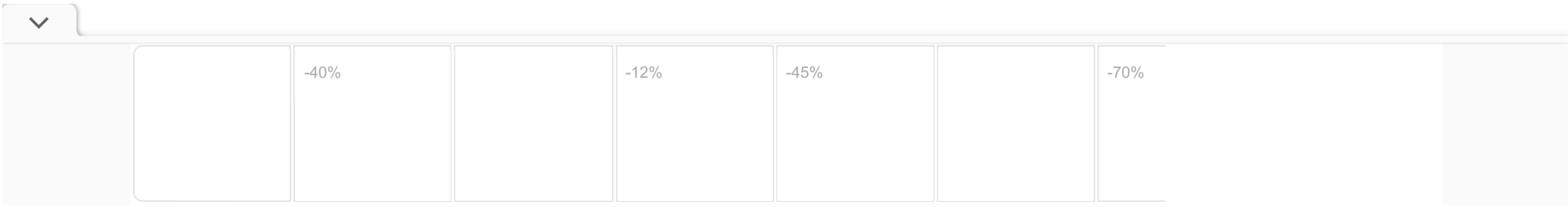
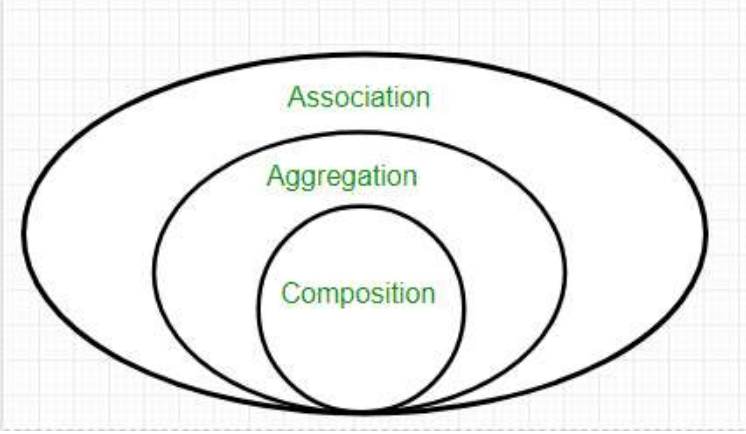
ASSOCIAÇÃO, COMPOSIÇÃO E AGREGAÇÃO EM JAVA





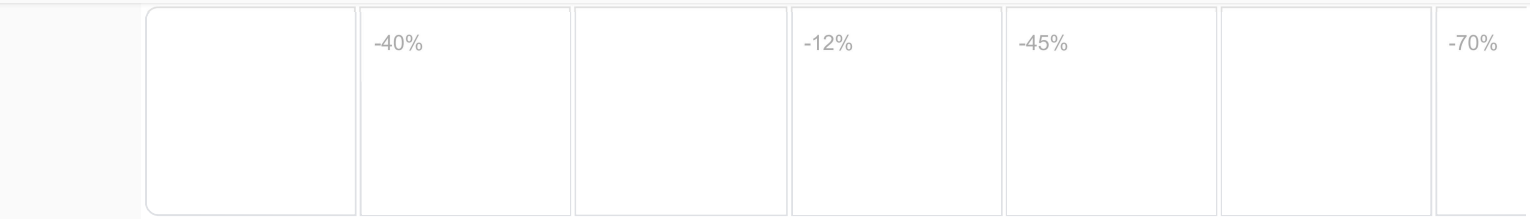
Associação

Associação é a relação entre duas classes distintas que se estabelecem por meio de seus objetos. A associação pode ser um para um, um para muitos, muitos para um, muitos para muitos.

Na programação orientada a objetos, um objeto se comunica com outro objeto para usar a funcionalidade e os serviços fornecidos por esse objeto. **Composição** e **agregação** são as duas formas de associação.

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

// class bank
class Bank
{
    private String name;

    // bank name
```

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

// 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("Axis");
```

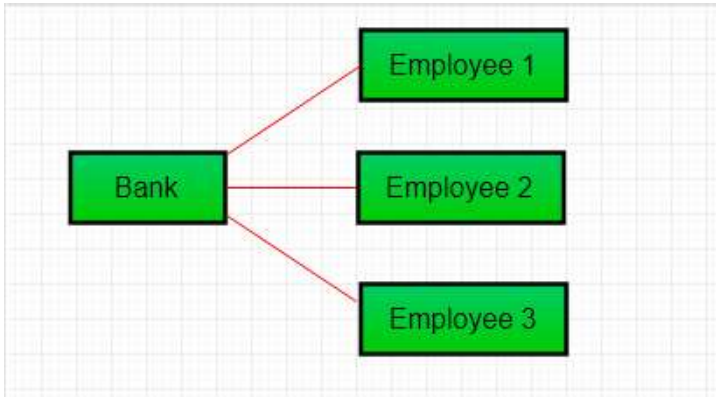


```
    }
}
```

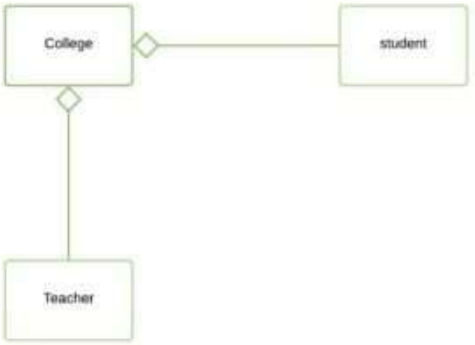
Saída:

```
Neha is employee of Axis
```

No exemplo acima, duas classes separadas Banco e Funcionário são associadas por meio de seus Objetos. O banco pode ter muitos funcionários, portanto, é um relacionamento de um para muitos.



Agregação



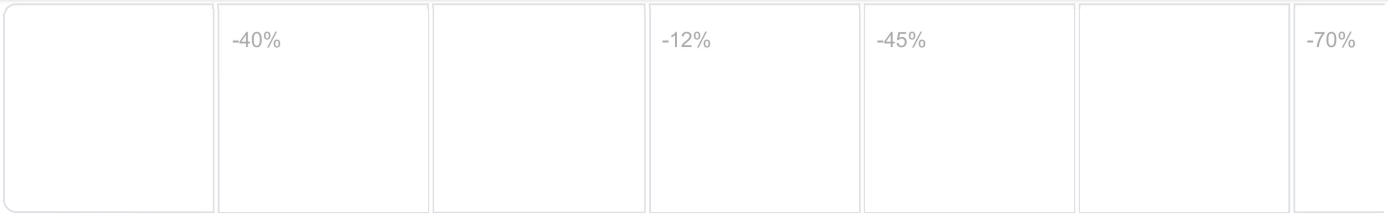
Agregação

É uma forma especial de associação onde:

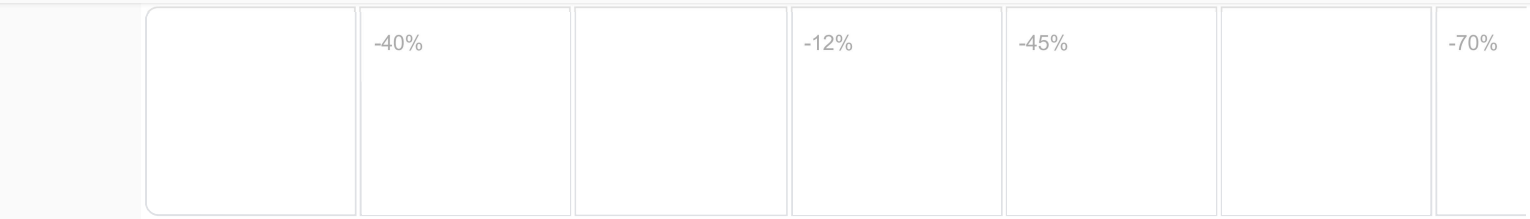
- Representa o relacionamento **Has-A** .
- É uma **associação unidirecional,** ou seja, um **relacionamento de mão** única. Por exemplo, o departamento pode ter alunos, mas vice-versa não é possível e, portanto, de natureza unidirecional.
- Na agregação, **ambas as entradas podem sobreviver individualmente, o** que significa que terminar uma entidade não afetará a outra entidade

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

// student class
class Student
{
    String name;
```

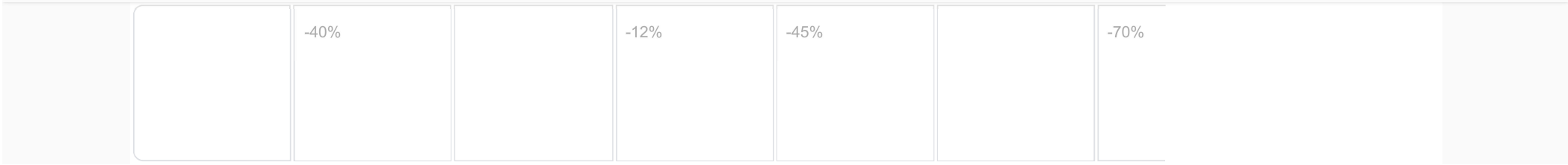
```
        this.name = name;
        this.id = id;
        this.dept = dept;

    }
}

/* 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;
    }
}
```

```
    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
```

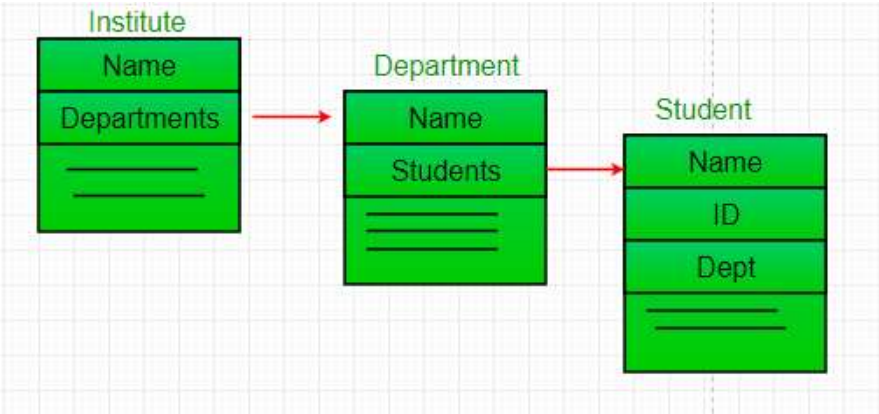


```
        Student s2 = new Student("Priya", 2, "CSE");
        Student s3 = new Student("John", 1, "EE");
        Student s4 = new Student("Rahul", 2, "EE");

        // 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());
    }
}
```



Neste exemplo, existe um Instituto que não possui. de departamentos como CSE, EE. Todo departamento não tem. de estudantes. Então, fazemos uma classe Institute que tem uma referência a Object ou não. de objetos (ou seja, lista de objetos) da classe Departamento. Isso significa que a classe Institute está associada à classe Department por meio de seu (s) objeto (s). E a classe Departamento também tem uma referência a Objetos ou Objetos (isto é, Lista de Objetos) da classe Aluno, o que significa que ela está associada à classe Aluno por meio de seu (s) Objeto (s).

Ele representa um relacionamento **Has-A** .



Quando usamos agregação ??

A reutilização de código é melhor alcançada por agregação.

Composição

Composição

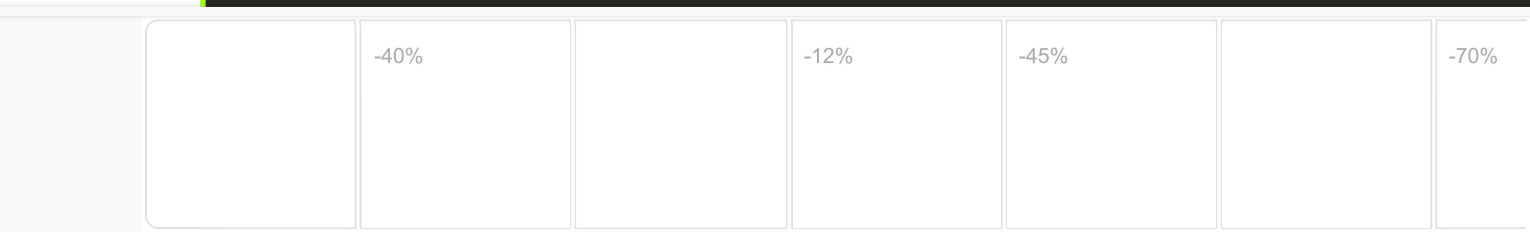
Composição é uma forma restrita de agregação em que duas entidades são altamente dependentes uma da outra.

- Representa **um** relacionamento **parcial** .
- Na composição, ambas as entidades são dependentes uma da outra.
- Quando há uma composição entre duas entidades, o objeto composto **não pode existir** sem a outra entidade.

Vamos pegar o exemplo da **Biblioteca** .

```
// 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;
    }
```

```
{

    // 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;
    }

}

// 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.
```



```
        Library library = new Library(books);

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

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

Saída:

```
Title : EffectiveJ Java and  Author : Joshua Bloch
Title : Thinking in Java and  Author : Bruce Eckel
Title : Java: The Complete Reference and  Author : Herbert Schildt
```

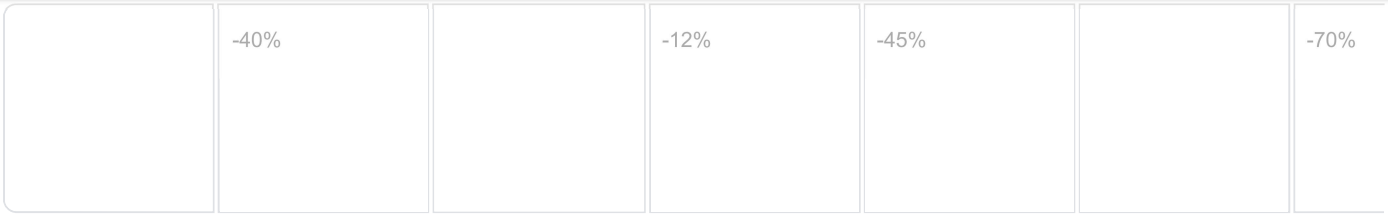
No exemplo acima, uma biblioteca pode não ter. de **livros** sobre assuntos iguais ou diferentes. Portanto, se a biblioteca for destruída, todos os livros dessa biblioteca específica serão destruídos. ou seja, o livro não pode existir sem biblioteca. É por isso que é composição.

Agregação vs Composição

1. **Dependência: a** agregação implica um relacionamento em que o filho **pode existir independentemente** do pai. Por exemplo, Banco e Funcionário, exclua o Banco e o Funcionário ainda existe. enquanto a Composição implica um relacionamento em que a criança **não pode existir independente** do pai. Exemplo: Humano e coração, coração não existe separado de um Humano
2. **Tipo de relacionamento: a** relação de agregação é **"tem um"** e a composição é a relação **"parte de"** .
3. **Tipo de associação: a** composição é uma associação **forte,** enquanto a agregação é uma associação

```
// Composition.

import java.io.*;

// Engine class which will
// be used by car. so 'Car'
// class will have a field
// of Engine type.
class Engine
{
    // starting an engine.
    public void work()
    {

        System.out.println("Engine of car has been started ");

    }

}

// Engine class
final class Car
{

    // For a car to move,
    // it need to have a engine.
    private final Engine engine; // Composition
    //private Engine engine;     // Aggregation

    Car(Engine engine)
```

```
    public void move()
    {

        //if(engine != null)
        {
            engine.work();
            System.out.println("Car is moving ");
        }
    }
}

class GFG
{
    public static void main (String[] args)
    {

        // making an engine by creating
        // an instance of Engine class.
        Engine engine = new Engine();

        // Making a car with engine.
        // so we are passing a engine
        // instance as an argument while
        // creating instace of Car.
        Car car = new Car(engine);
        car.move();

    }
}
```



Em caso de agregação, o Carro também desempenha suas funções por meio de um Motor. mas o motor nem sempre é uma parte interna do carro. Um motor pode ser trocado ou mesmo removido do carro. É por isso que tornamos o campo do tipo Motor não final.

Este artigo é uma contribuição de **Nitsdheerendra** . Se você gosta de GeeksforGeeks e gostaria de contribuir, você também pode escrever um artigo usando write.geeksforgeeks.org ou enviar seu artigo para review-team@geeksforgeeks.org. Veja o seu artigo na página principal do GeeksforGeeks e ajude outros Geeks.

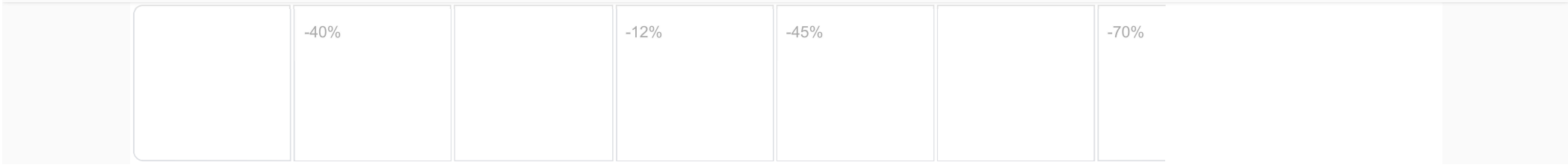
LATEST POSTS

- Experiência de Entrevista Epicor
- Experiência de entrevista Ebix
- Experiência de entrevista com Pickyourtrail (SET 1)
- C-DOT (experiência de entrevista em tempo integral)
- Diferença Máxima de Peso

MOST POPULAR POSTS

Java Interface | Java Object Oriented | Java

- Lidando com linhas e colunas no Pandas DataFrame
- Projetos Python - do iniciante ao avançado
- 7 ideias interessantes de projetos em Python para desenvolvedores intermediários
- As 7 principais ideias de projetos Java para aprimorar as habilidades de programação
- As 10 principais bibliotecas Python para ciência de dados em 2020



❖ ACERVO LIMA

Acervo Lima provides translations of articles published on GeekForGeeks for several languages.

MAIN CATEGORIES

Python

JavaScript

PHP

Java

MORE CATEGORIES

Ruby

CONTACT

Icapuí-CE, Brasil

contact@acervolima.com