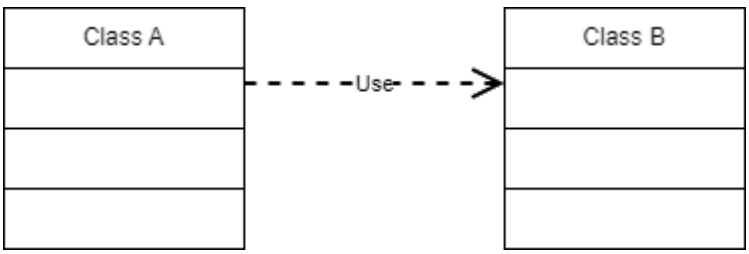
Класс "Class B" наследник класса "Class A".

2. Реализация.



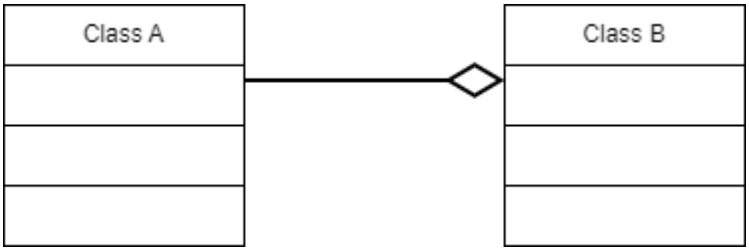
Класс "Class B" реализует интерфейс "Class A".

3. Зависимость.



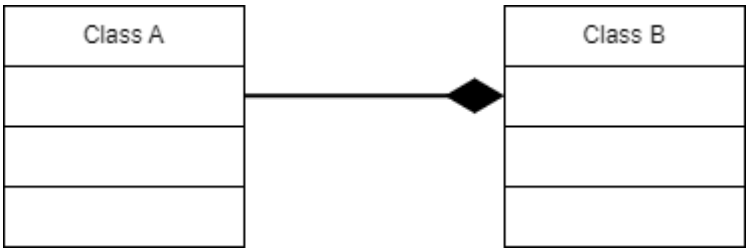
Класс "Class A" зависист от реализации класса "Class B" (например методы "Class A" принимают объекты типа "Class B" в качестве параметра).

4. Агрегирование.



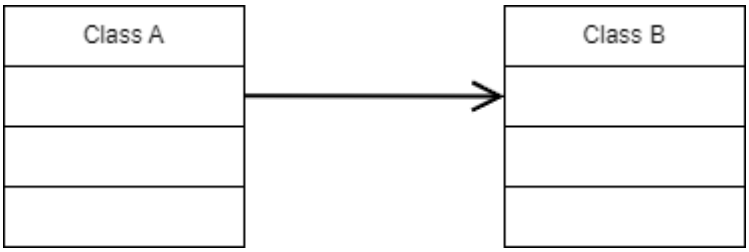
Класс "Class B" является контейнером для объектов класса "Class A". Время жизни содержимого не зависит от времени жизни контейнера.

5. Композиция.



Класс "Class B" является контейнером для объектов класса "Class A". Время жизни содержимого равное времени жизни контейнера.

6. Ассоциация.



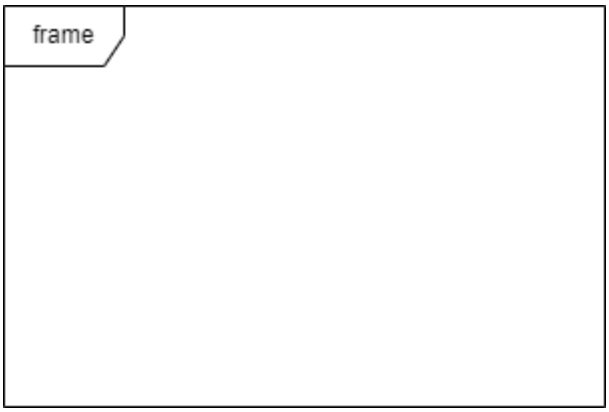
Класс "Class A" ассоциирован с классом "Class B" (может например вызывать методы класса "Class B");

Диаграмма последовательности

Диаграмма последовательности показывает взаимодействие объектов во времени.

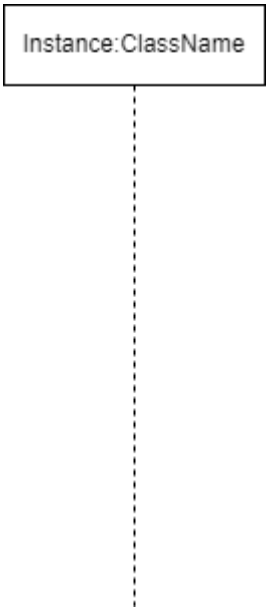
Frame

Фрейм - облать на которой описывается диаграмма поледовательности процесса. У фрейма есть имя, возможно входные параметры, если описываемый процесс подразумевает входные данные.



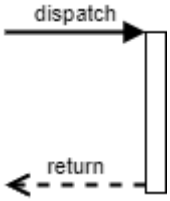
Lifeline

Линия жизни объекта показывает жизненный цикл объекта. Линия жизни идёт от названия объекта к которому она относится. Если объект во время жизненного цикла взаимодействует с другими объектами, то эти взаимодействия показаны стрелками идущими от / в линию жизни объекта.



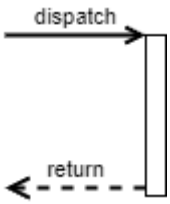
Синхронное взаимодействие

Синхронное взаимодействие подразумевает, что объект, отправивший сообщение другому объекту приостанавливает работу до получения ответа.

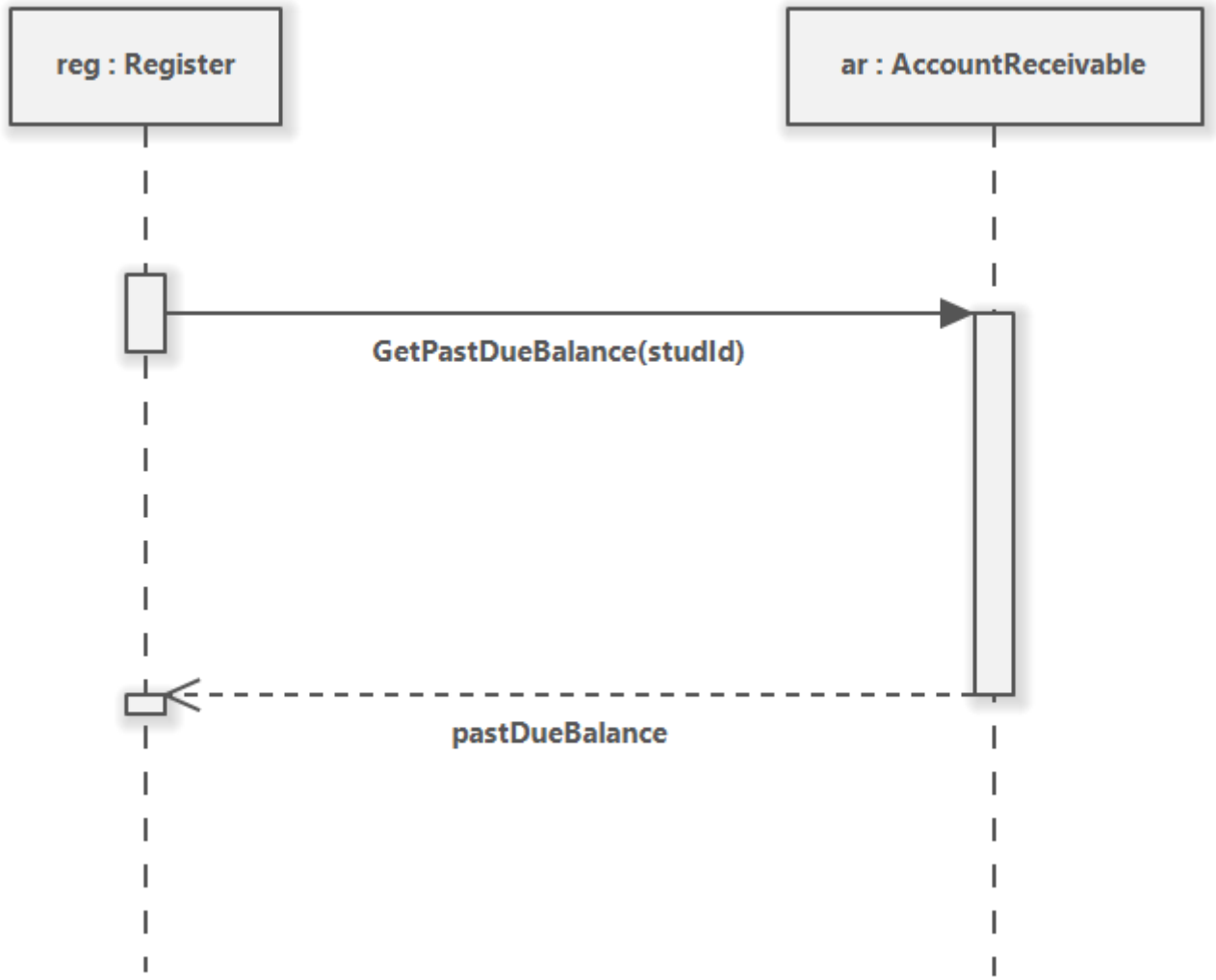


Асинхронное взаимодействие

Асинхронное взаимодействие подразумевает, что после отправки сообщения другому объекту, текущий объект продолжает работу и не ждёт пока придёт ответ на запрос.



Пример диаграммы:



Исходный код:

```
internal class AccountReceivable
{
    List<Student> students;
    public AccountReceivable()
    {
        students = DataBaseStud.Students;
    }

    public int GetPastDueBalance(int studid)
    {
```

```
        return students.Where(x => x.Id == studid).Select(x =>
x.PastDueBalance).First();
    }
}

internal class Register
{
    AccountReceivable ar;
    public Register()
    {
        ar = new AccountReceivable();
    }

    public void AddStudentInClass(int studId)
    {
        int pastDueBalance = ar.GetPastDueBalance(studId);
    }
}
```

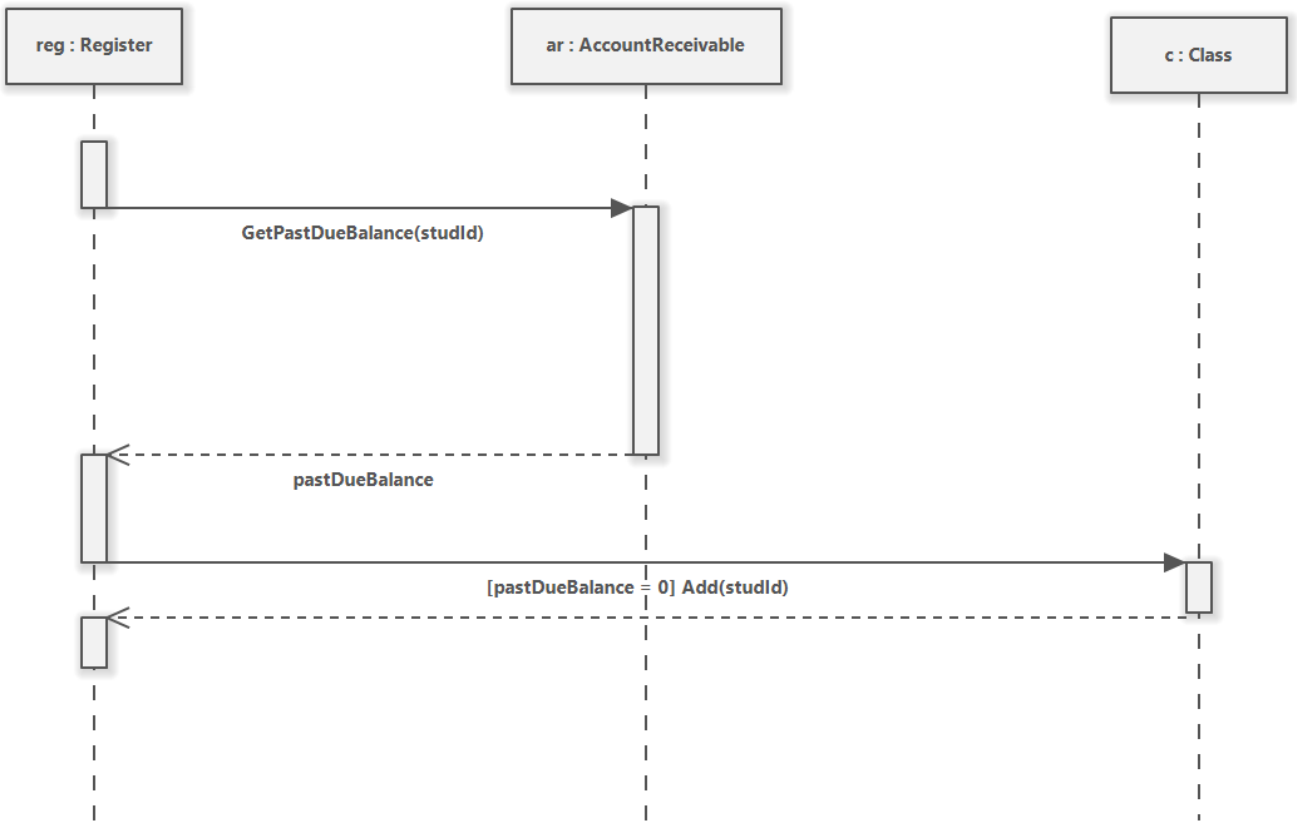
Guards

Охранные выражения - условия которые должны выполняться для отправки сообщения.

Исходный код:

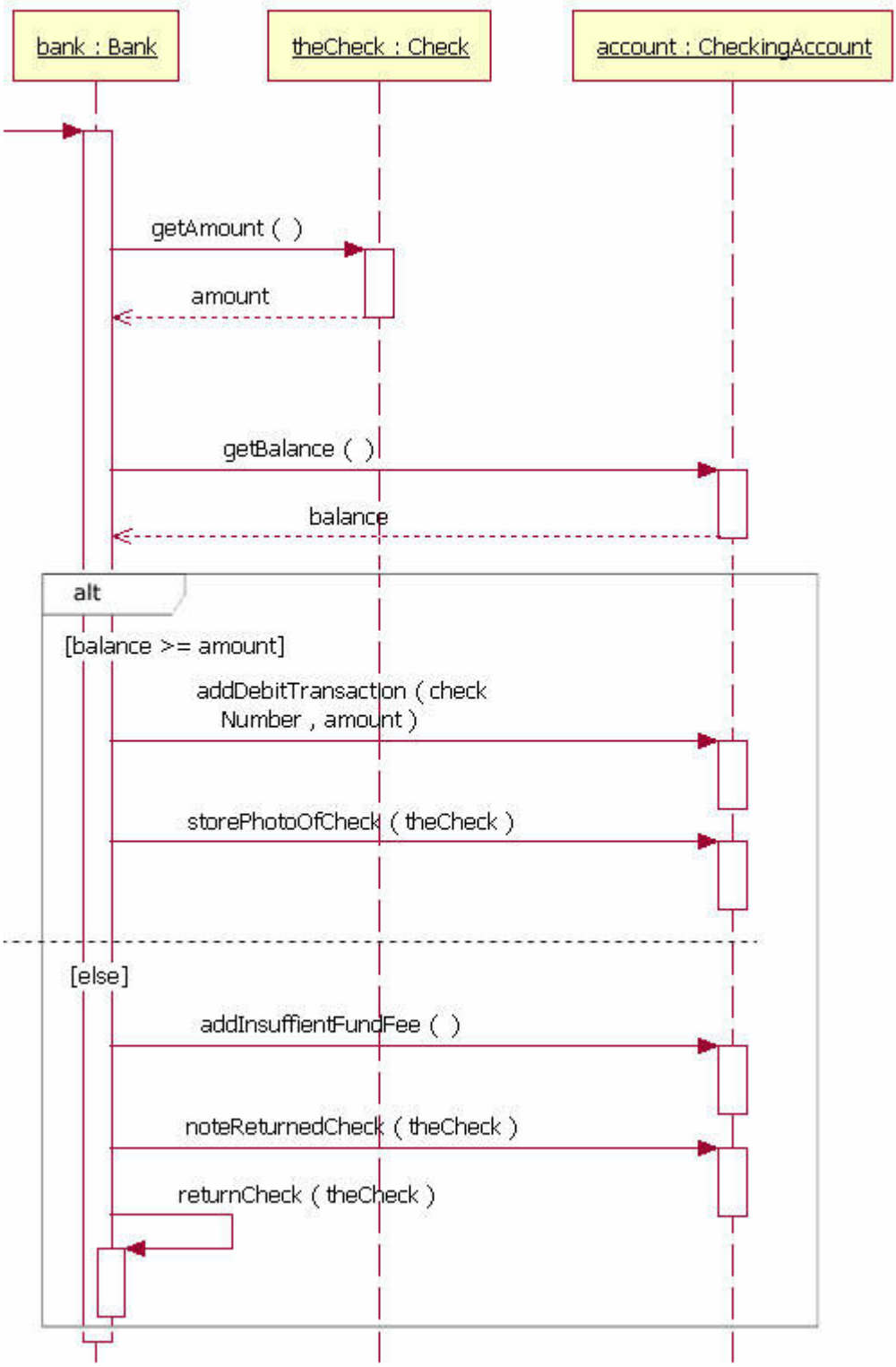
```
internal class AccountReceivable
{
    List<Student> students;
    public AccountReceivable()
    {
        students = DataBaseStud.Students;
    }

    public int GetPastDueBalance(int studid)
    {
        return students.Where(x => x.Id == studid).Select(x =>
x.PastDueBalance).First();
    }
}

namespace ReactiveConsoleApp
{
    internal class Class
    {
        public List<int> StudId { get; set; }
    }
}

internal class Register
```

```
{
    AccountReceivable ar;
    public Register()
    {
        ar = new AccountReceivable();
    }

    public void AddStudentInClass(int studId, Class c)
    {
        int pastDueBalance = ar.GetPastDueBalance(studId);

        if(pastDueBalance == 0)
        {
            c.StudId.Add(studId);
        }
    }
}
```



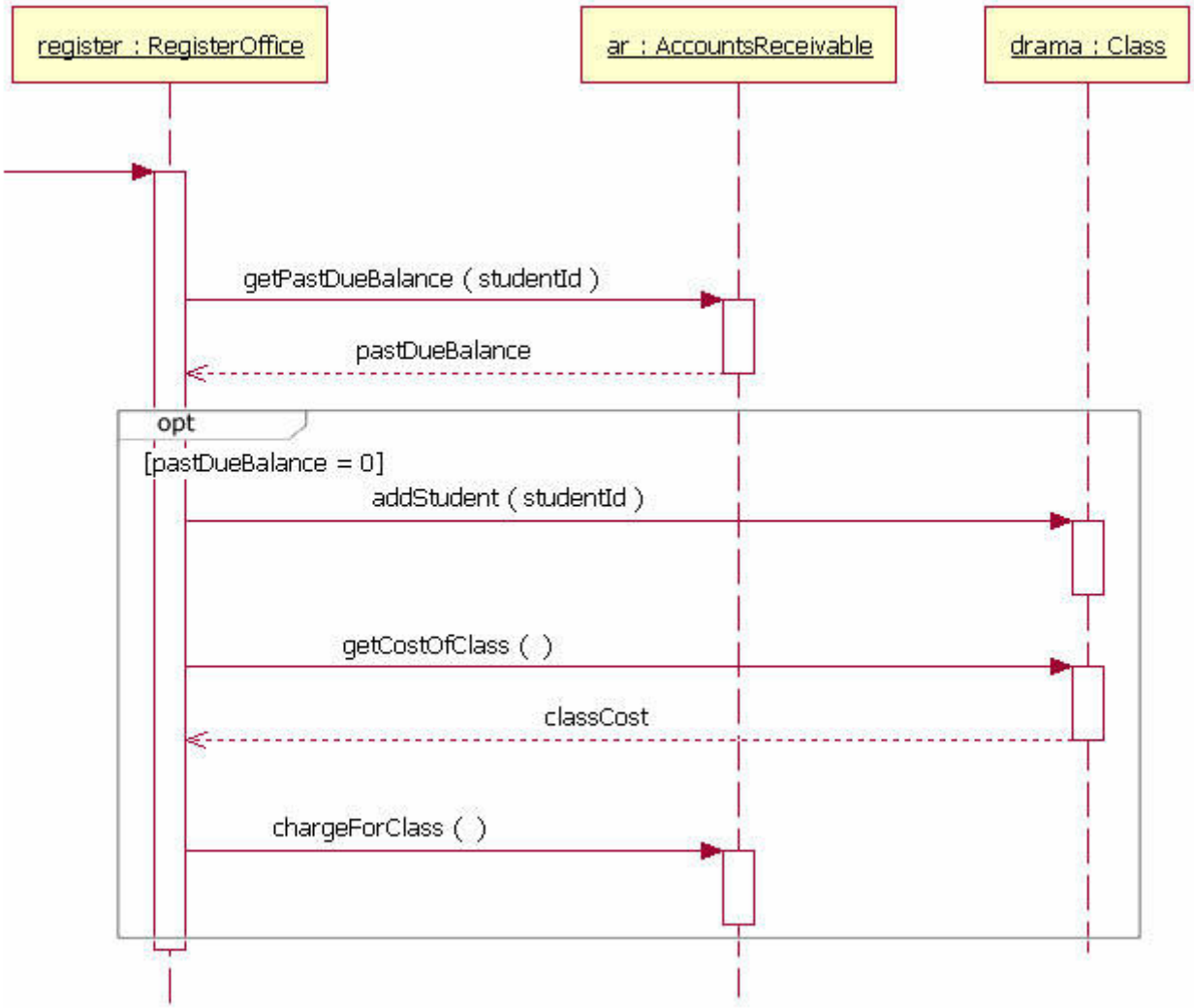
Alternatives

Альтарнативы - условное выражение вида "Если - то - иначе".



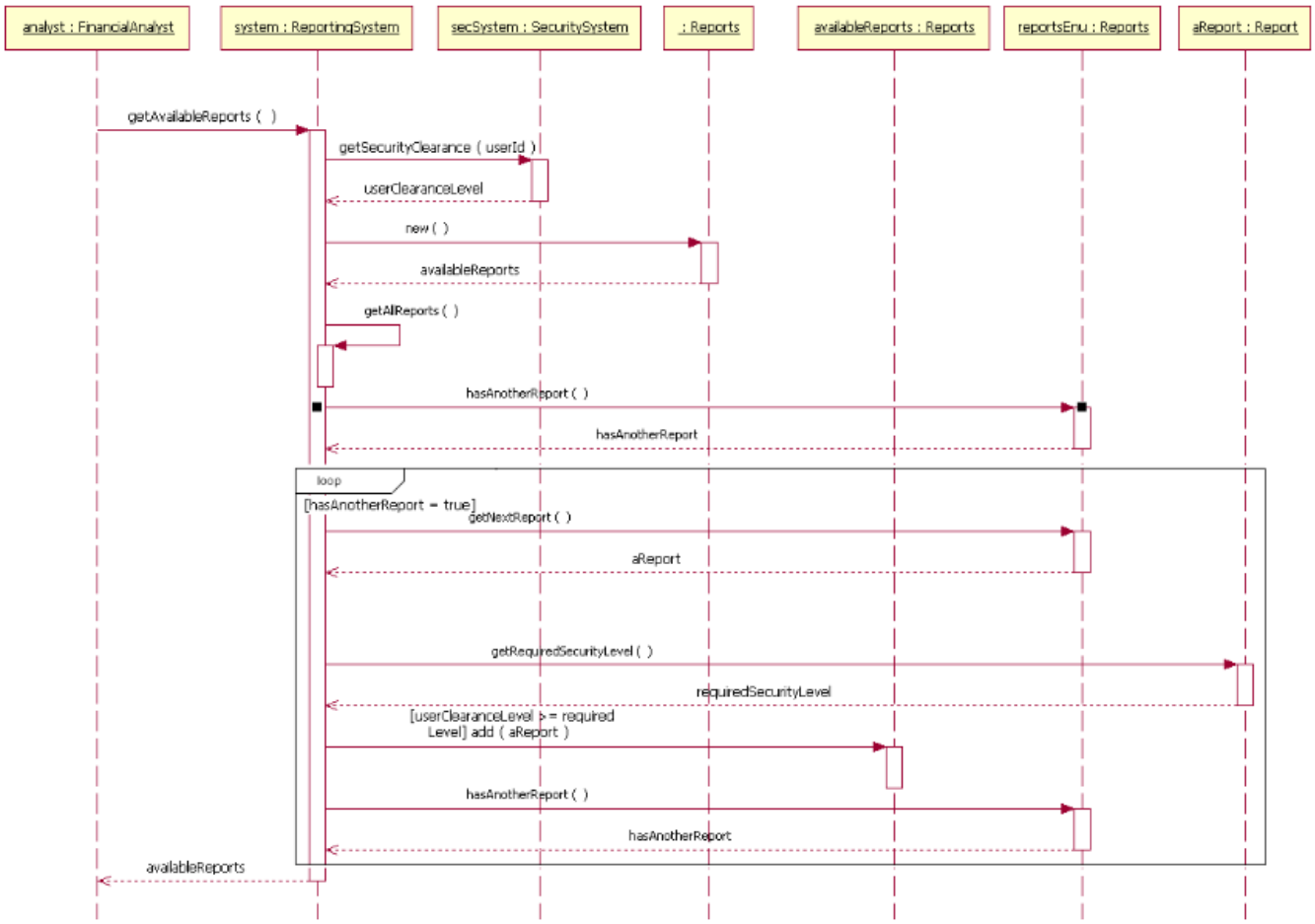
Option

Опционально исполняемый блок - участок диаграммы, исполняемый в случае истинности заданного условного выражения.



Loops

Циклы - участки диаграммы, выполняющиеся до тех пок пока условие истинно.



Также можно ссылаться на другие диаграммы последовательности, подставляя их в линии жизни объектов.

Диаграмма прецедентов (use case)

Акторы

Акторы - сущности использующие систему. Обозначаются в виде схематичных человечков, но не обязательно являются людьми.

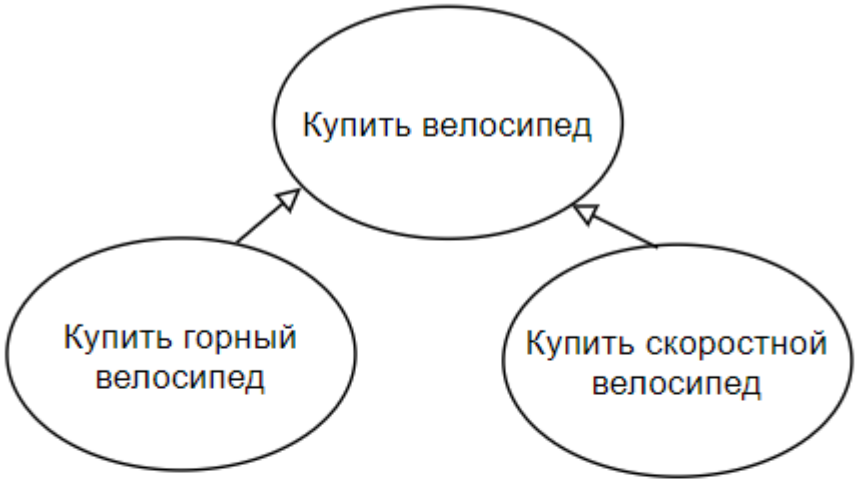
Прецеденты (use case, действия)

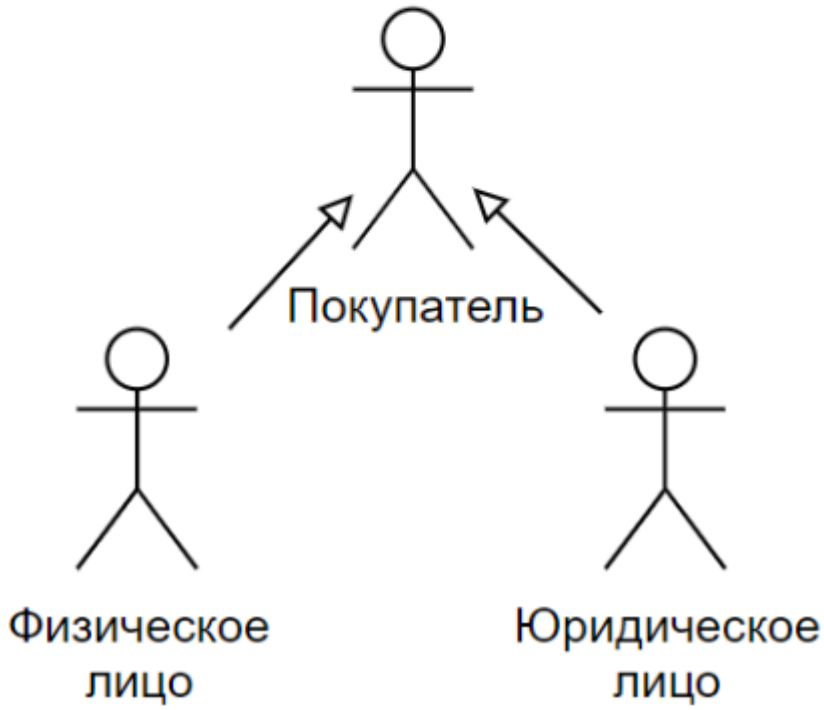
Действия изображаются в виде эллипса внутри которого - название действия.

Связи

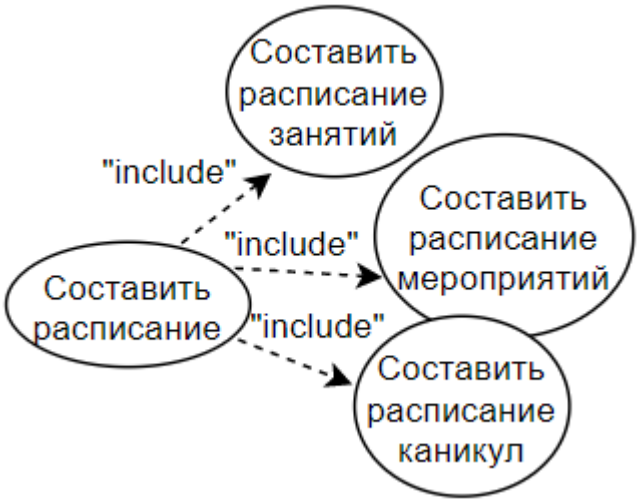
1. Ассоциация. Показывает какой тип актора какие действия с системой может выполнить. Ассоциация изображается ненаправленной линией.

2. Обобщение. Обобщать можно как акторов так и прецеденты. Изображается в виде пустой стрелки, направленной к обобщению от частного.





3. Включение(include). Показывает, что прецедент включает в себя другие прецеденты как составные части.



4. Расширение(extend). Отношение показывающее выборочное включение. Прецедент может включать в себя другие прецеденты как составные части, например по желанию пользователя, или в зависимости от сценария.