Jurusan Teknik Komputer dan Informatika

Politeknik Negeri Bandung

Pertemuan 6 Class Relationship

D3 Kelas 2A/2B

Dosen Pengampu :

Zulkifli Arsyad, Wendi Wirasta, Ardhian Ekawijana

Association

- Association (asosiasi) adalah hubungan yang menyatakan aktivitas di antara dua class yang saling “berkomunikasi”.
- Asosiasi dapat berupa satu-ke-satu, satu-ke-banyak, banyak-ke-satu, banyak-ke-banyak.
- Sebagai contoh, dua class Student dan Mentor memiliki asosiasi sebagai berikut: Student dapat diajari oleh banyak Mentor, sedangkan Mentor dapat mengajari banyak Student. Berikut contoh UML diagramnya:

Aggregation

- **Has-A** relationship.
- Aggregation adalah bentuk khusus dari association yang merepresentasikan hubungan kepemilikan (has-a) antara dua object (tidak harus dari dua class yang berbeda).
- Object pemilik (owner) disebut aggregating object (class-nya disebut aggregating class) dan object yang dimiliki oleh owner disebut aggregated object (class-nya disebut aggregated class).
- Sebagai contoh, antara object Student dan Address terdapat aggregation berupa Student has-a Address. Berikut contoh UML diagramnya:

Composition

- ***Part of*** Relationship
- Composition adalah bentuk khusus dari aggregation, di mana sebuah aggregated object hanya dimiliki oleh suatu aggregating object tertentu. Misalkan object Name hanya dapat dimiliki oleh object Person, bukan object lain.
- Dalam komposisi, kedua entitas bergantung satu sama lain.
- Ketika ada komposisi antara dua entitas, objek yang dikomposisikan tidak dapat eksis tanpa entitas lainnya.
- Berikut contoh UML diagramnya:

dependence

- “uses–a” relationship
- Weakest Class Relationship
  - A class using another class as a parameter passed in a method
  - A class using another inside a method

Dependence Example

```
class Account{
  public void deposit{}
 }
 class Customer{
   public void
makeDeposit(Account acc){
•          acc.deposit();
  //temporary
•    }
```

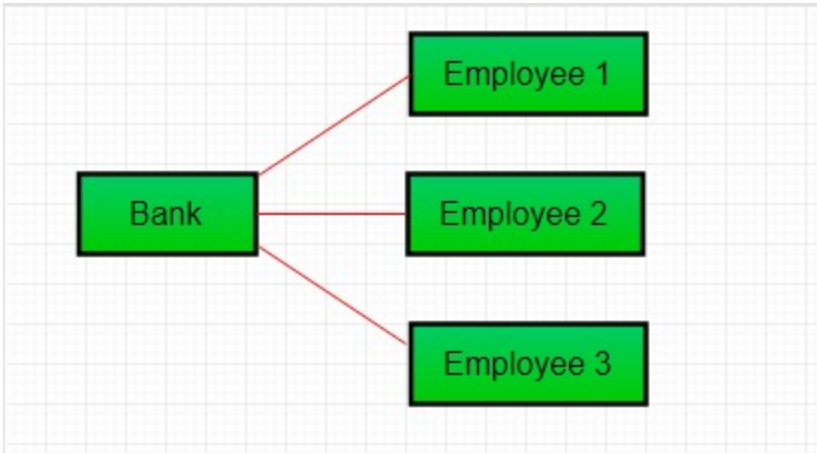
-
```
class Die { public void
Roll() { ... } }
```

```
class Player {
/*Look, I am dependent on Die and it's
Roll method to do my work*/
 public void TakeTurn(Die
die) {
    die.Roll(); ...
  }
}
```

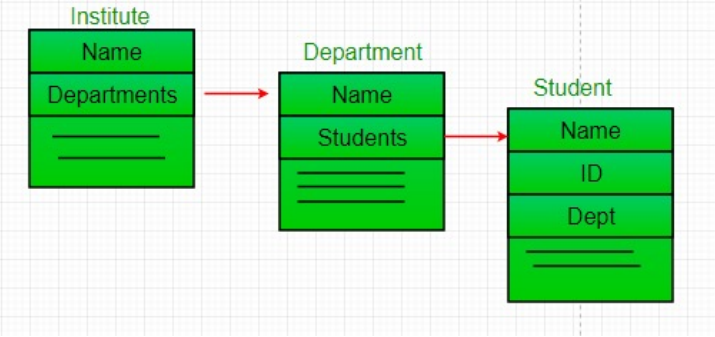
Association Example

```java
// Java program to illustrate the
// concept of Association
import java.io.*;


// class bank
class Bank
{
    private String name;

    // bank name
    Bank(String name)
    {
        this.name = name;
    }

    public String getBankName()
    {
        return this.name;
    }
}
```

```java
// employee class
class Employee
{
    private String name;

    // employee name
    Employee(String name)
    {
        this.name = name;
    }

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


// Association between both the
// classes in main method
class Association
{
    public static void main (String[] args)
    {
        Bank bank = new Bank("BRI");
        Employee emp = new Employee("Ujang");

        System.out.println(emp.getEmployeeName() +
                " is employee of " + bank.getBankName());
    }
}
```



Aggregation Example




```java
// Java program to illustrate
//the concept of Aggregation.
import java.io.*;
import java.util.*;


// student class
class Student
{
    String name;
    int id ;
    String dept;

    Student(String name, int id, String dept)
    {

        this.name = name;
        this.id = id;
        this.dept = dept;

    }
}
```

```java
/* Department class contains list of student
Objects. It is associated with student
class through its Object(s). */
class Department
{

    String name;
    private List<Student> students;
    Department(String name, List<Student> students)
    {

        this.name = name;
        this.students = students;

    }

    public List<Student> getStudents()
    {
        return students;
    }
}
```

Aggregation Example

```
/* Institute class contains list of Department
Objects. It is asoociated with Department
class through its Object(s).*/
class Institute
{

    String instituteName;
    private List<Department> departments;

    Institute(String instituteName,
List<Department> departments)
    {
        this.instituteName = instituteName;
        this.departments = departments;
    }

    // count total students of all departments
    // in a given institute
    public int getTotalStudentsInInstitute()
    {
        int noOfStudents = 0;
        List<Student> students;
        for(Department dept : departments)
        {
            students = dept.getStudents();
            for(Student s : students)
            {
                noOfStudents++;
            }
        }
        return noOfStudents;
    }

}
```

```
/ main method
class GFG
{
    public static void main (String[] args)
    {
        Student s1 = new Student(“Ardhian", 1, "CSE");
        Student s2 = new Student(“Wendi", 2, "CSE");
        Student s3 = new Student(“Zulkifli", 1, "EE");
        Student s4 = new Student("Rahul", 2, "EE");

        // making a List of
        // CSE Students.
        List <Student> cse_students = new ArrayList<Student>();
        cse_students.add(s1);
        cse_students.add(s2);

        // making a List of
        // EE Students
        List <Student> ee_students = new ArrayList<Student>();
        ee_students.add(s3);
        ee_students.add(s4);

        Department CSE = new Department("CSE", cse_students);
        Department EE = new Department("EE", ee_students);

        List <Department> departments = new
ArrayList<Department>();
        departments.add(CSE);
        departments.add(EE);

        // creating an instance of Institute.
        Institute institute = new Institute("BITS",
departments);

        System.out.print("Total students in institute: ");
        System.out.print(institute.getTotalStudentsInInstitute(
));
    }
}
```

Composition

```java
// Java program to illustrate
// the concept of Composition
import java.io.*;
import java.util.*;
// class book
class Book
{
    public String title;
    public String author;
    Book(String title, String author){
        this.title = title;
        this.author = author;
    }
}


// Library class contains
// list of books.
class Library{
   // reference to refer to list of books.
    private final List<Book> books;
    Library (List<Book> books)
    {
        this.books = books;
    }
    public List<Book> getTotalBooksInLibrary(){
       return books;
    }

}
```

```java
/ main method
class GFG
{
    public static void main (String[] args)
    {

        // Creating the Objects of Book class.
        Book b1 = new Book("EffectiveJ Java", "Joshua Bloch");
        Book b2 = new Book("Thinking in Java", "Bruce Eckel");
        Book b3 = new Book("Java: The Complete Reference", "Herbert Schildt");

        // Creating the list which contains the
        // no. of books.
        List<Book> books = new ArrayList<Book>();
        books.add(b1);
        books.add(b2);
        books.add(b3);

        Library library = new Library(books);

        List<Book> bks = library.getTotalBooksInLibrary();
        for(Book bk : bks){

            System.out.println("Title : " + bk.title + " and "
            +" Author : " + bk.author);
        }
    }
}
```

Dalam contoh di atas, perpustakaan dapat memiliki no. buku-buku tentang topik yang sama atau berbeda. Jadi, Jika Perpustakaan dihancurkan maka Semua buku di dalam perpustakaan tersebut akan dimusnahkan. yaitu buku tidak akan ada tanpa perpustakaan. Itu sebabnya komposisi.

Praktikum

- Buat program yang mengimplementasikan Class Relationship terkait Association, Agregation, Composition, Dependence
- Buat laporan dalam bentuk PPT