Отношения между классами - UML и код

Оглавление

Введение

Диаграмма классов UML позволяет обозначать отношения между классами, которые, в отличии от наследования, не имеют явной реализации в языках программирования. Для чего они нужны? Они нужны для максимально приближенного моделирования прикладной области, для которой создается программный продукт. Но как они отражаются в программном коде? Данное небольшое исследование пытается ответить на этот вопрос и показать, как в коде описываются связи между классами.

Сначала попробуем прояснить, как относятся друг к другу отношения между классами в UML. Используя различные источники удалось построить следующую структурную схему, демонстрирующую разновидности отношений:

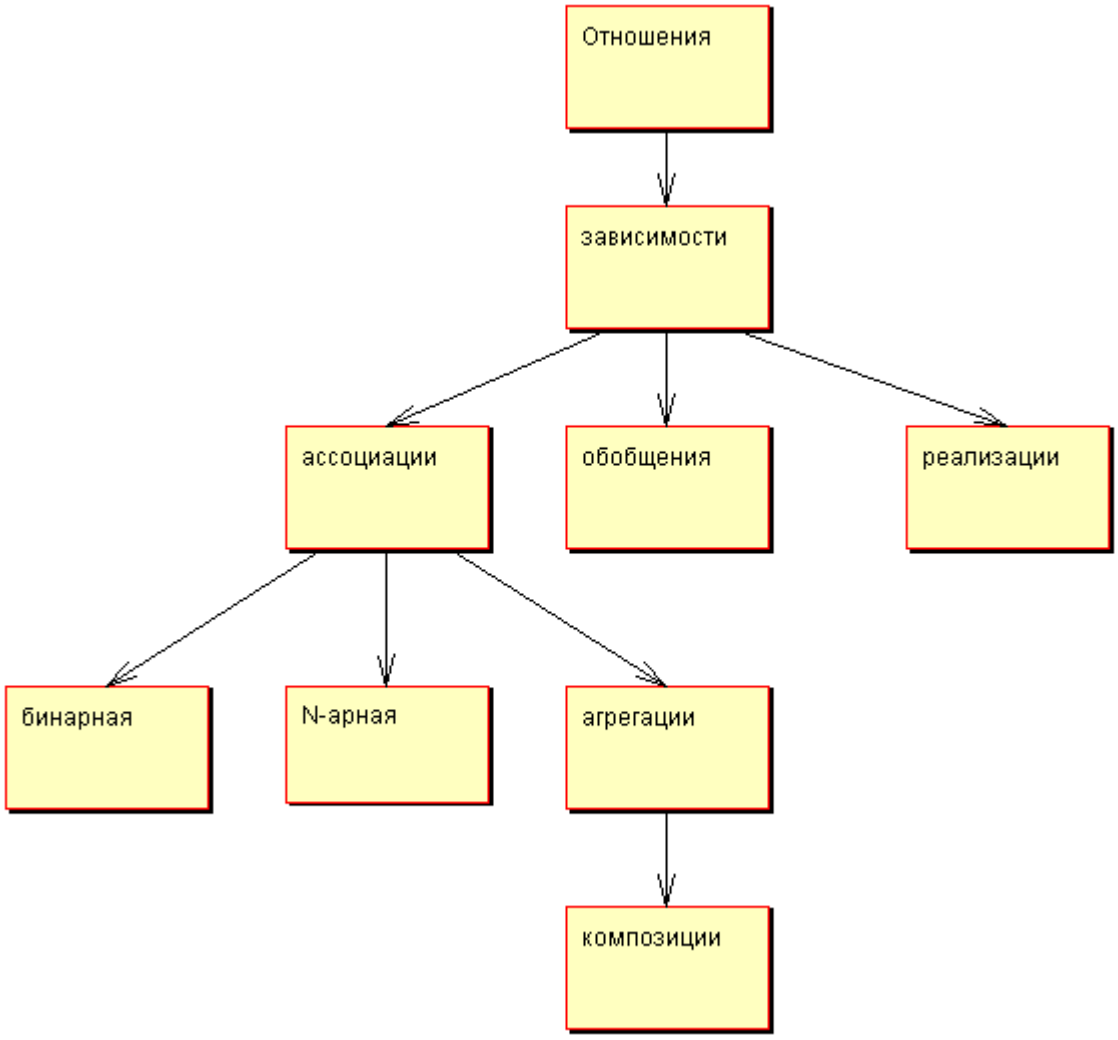

Рис. 1 - Отношения между классами

В качестве прикладной области возьмем отдел кадров некого предприятия и наченем строить его модель. Для примеров будем использовать язык Java.

1. Обобщение

Отношение обобщения - это наследование. Это отношение хорошо рассматривается в каждом учебнике какому-либо ООП языку.
В языке Java имеет явную реализацию через расширение(extends) одного класса другим.

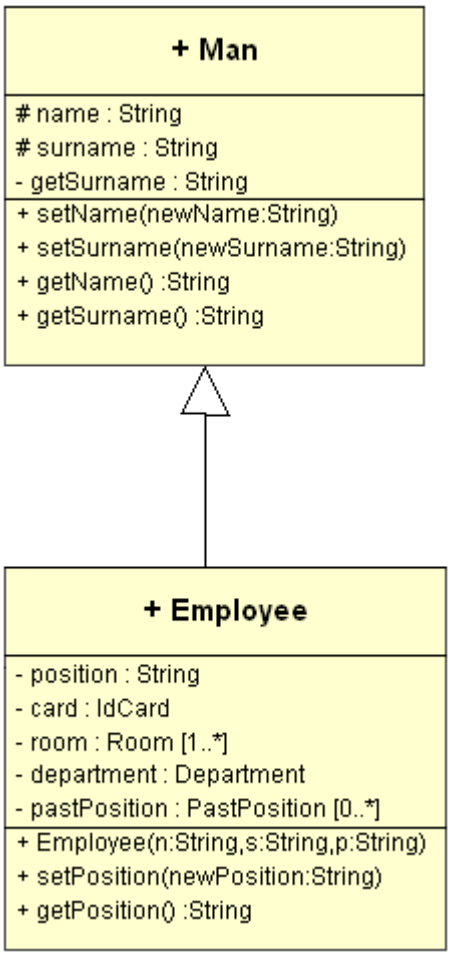

Рис. 2 - Отношение обобщения

Класс "Man"(человек) - более абстрактный, а "Employee"(сотрудник) более специализированный. Класс "Employee" наследует свойства и методы "Man".

Попробуем написать код для этой диаграммы:
[code]

```
public class generalization
{
	public static class Man
	{
		protected String name;
		protected String surname;

		public void setName(String newName)
		{
			name = newName;
		}
```

```
		public String getName()
		{
			return name;

		}

		public void setSurname(String newSurname)
		{
			name = newSurname;
		}

		public String getSurname()
		{
			return surname;

		}
	}

	// наследуем класс Man
	public static class Employee extends Man
	{
		private String position;

		public  Employee(String n, String s, String p)
		{
			name = n;
			surname = s;
			position = p;
		}

		public void setPosition(String newProfession)
		{
				position = newProfession;
		}

		public String getPosition()
		{
				return position;
		}
	}


	public static void main(String[] args)
	{
		Employee sysEngineer = new Employee("Жора", "Кустов", "Управделами");

	}
}
[/code]
```

2. Ассоциация

2.1 Бинарная

В модель добавили класс "IdCard", представляющий идентификационную карточку(пропуск) сотрудника. Каждому сотруднику

может соответствовать только одна идентификационная карточка, мощность связи 1 к 1.

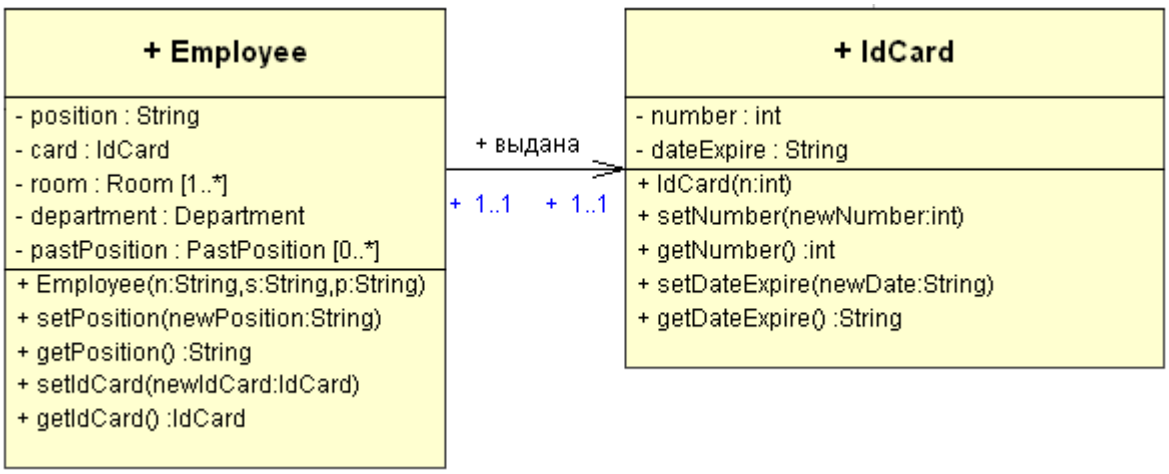

Рис. 3 - Бинарная ассоциация

Классы:

```
[code]
public static class Employee extends Man
{
	private String position;
	private IdCard iCard;
	public  Employee(String n, String s, String p)
	{
		name = n;
		surname = s;
		position = p;
	}

	public void setPosition(String newPosition)
	{
			position = newPosition;
	}

	public String getPosition()
	{
		return position;
	}
	public void setIdCard(IdCard c)
	{
		iCard = c;
	}
	public IdCard getIdCard()
	{
		return iCard;
	}

}

public static class IdCard
{
	private String dateExpire;
	private int number;
	public IdCard(int n)
```

```
	{
		number = n;
	}
	public void setNumber(int newNumber)
	{
		number = newNumber;
	}
	public int getNumber()
	{
		return number;
	}
	public void setDateExpire(String newDateExpire)
	{
		dateExpire = newDateExpire;
	}
	public String getDateExpire()
	{
		return dateExpire;
	}
}
```
[/code]

В теле программы создаем объекты и связываем их:
[code]
```
IdCard card = new IdCard(123);
card.setDateExpire("2010-12-01");
sysEngineer.setIdCard(card);

System.out.println(sysEngineer.getName() +" работает в должности "+
sysEngineer.getPosition());
System.out.println("Удостовирение действует до " + sysEngineer.getIdCard().dateExpire);
```
[/code]

Класс Employee имеет поле card, у которого тип IdCard, так же класс имеет методы для присваивания значения(setIdCard) этому полю и для
получения значения(getIdCard). Из экземпляра объекта Employee мы можем узнать о связанном с ним объектом типа IdCard, значит
навигация (стрелочка на линии) направлена от Employee к IdCard.

2.2 N-арная ассоциация

Представим, что в организации положено закреплять за работниками помещения.
Добавляем новый класс Room.
Каждому объекты работник(Employee) может соответстовать несколько рабочих помещений. Мощность связи один-ко-многим.
Навигация от Employee к Room.

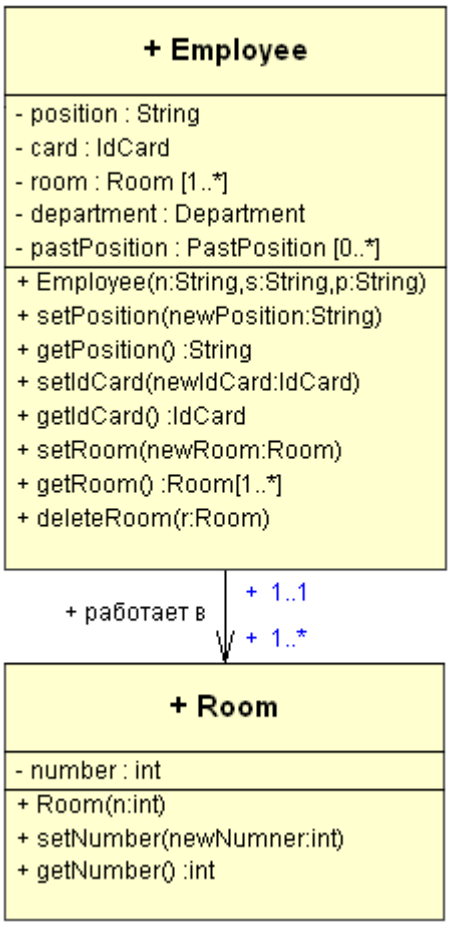

Рис. 4 - N-арная ассоциация

Теперь попробуем отразить это в коде. Новый класс Room:

[code]

```
public static class Room
{
	private int number;

	public Room(int n)
	{
		number = n;
	}

	public void setNumber(int newNumber)
	{
		number = newNumber;
	}

	public int getNumber()
	{
		return number;
	}
}
```

[/code]

Добавили в класс Employee поле и методы для работы с Room:

[code]
```
	...
	private Set room = new HashSet();
	...
	public void setRoom(Room newRoom)
	{
		room.add(newRoom);
	}

	public Set  getRoom()
	{
		return room;
	}

	public void  deleteRoom(Room r)
	{
		room.remove(r);
	}
	...
```
[/code]

Пример использвания:

[code]
```
	public static void main(String[] args)
	{

		Employee sysEngineer = new Employee("John", "Connor", "Manager");
		IdCard card = new IdCard(123);
		card.setDateExpire("2010-12-01");
		sysEngineer.setIdCard(card);

		Room room101 = new Room(101);
		Room room321 = new Room(321);
		sysEngineer.setRoom(room101);
		sysEngineer.setRoom(room321);

		System.out.println(sysEngineer.getName() +" работает в должности "+
sysEngineer.getPosition());
		System.out.println("Удостовирение действует до " +
sysEngineer.getIdCard().dateExpire);

		System.out.println("Может находиться в помещеньях:");
		Iterator iter =  sysEngineer.getRoom().iterator();
		while(iter.hasNext())
			System.out.println( ((Room) iter.next()).getNumber());
	}
```
[/code]

2.3 Агрегация

Введем в модель класс Departmen(отдел), то есть наше предприятие структурировано по отделам. В каждом отделе может работать один или более человек. Можно сказать, что отдел включает в себя одного или более сотрудников и таким образом их

агрегирует. На предприятии могут быть сотрудники, которые не пренадлежат ни одному отделу, например директор предприятия.

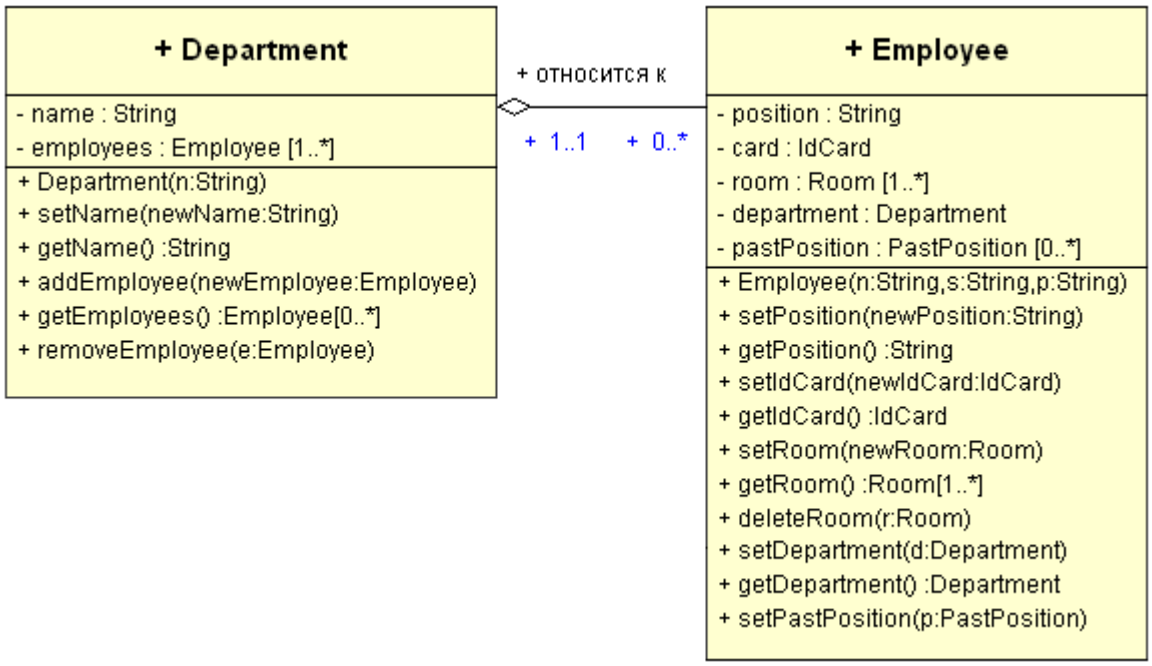

Рис. 5 - Агрегация

Класс Department:
[code]

```
public static class Department
	{
		private String name;
		private Set employees = new HashSet();

		public Department(String n)
		{
			name = n;
		}

		public void setName(String newName)
		{
			name = newName;
		}

		public String getName()
		{
			return name;
		}

		public void  addEmployee(Employee newEmployee)
		{
			employees.add(newEmployee);
			// связываем сотрудника с этим отделом
			newEmployee.setDepartment(this);
		}

		public Set  getEmployees()
```

```
		{
			return employees;
		}
		public void  removeEmployee(Employee e)
		{
			employees.remove(e);
		}
	}
```

Итак, наш класс, помимо конструтора и метода изменения имени отдела, имеет методы для занесения в отдел нового сотрудника, для удаления сотрудника и для получения всех сотрудников входящих в данный отдел. Навигация на диаграмме не показана, значит она является двунаправленно: от объекта типа "Department" можно узнать о сотруднике и от объекта типа "Employee" можно узнать к какому отделу он относится. Так как нам нужно легко узнавать какому отделу относится какой-либо сотрудник, то добавим в класс Employee поле и методы для назаначения и получения отдела.

```
...
private Department department;
...
public void setDepartment(Department d)
{
	department = d;
}

public Department getDepartment()
{
	return department;
}
```

Использование:

```
Department programmersDepartment = new Department("Программисты");
programmersDepartment.addEmployee(sysEngineer);
System.out.println("Относится к   отделу "+sysEngineer.getDepartment().name);
```

2.3.1 Композиция

Предположим, что одним из требований к нашей системе является требование о том, чтоб хранить данные о прежней занимаемой должности на предприятии.
Введем новый класс "pastPosition". В него, помимо свойства "имя"(name), введем и свойство "department", которое свяжет его с классом "Department".

Данные о прошлых занимаемых должностях являются частью данных о сотруднике, таким образом между ними связ целое-часть и в то же время,
данные о прошлых должностях не могут существовать без объекта типа "Employee". Уничтожение объекта "Employee" должно привести к уничтожению
объектов "pastPosition".

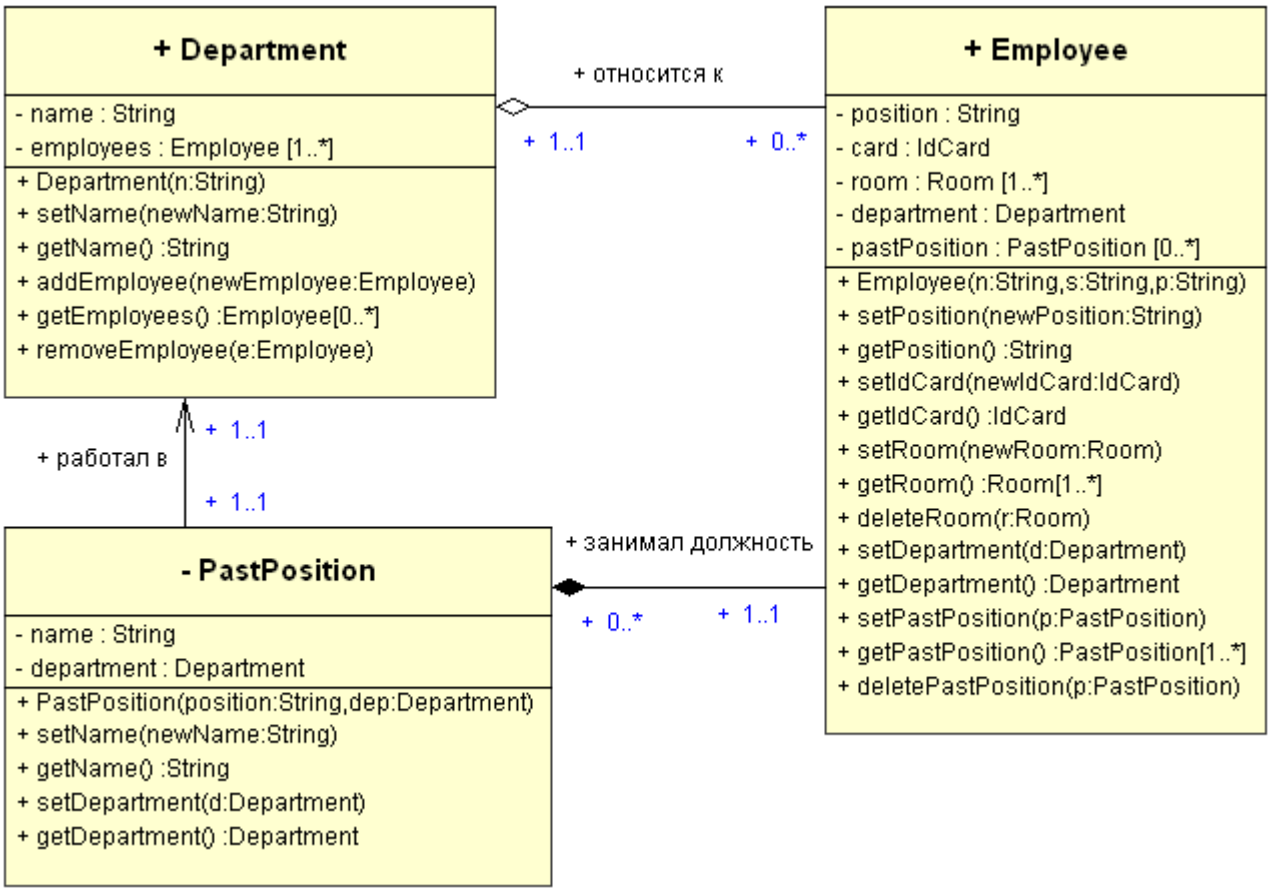

Рис. 6 - Композиция

Класс "pastPosition":

```
[code]
        private static class PastPosition
        {
                private String name;
                private Department department;
                public PastPosition(String position, Department dep)
                {
                        name = position;
                        department = dep;
                }

                public void setName(String newName)
                {
                        name = newName;
                }

                public String getName()
                {
                        return name;
                }

                public void setDepartment(Department d)
                {
                        department = d;
                }
```

```
		public Department getDepartment()
		{
			return department;
		}
	}
```
[/code]

В класс Employee добавим свойства и методы для работы с данными о прошлой должности:
[code]
```
...
private Set pastPosition = new HashSet();
...
public void setPastPosition(PastPosition p)
{
	pastPosition.add(p);
}
public Set  getPastPosition()
{
	return pastPosition;
}

	public void deletePastPosition(PastPosition p)
{
	pastPosition.remove(p);
}
...
```
[/code]

Применение:
[code]
```
//изменяем должность
sysEngineer.setPosition("Сторож");

// смотрим ранее занимаемые должности:
System.out.println("В прошлом работал как:");
Iterator iter =  sysEngineer.getPastPosition().iterator();
while(iter.hasNext())
	System.out.println( ((PastPosition) iter.next()).getName());
```
[/code]

3. Зависимость

Для диалога с пользователем введем в систему класс "Menu". Для упрощения в него встроим один метод "showEmployees", который показывет список сотрудников и их должности. Параметром для метода является массив объектов "Employee". Таким образом, изменения внесенные в класс "Employee" могут потребовать и изменения класса "Menu".

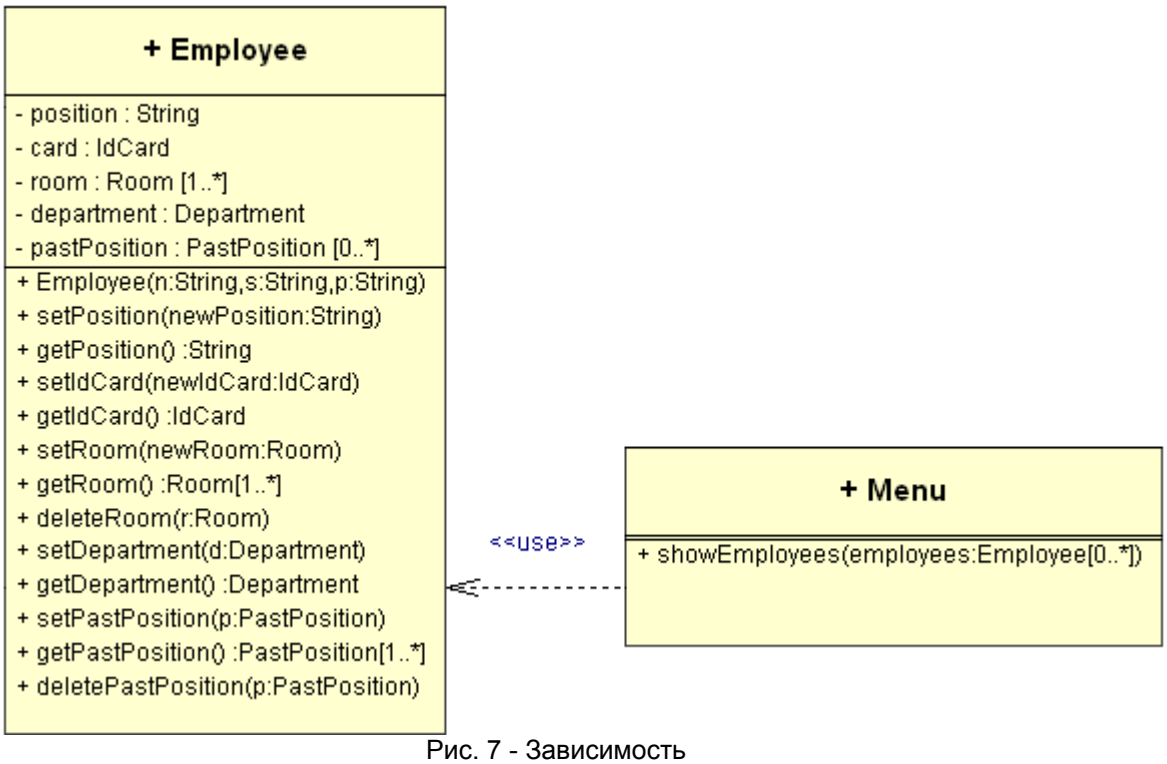

Рис. 7 - Зависимость

Заметим, что класс "Menu" не относится к исследуемой прикладной области, а представляет собой класс модели приложения.
Класс "Menu":

```
public static class Menu
{
	private static int i=0;
	public static void showEmployees(Employee[] employees)
	{
		System.out.println("Список сотрудников:");
		for (i=0; i<employees.length; i++)
			if(employees[i] instanceof Employee)
				System.out.println(employees[i].getName() +" - " +
employees[i].getPosition());
	}
}
```

Использование:

```
// добавим еще одного сотрудника
Employee director = new Employee("Федор", "Дубов", "Директор");

Menu menu = new Menu();
Employee employees[] = new Employee[10];

employees[0]= sysEngineer;
employees[1] = director;
Menu.showEmployees(employees);
```

4. Реализация

Реализация, как и наследование имеет явное выражение в языке Java : объявление интерфейса и возможность его раелизации каким-либо классом.

Для демонстрации отношения "реализация" создадим интерфейс "Unit". Если представить, что организация может делиться не только на отделы, а например, на цеха, филиалы и т.д. Интерфейс "Unit" представялет собой самый абстрактную еденицу деления. В каждой еденице деления работает какое-то количество сотрудников, поэтому метод для получения количества работающих людей будет актуален для каждого класса реализующего интерфейс "Unit".

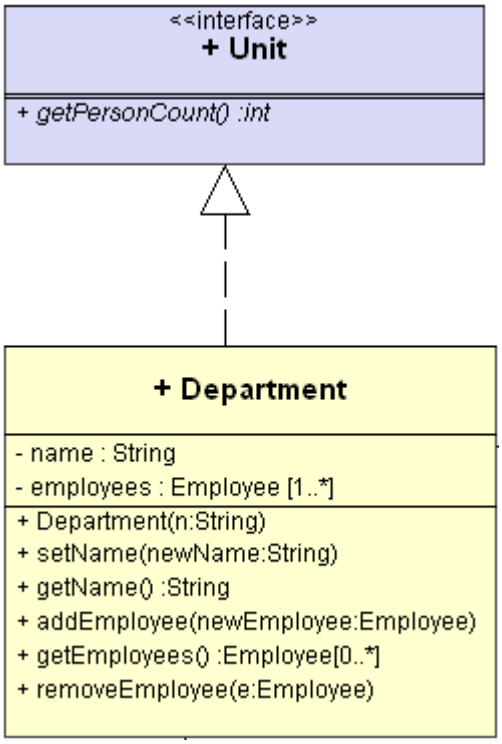

Рис. 7 - Реализация

Интерфейс "Unit":
[code]
```
public interface Unit
{
	int getPersonCount();
}
```
[/code]

Реализация в классе "Department":
[code]
```
public static class Department implements Unit
...
public int getPersonCount()
{
	return getEmployees().size();
}
```
[/code]

Применение:
[code]

```
System.out.println("В отделе "+sysEngineer.getDepartment().name+" работает "
		+sysEngineer.getDepartment().getPersonCount()+" человек.");
```

[/code]

Как видим, реализация метода "getPersonCount" не совсем актульна для класса "Department", так как он имеет метод "getEmployees", который возвращает коллекцию объектов "Employee".

Вывод

Язык моделирования UML предоставляет богатый набор отношений для построения классовой модели. Но даже такой развитой ООП язык, как Java имеет две констукции для отражения связей: extends(расширение) и interface/implements(реализация).
В результате моделирования получили следующую диаграмму:

UML Class Model

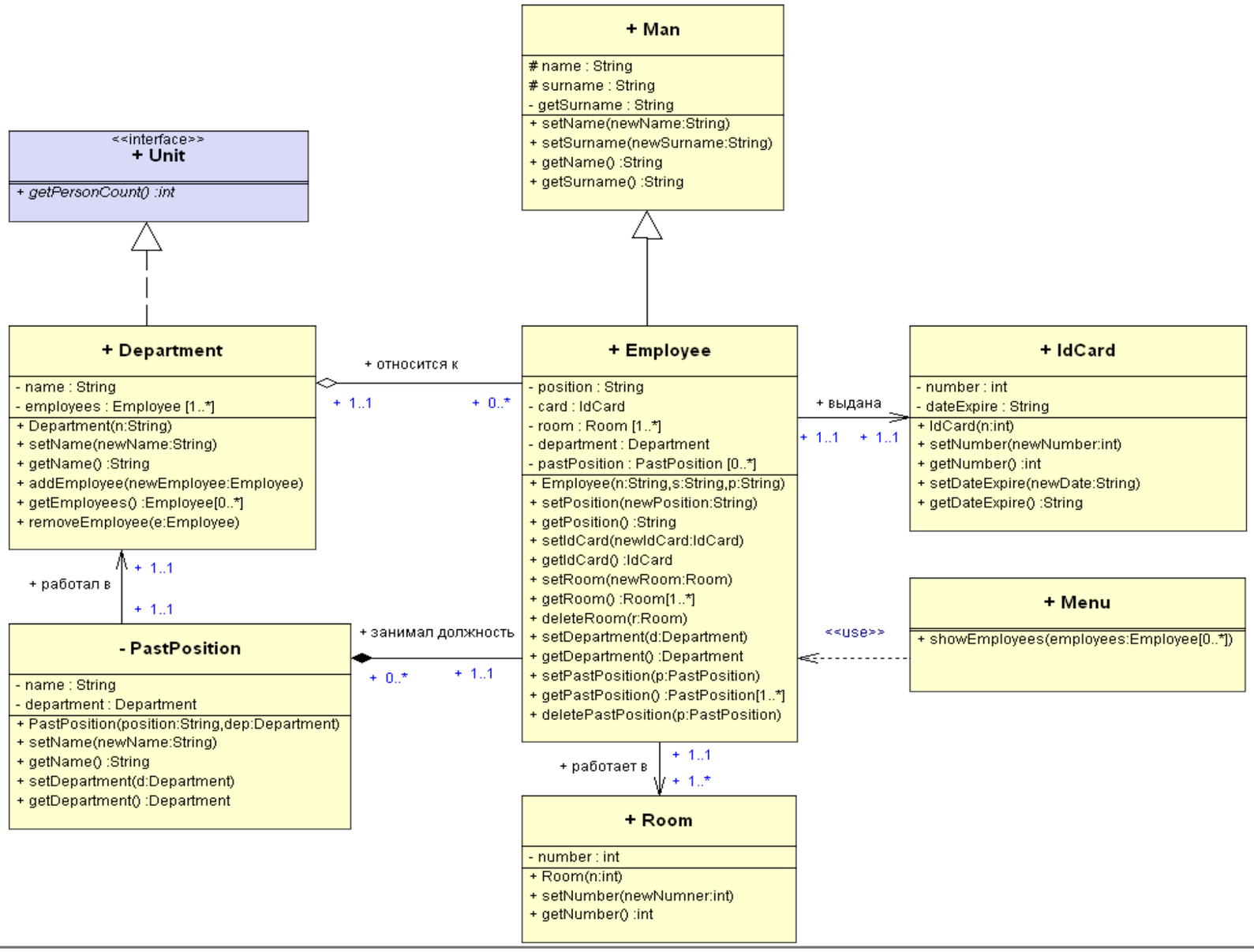

Рис. 8 - Диаграмма классов

Литература

1) Г. Буч, Д. Рамбо, А. Джекобсон. Язык UML Руководство пользователя.

2) Самоучитель UML
http://fictionbook.ru/author/aleksandr_vasilevich_leonenkov/samouchitel_uml/read_online.html?page=11

3) Язык UML. Руководство пользователя
http://alice.pnzgu.ru/~dvn/uproc/books/uml_user_guide/index.htm

4) Эккель Б. Философия Java. Библиотека программиста. - СПб: Питер, 2001. - 880 с.

5) Орлов С. Технологии разработки программного обеспечения: Учебник. - СПб: Питер, 2002. - 464 с.

6) Мухортов В.В.,Рылов В.Ю.Объектно-ориентированноепрограммирование, анализ и дизайн. Методическоепособие. - Новосибирск, 2002. -

7) Modeling Class Relationships in UML
http://www.byteonic.com/2006/modeling-class-relationships-in-uml-part-1/