Диаграмма активностей

Показывает все возможные состояния, в которых может находиться система, а также переходы между этими состояниями. Эта диаграмма используется для описания поведения системы в различных ситуациях.

У нас есть начальное состояние и конечное состояние. Все действия такие как (нанять научного сотрудника, преподавать курс, старт проекта, завершение проекта) связаны переходами потока, мы используем стрелку чтобы изобразить поток управления

Узел принятия решений(ромб) позволяет нам один входящий поток разделить на множество исходящих.

Узел слияния – объединяет несколько входящих потоков в один.

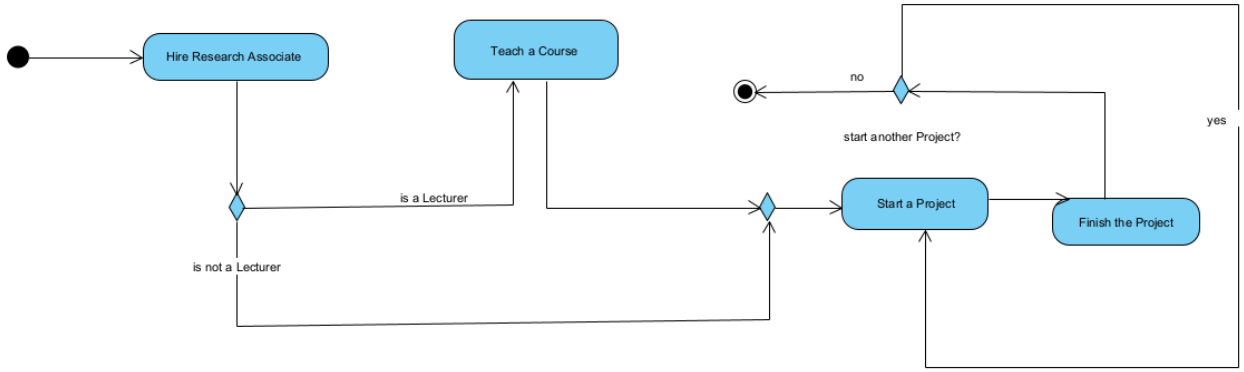


Диаграмма последовательностей

Эта диаграмма описывает взаимодействие между набором объектов, участвующих в сотрудничестве (или сценарии), расположенных в хронологическом порядке; он показывает объекты, участвующие во взаимодействии, по их «линиям жизни» и сообщениям, которые они посылают друг другу, пунктирные линии это обратные сообщения.

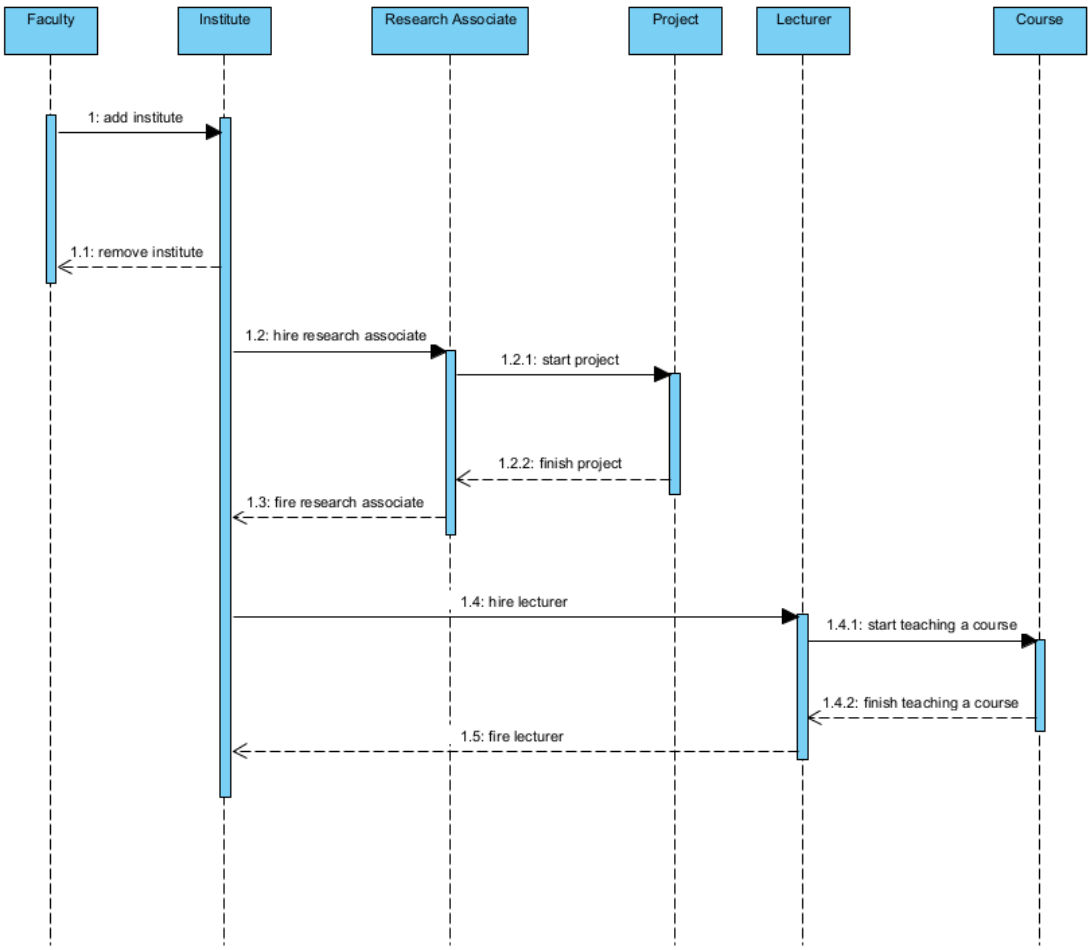


Диаграмма классов

Диаграмма классов описывает структуру объектно-ориентированной системы, показывая классы в этой системе и отношения между классами. Диаграмма классов также показывает ограничения и атрибуты классов.

+ это операции. Которые видимы всем т.е public – это атрибуты которые приватны

Ассоциация — это связь между двумя классами.

Существует двунаправленная - это означает, что оба класса знают друг о друге и об их отношениях.

Множественность символы (1 0..1 * 0..* 1..*) обозначают количество экземпляров одного класса, связанных с одним экземпляром другого класса

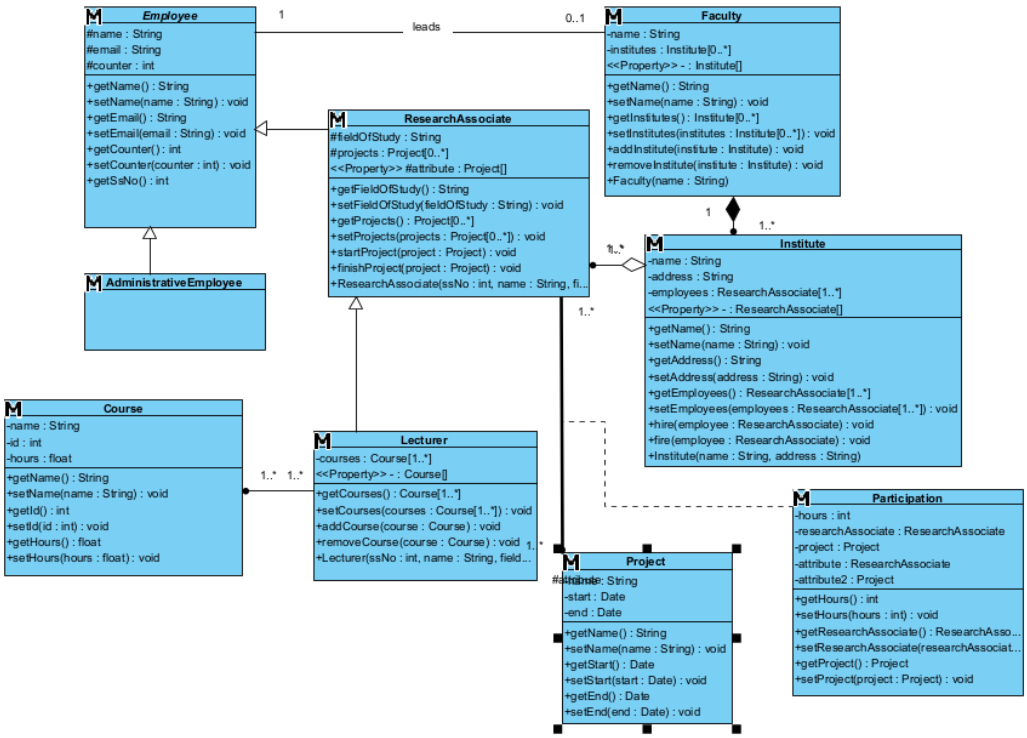
Обобщение - связь между классом и подклассом.(родительский, дочерний) обозначается стрелкой

Реализация — это связь между двумя вещами, где одна вещь (интерфейс) определяет контракт, который другая вещь (класс) гарантирует выполнение, реализуя операции, указанные в этом контракте. Обозначается пунктирной линией либо обозначает зависимость одного класса от другого.

Агрегация встречается, когда один класс является коллекцией или контейнером других. Причём по умолчанию, агрегацией называют агрегацию по ссылке, то есть когда время существования

содержащихся классов не зависит от времени существования содержащего их класса. Если контейнер будет уничтожен, то его содержимое — нет. Графически агрегация представляется пустым ромбом

Композиция имеет жёсткую зависимость времени существования экземпляров класса контейнера и экземпляров содержащихся классов. Если контейнер будет уничтожен, то всё его содержимое будет также уничтожено. Графически представляется, как и агрегация, но с закрашенным ромбиком.



Перейдем к небольшому описанию

1. У нас есть факультет, который имеет множество кафедр и он жестко связны связью композиций без одного не будет другого. У него есть имя, атрибуты кафедра факультет и возможность добавлять и удалять кафедры

```
public class Faculty {
        private String name;
        private Institute[] institutes;
        public Faculty(String name) {
                this.name = name;
        }
        public String getName() {
                return this.name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public Institute[] getInstitutes() {
                return this.institutes;
        }
        public void setInstitutes(Institute[] institutes) {
```

```
            this.institutes = institutes;
        }
        public void addInstitute(Institute institute) {
            // TODO - implement Faculty.addInstitute
            throw new UnsupportedOperationException();
    }
        public void removeInstitute(Institute institute) {
            // TODO - implement Faculty.removeInstitute
            throw new UnsupportedOperationException();
        }}
```

2. На факультете есть класс сотрудники которые могут быть научными сотрудниками(они к свою очередь являются подклассом класса сотрудники)

```
public abstract class Employee {
        protected int ssNo;
        protected String name;
        protected String email;
        protected int counter;
        public int getSsNo() {
            return this.ssNo;
        }
        public void setSsNo(int ssNo) {
            this.ssNo = ssNo;
        }
        public String getName() {
            return this.name;
        }
        public void setName(String name) {

        this.name = name;

        }
        public String getEmail() {
            return this.email;
        }
        public void setEmail(String email) {
            this.email = email;
        }
        public int getCounter() {
        return this.counter;
        }

    public void setCounter(int counter) {

            this.counter = counter;

    }}

    }
```

3. На кафедрах есть научные сотрудники, каждый сотрудник прикреплен к одной кафедре она является для них контейнером образовано связью агрегации.

```
public class ResearchAssociate extends Employee {
	protected String fieldOfStudy;
	protected Project[] projects;
	public ResearchAssociate(int ssNo, String name, String fieldOfStudy) {
		this.ssNo = ssNo;
		this.name = name;
		this.fieldOfStudy = fieldOfStudy;
	}
	public String getFieldOfStudy() {
		return this.fieldOfStudy;
	}
	public void setFieldOfStudy(String fieldOfStudy) {
		this.fieldOfStudy = fieldOfStudy;
	}
	public Project[] getProjects() {
		return this.projects;
	}
	public void setProjects(Project[] projects) {
		this.projects = projects;
	}
	public void startProject(Project project) {
		// TODO - implement ResearchAssociate.startProject
		throw new UnsupportedOperationException();
	}
	public void finishProject(Project project) {
		// TODO - implement ResearchAssociate.finishProject
		throw new UnsupportedOperationException();
	}


}
```

4. каждый научный сотрудник может быть участником в проектах (project). На опр время

```
public class Project {

	private String name;
	private Date start;
	private Date end;

	public String getName() {
		return this.name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public Date getStart() {
```

```
            return this.start;
        }

        public void setStart(Date start) {
            this.start = start;
        }

        public Date getEnd() {
            return this.end;
        }

        public void setEnd(Date end) {
            this.end = end;
        }
    }
```

Вывод

```
public class Main {
    public static void main(String[] args) {
        Faculty fkn = new Faculty("FKN");
        Institute toizi = new Institute("TOIZI", "297");
        Institute is = new Institute("IS", "387");
        Institute itu = new Institute("ITU", "312п");
        Institute piit = new Institute("PIIT", "380");
        fkn.setInstitutes(new Institute[]{toizi, is, itu, piit});
        ResearchAssociate emp1 = new ResearchAssociate(1234, "Denis Ryindin
Dmitrievich",
                "data protection");
        ResearchAssociate emp2 = new ResearchAssociate(1234, "Golovlev Gordey
Stepanovich",
                "systems modeling");
        ResearchAssociate emp3 = new ResearchAssociate(1234, "Sviridov Danil
Petrovich",
                "\n" +
                        "intelligent systems");
        ResearchAssociate emp4 = new ResearchAssociate(1234, "Denis Ryindin
Dmitrievich",
                "pattern recognition");
        ResearchAssociate emp5 = new ResearchAssociate(1234, "Denis Ryindin
Dmitrievich",
                "information processing technologies");
        toizi.setEmployees(new ResearchAssociate[]{emp1, emp2, emp3, emp4,
emp5});
        Institute[] fkn_institutes = fkn.getInstitutes();
        System.out.println("\n" +
                "Departments " + fkn.getName() + ": \n");
        for (int i = 0; i < fkn_institutes.length; i++) {
            System.out.println(fkn_institutes[i].getName());
        }
        System.out.println("\n");

    }
```