Advanced Programming

→ UML Class Diagram

Hello!

I am Hamzeh Asefan

Master Degree from King Abdullah II School of Information Technology in Information Systems (2020/2021) / The University of Jordan.

1. Unified Modeling Language (UML)

Provide a standard way to visualize the design of a system. It was adopted as the standard in 1997.

“UML is a modern approach to modeling and documenting software. By using visual representations, we are able to better understand possible flaws or errors in software or business processes.”

2. UML Class Diagram

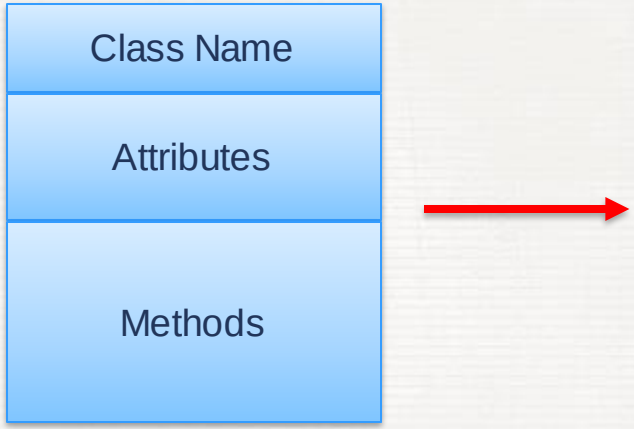
Describes the structure of a system by showing the system's classes, their attributes, methods, and the relationships.

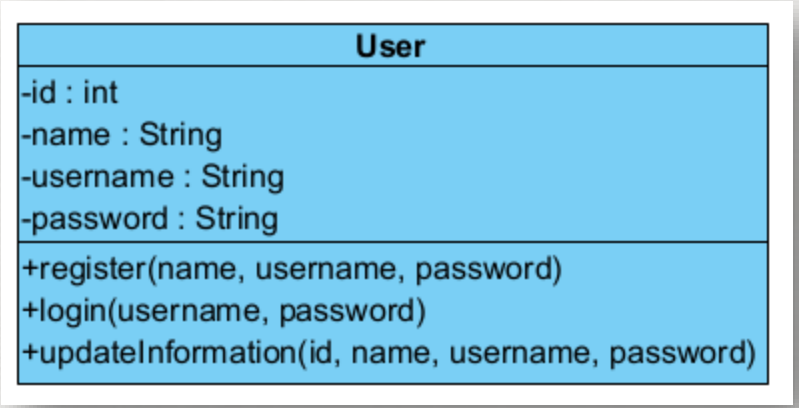
"Class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes."

Visual Paradigm

- Is a UML CASE Tool.
- Powerful visual modeling tools that helps you build and manage your diagrams and model elements.
- Community Edition: a free edition for non-commercial use only, supports all 14 UML diagram types (Class diagram, Use case diagram, Sequence diagram, Object diagram, ...).

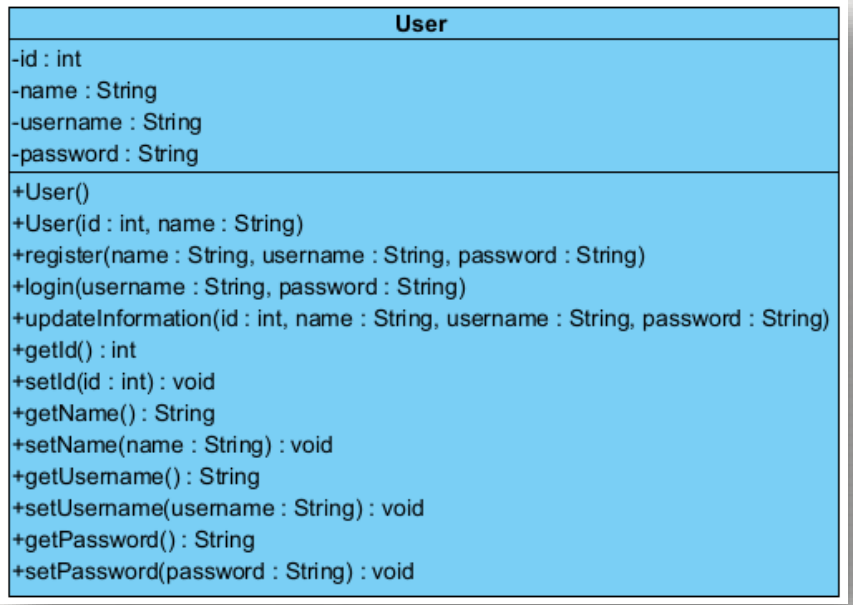
Example


Class Name
Attributes
Methods


User
-id : int
-name : String
-username : String
-password : String
+register(name, username, password)
+login(username, password)
+updateInformation(id, name, username, password)

```java
class User {

    private int id;
    private String name;
    private String username;
    private String password;

    public void register(String name, String username, String password) {
        // TODO - implement User.register
    }

    public void login(String username, String password) {
        // TODO - implement User.login
    }

    public void updateInformation(int id, String name, String username, String password) {
        // TODO - implement User.updateInformation
    }
```

Example



Generate Code

Visibility

Visibility	Symbol
Private	-
Public	+
Package	~
Protected	#

Exercise → Employee

Employee
-id : int -name : String -salary : double
+Employee() +Employee(id : int, name : String, salary : double) +getId() : int +setId(id : int) : void +getName() : String +setName(name : String) : void +getSalary() : double +setSalary(salary : double) : void

3. Relationships

In UML, the ways that things can connect to one another, either logically or physically, are modelled as relationships.

Relationships

1. **Dependencies**
   Represents 'using' relationship among classes. One-sided relationship.

2. **Generalizations**
   Connects generalized classes to more specialized ones (inheritance). One-sided relationship.

3. **Realization**
   Represents relationships between an interface and a class to carry out the implementation.

Relationships → Associations

4 **Associations**

Represents structural relationships among instances/objects. Two-sided relationship

5 **Aggregation**

The aggregate class contains a reference to another class and is said to have ownership of that class.

6 **Composition**

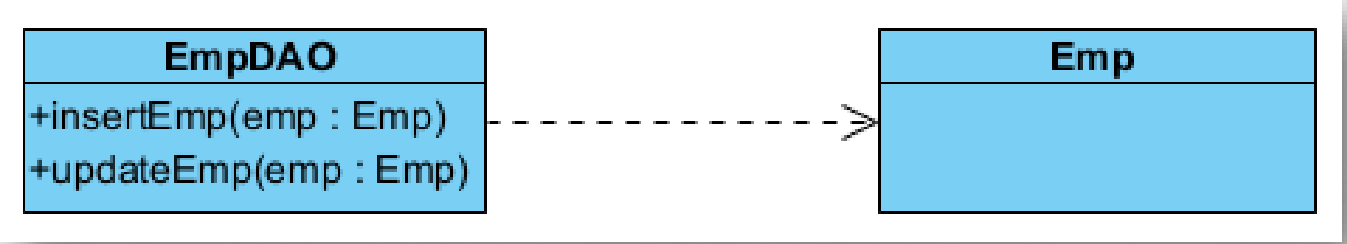
There is a strong relationship between one class and another. Other classes cannot exist without the owner or parent class.

Dependencies

- Show that one class uses another class as an argument in the signature of an operation.

- Create a dependency pointing from the class with the operation to the class used as a parameter in the operation.

Dependencies Example



Dependencies Example

Database Table (Emp)

SQL | Output | Statistics

```
SELECT EMPNO,
       ENAME,
       SAL
  FROM EMP;
```

	EMPNO	ENAME	SAL
1	7369	SMITH	800.00
2	7499	ALLEN	1600.00
3	7521	WARD	1250.00
4	7566	JONES	2975.00
5	7654	MARTIN	1250.00
6	7698	BLAKE	2850.00
7	7782	CLARK	2450.00
8	7788	SCOTT	3000.00
9	7839	KING	5000.00
10	7844	TURNER	1500.00
11	7876	ADAMS	1100.00

Bean (Emp.java)

```java
public class Emp {

    private int id;
    private String name;
    private double salary;

    public Emp() {

    }

    public Emp(int id, String name, double salary) {

        this.id = id;
        this.name = name;
        this.salary = salary;
    }
```

DAO (EmpDAO.java)

The Data Access Object (DAO) pattern is allows us to isolate the business layer from the persistence layer.

Dependencies Example

DAO (EmpDAO.java)

```java
public class EmpDAO {

    public void insert(int id, String name, double salary) {
        Connection conn;
        try {
            conn = getConnection();

            PreparedStatement ps = conn.prepareStatement("insert into emp(empno, ename, sal) values (?, ?, ?)");

            ps.setInt(1, id);
            ps.setString(2, name);
            ps.setDouble(3, salary);

            ps.executeUpdate();
            conn.commit();

            ps.close();
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

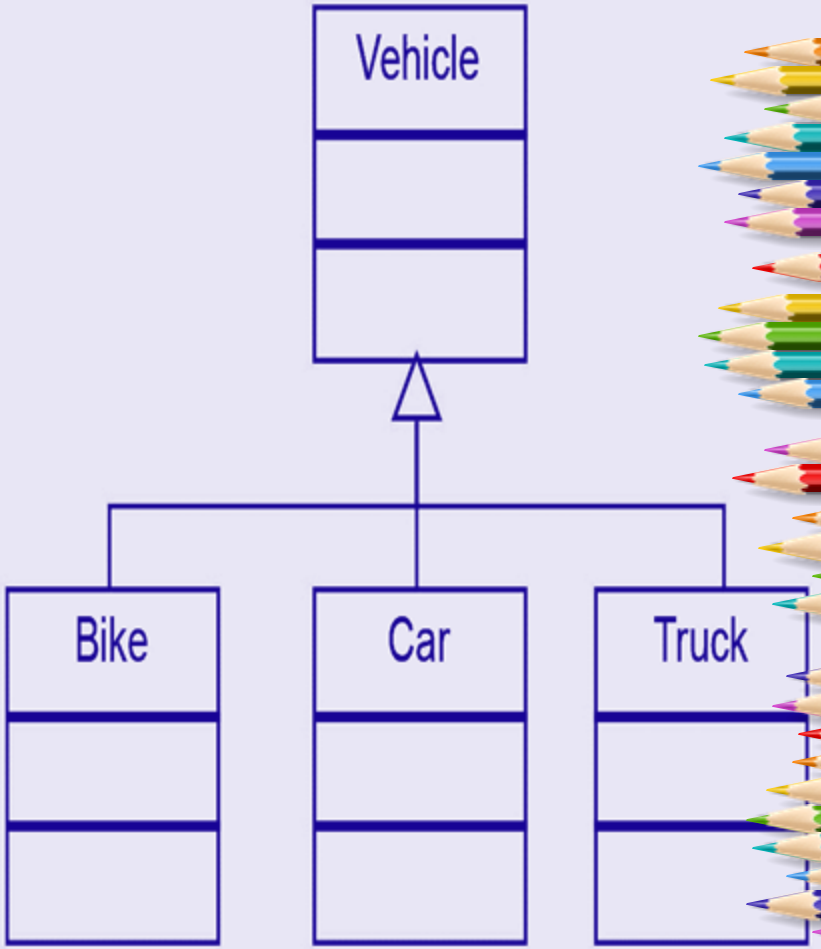
Dependencies Example

DAO (EmpDAO.java)

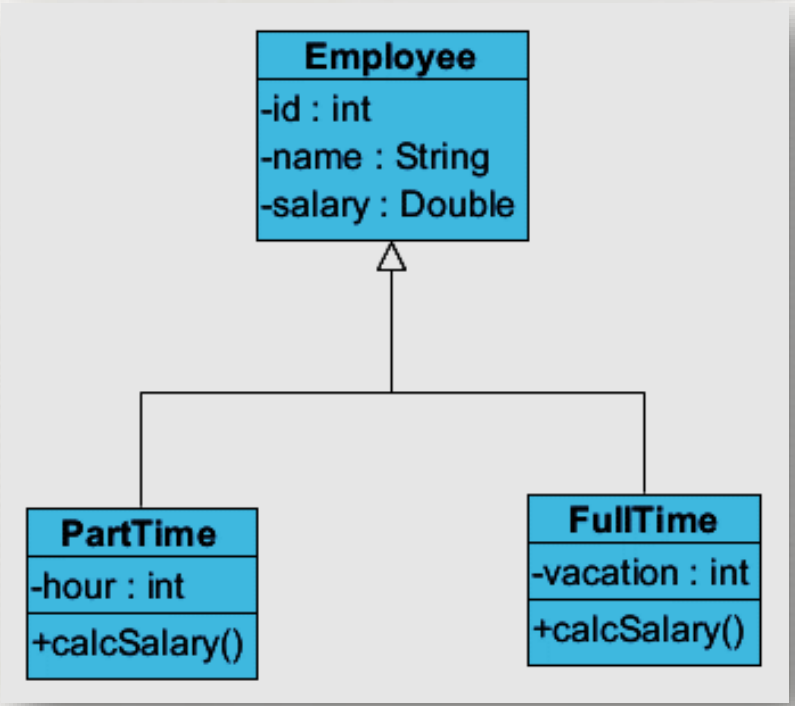
```
public class EmpDAO {

    public void insert(Emp emp) {
        Connection conn;
        try {
            conn = getConnection();

            PreparedStatement ps = conn.prepareStatement("insert into emp(empno, ename, sal) values (?, ?, ?)");

            ps.setInt(1, emp.getId());
            ps.setString(2, emp.getName());
            ps.setDouble(3, emp.getSalary());

            ps.executeUpdate();
            conn.commit();

            ps.close();
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

Generalization (Inheritance)

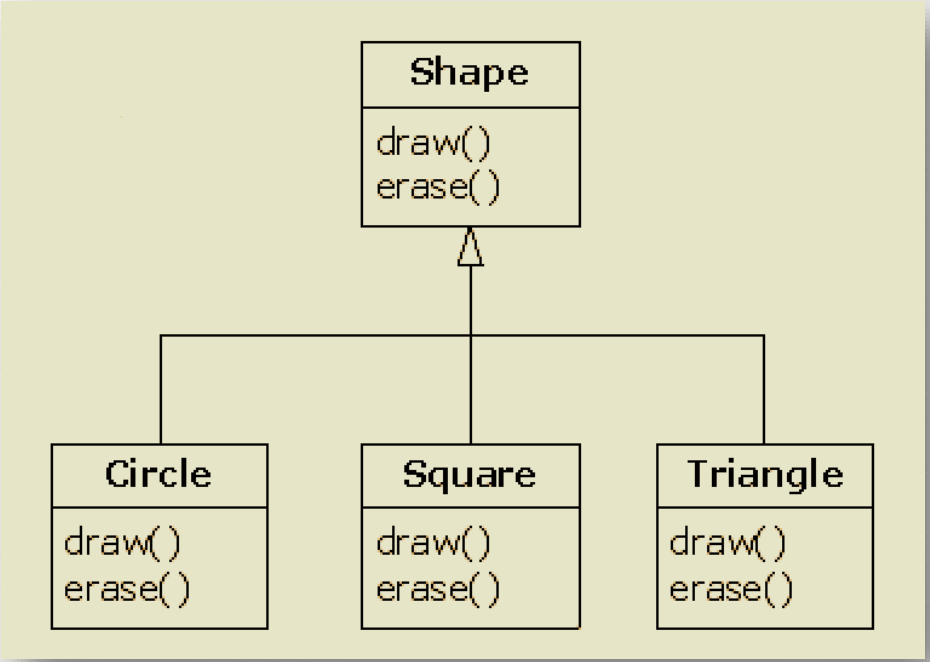
- It represents <u>is a</u> relationship.
- Is a relationship between a general thing (superclass) and a more specific kind of that thing (subclass).



Generalization Example



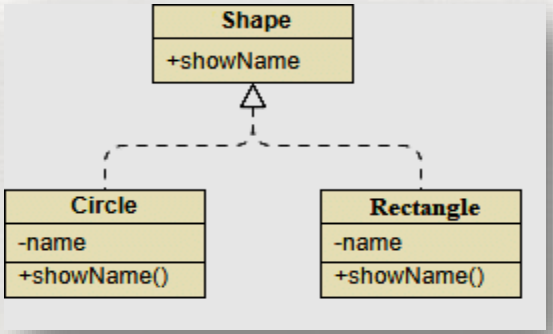
Generalization Example



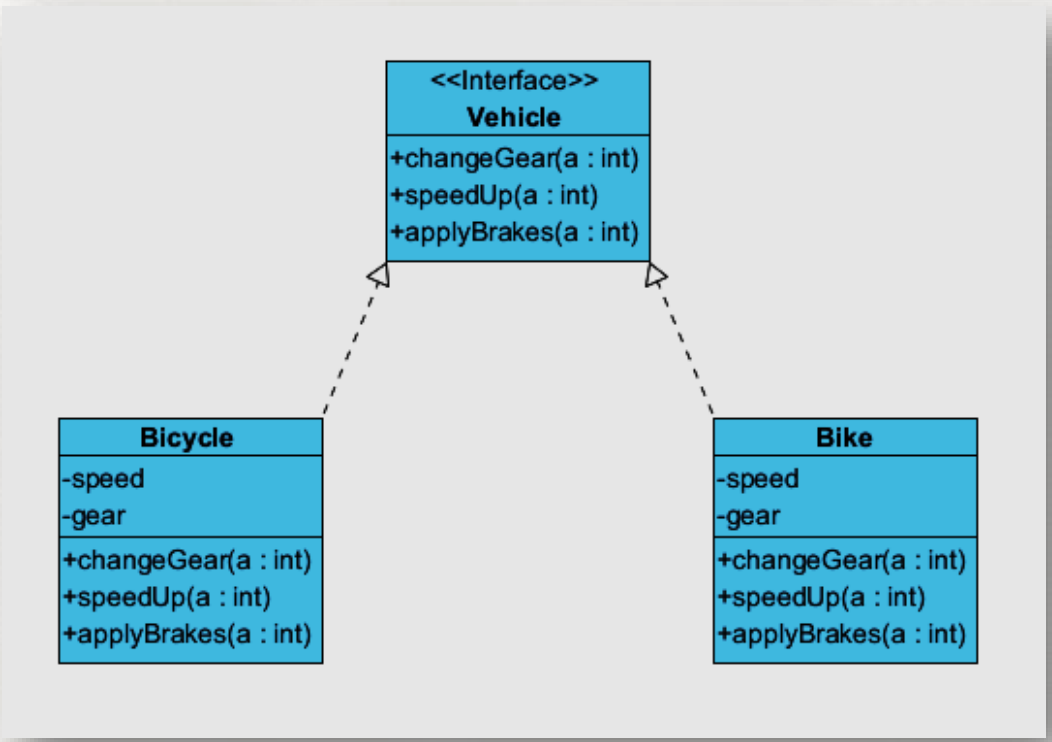
Realization

- A realization is a relationship between two things where one thing (an interface) specifies a contract that another thing (a class) guarantees to carry out by implementing the operations specified in that contract.

- In a class diagram, realization relationship is rendered as a dashed directed line with an open arrowhead pointing to the interface.



Realization Example



Association

- Association is a relation between two separate classes which establishes through their objects.
- Binary association: connects exactly two classes.
- Ternary association: connects more than two classes.
- Association is shown as a solid line connecting the same or different classes.

Association

We can apply role and multiplicity for association:

1. **Role** → Explicit name can be given to the role a class plays in an association.
2. **Multiplicity** → State how many objects may be connected across an instance of an association.

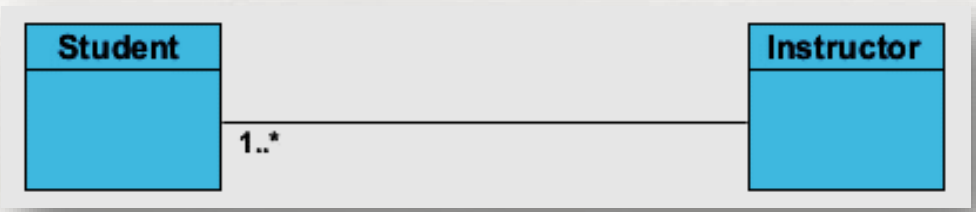
Association → Multiplicity

Values	Meaning
1	Exactly one
0..1	Zero or one
0..*	Zero or many
1..*	One or many
*	Many

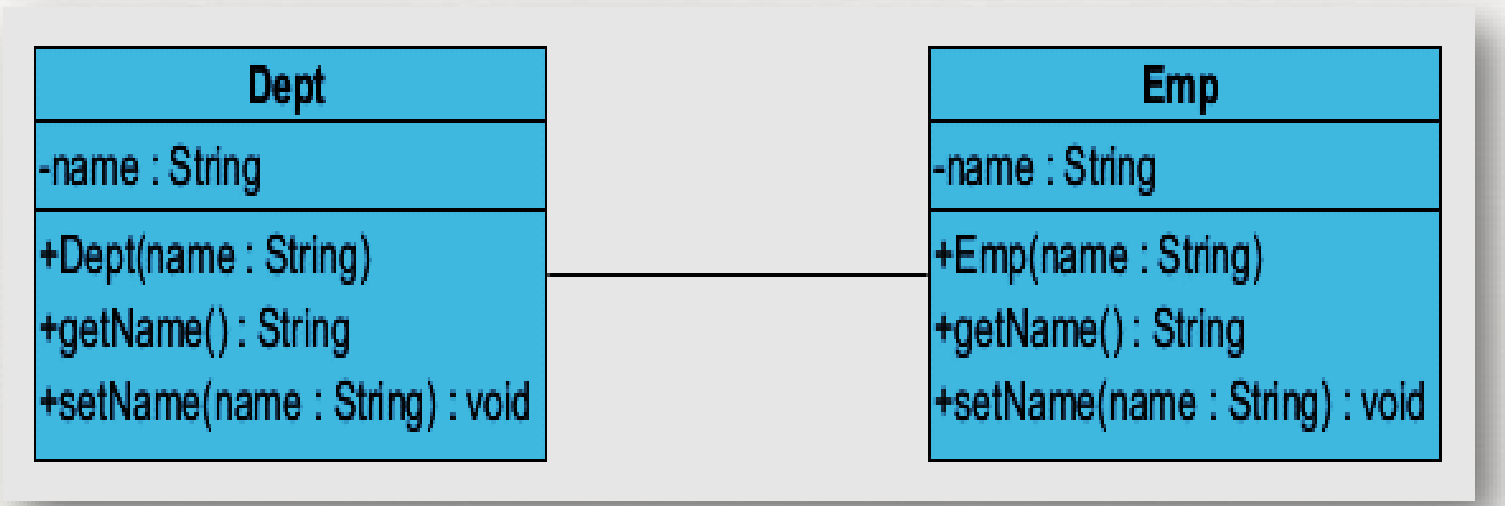
Association

- A single student can associate with multiple teachers:

- Every instructor has one or more students:



Association Example



Association Example

```java
public class Dept {

    private String name;

    public Dept(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

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

}
```

```java
public class Emp {

    private String name;

    public Emp(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

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

}
```

Association Example

```java
package hmz;


public class Test {

    public static void main(String[] args) {
        Dept dept = new Dept("Development");
        Emp emp = new Emp("Ali");

        System.out.println(emp.getName() + " is employee of " + dept.getName());

    }
}
```

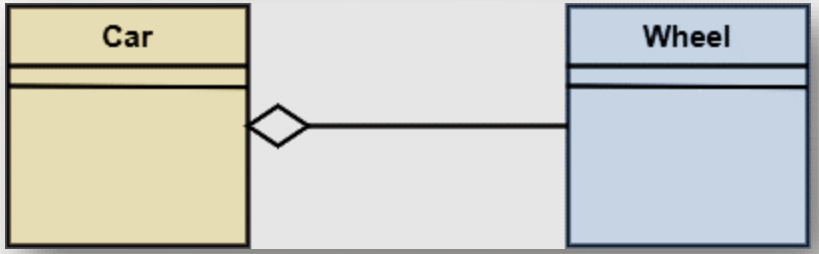
Problems | Console | Progress

```
<terminated> Test (1) [Java Application] D:\jsfCourse\java\jdk1.8.0_202\bin\javaw.exe (14/11/2021 11:14:43 ص – ١١:١٤:٤٣ ص)
Ali is employee of Development
```

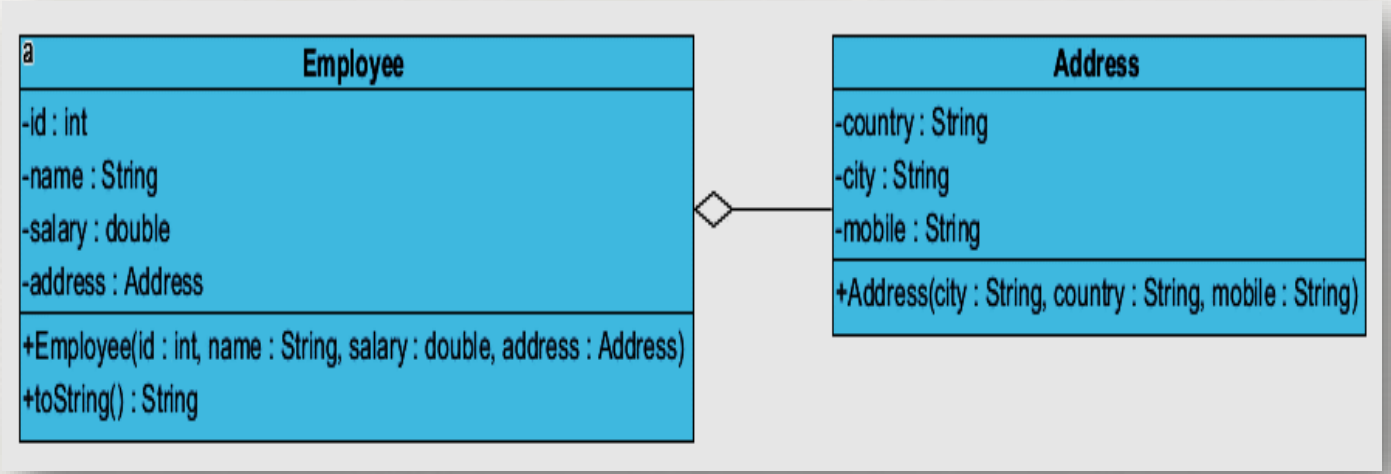

HTU
جامعة الحسين التقنية
Al Hussein Technical University
HTU

Aggregation

- It represents <u>has a</u> relationship.
- Implies a relationship where the child can exist independently of the parent.
- For example → Car has a wheel:
  - A car cannot move without a wheel, but the wheel can be independently used with the bike, scooter, ….
  - The wheel object can exist without the car object.



Aggregation Example



Aggregation Example

```java
public class Address {

    private String city;
    private String country;
    private String mobile;

    public Address() {

    }

    public Address(String city, String country, String mobile) {
        this.city = city;
        this.country = country;
        this.mobile = mobile;
    }

    @Override
    public String toString() {
        return "[city=" + city + ", country=" + country + ", mobile=" + mobile + "]";
    }
```

Aggregation Example

```java
public class Employee {

    private int id;
    private String name;
    private double salary;
    private Address address;

    public Employee() {

    }

    public Employee(int id, String name, double salary, Address address) {
        super();
        this.id = id;
        this.name = name;
        this.salary = salary;
        this.address = address;
    }

    @Override
    public String toString() {
        return "[id=" + id + ", name=" + name + ", salary=" + salary + ", address=" + address + "]";
    }
```

Aggregation Example

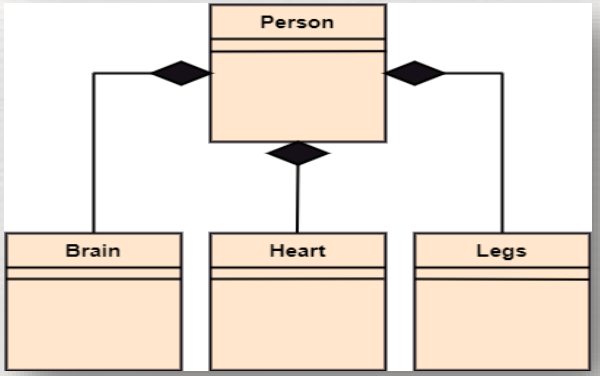
```java
package hmz;

public class Test {

    public static void main(String[] args) {
        Address address1 = new Address("Amman", "Jordan", "96279...");
        Address address2 = new Address("Irbid", "Jordan", "96278...");

        Employee e1 = new Employee(111, "Ali ", 1000, address1);
        Employee e2 = new Employee(112, "Sami", 1200, address2);

        System.out.println("Employee 1 --> " + e1);
        System.out.println("Employee 2 --> " + e2);

    }
}
```

Problems | Console | Progress

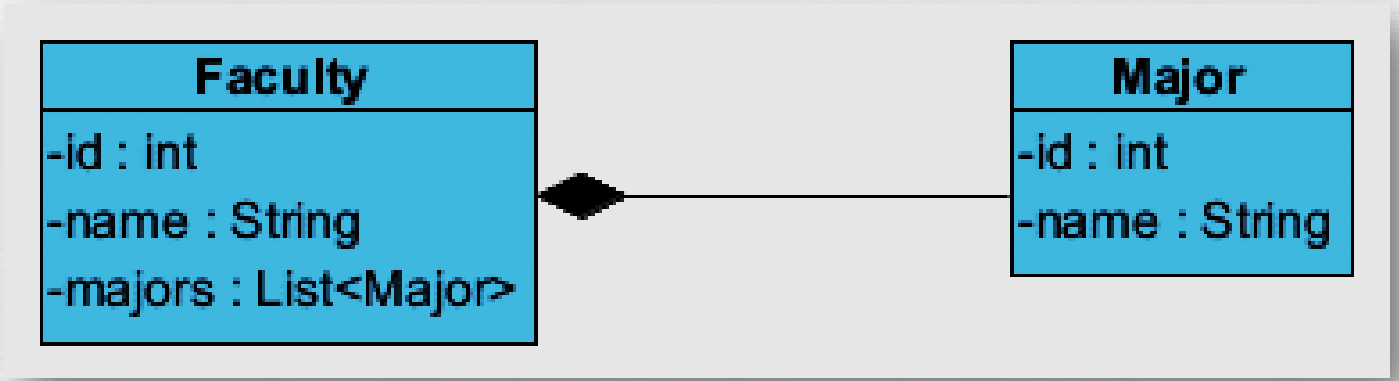
```
<terminated> Test (1) [Java Application] D:\jsfCourse\java\jdk1.8.0_202\bin\javaw.exe (12/11/2021 06:26:33 م ٠٦:٢٦:٣٣ – م)
Employee 1 --> [id=111, name=Ali , salary=1000.0, address=[city=Amman, country=Jordan, mobile=96279...]]
Employee 2 --> [id=112, name=Sami, salary=1200.0, address=[city=Irbid, country=Jordan, mobile=96278...]]
```

Composition

- It represents “belongs-to / part-of ” relationship.
- Implies a relationship where the child can not exist independently of the parent.
- For example → Brain, Heart and Legs belong to Person:
  - If the person is destroyed, the brain, heart, and legs will also get discarded.



Composition Example



Composition Example

```java
public class Major {

    private int id;
    private String name;

    public Major() {

    }

    public Major(int id, String name) {
        this.id = id;
        this.name = name;
    }
```

```java
public class Faculty {

    private int id;
    private String name;
    private List<Major> majors;

    public Faculty() {

    }

    public Faculty(int id, String name, List<Major> majors) {
        this.id = id;
        this.name = name;
        this.majors = majors;
    }
```

Composition Example

```java
package hmz;

import java.util.ArrayList;
import java.util.List;

public class Test {

    public static void main(String[] args) {

        Major major1 = new Major(1, "Computer Science");
        Major major2 = new Major(2, "Data Science");
        Major major3 = new Major(3, "Cyber Security");

        List<Major> majors = new ArrayList<Major>();
        majors.add(major1);
        majors.add(major2);
        majors.add(major3);

        Faculty faculty = new Faculty(1, "School of Computing and Informatics", majors);

        System.out.println(faculty);
    }
}
```

Problems | Console | Progress

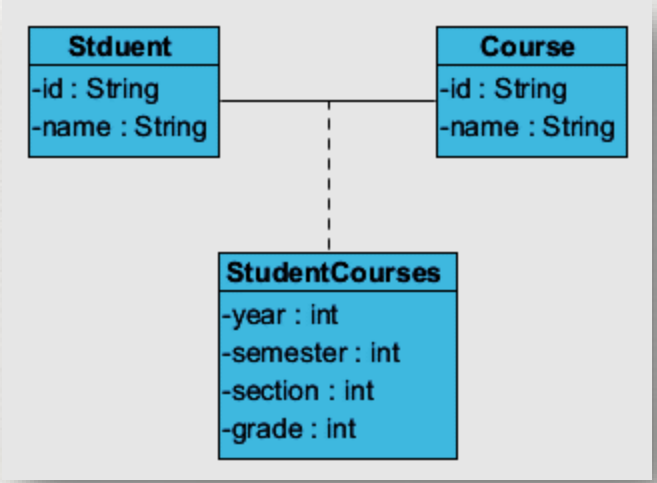
```
<terminated> Test (1) [Java Application] D:\jsfCourse\java\jdk1.8.0_202\bin\javaw.exe (12/11/2021 10:13:41 م ١٠:١٣:٤٢ – م)
[id=1, name=School of Computing and Informatics, majors=[[id=1, name=Computer Science], [id=2, name=Data Science], [id=3, name=Cyber Security]]]
```

Association Class

- Association class is a class that is part of an association relationship between two other classes.
- You can attach an association class to an association relationship to provide additional information about the relationship.
- An association class is both a class and a relation.

Association Class Example

- A class called Student represents a student and has an association with a class called Course, which represents an educational course.
- The Student class can register in a course. An association class called StudentCourse which defines the relationship between the Student and Course classes by providing year, semester, section and grade.



Association Class Example

```java
public class Student {

    private String id;
    private String name;
    private List<StudentCourses> studentCourses = new ArrayList<StudentCourses>();

    public Student() {

    }

    public Student(String id, String name) {
        this.id = id;
        this.name = name;
    }

    public void addStudentCourse(StudentCourses ... course) {
        for (int i = 0; i < course.length; i++) {
            studentCourses.add(course[i]);
        }
    }
```

Association Class Example

```java
public class Course {

    private String id;
    private String name;

    public Course(String id, String name) {
        this.id = id;
        this.name = name;
    }
```

```java
public class StudentCourses {

    private Course course;
    private int year;
    private String semester;
    private int section;
    private int grade;

    public StudentCourses(Course course, int year, String semester, int section, int grade) {
        this.course = course;
        this.year = year;
        this.semester = semester;
        this.section = section;
        this.grade = grade;
    }
```

Association Class Example

```java
public class Test {

    public static void main(String[] args) {

        Student s1 = new Student("0210001", "Hamzeh");

        Course c1 = new Course("01" , "Database                 ");
        Course c2 = new Course("02" , "Advanced Programming     ");
        Course c3 = new Course("03" , "Artificial Intelligence");

        StudentCourses studentCourse1 = new StudentCourses(c1, 2020, "First ", 1, 95);
        StudentCourses studentCourse2 = new StudentCourses(c2, 2020, "Second", 3, 92);
        StudentCourses studentCourse3 = new StudentCourses(c3, 2020, "Summer", 3, 88);

        s1.addStudentCourse(studentCourse1, studentCourse2, studentCourse3);
        System.out.println(s1);

    }
}
```

```
Student --> id=0210001, name=Hamzeh, [
                Course --> id=01, name=DATABASE               , year=2020, semester=First , section=1, grade=95,
                Course --> id=02, name=ADVANCED PROGRAMING    , year=2020, semester=Second, section=3, grade=92,
                Course --> id=03, name=ARTIFICIAL INTELLIGENCE, year=2020, semester=Summer, section=3, grade=88]
```

Thanks!

Any questions?