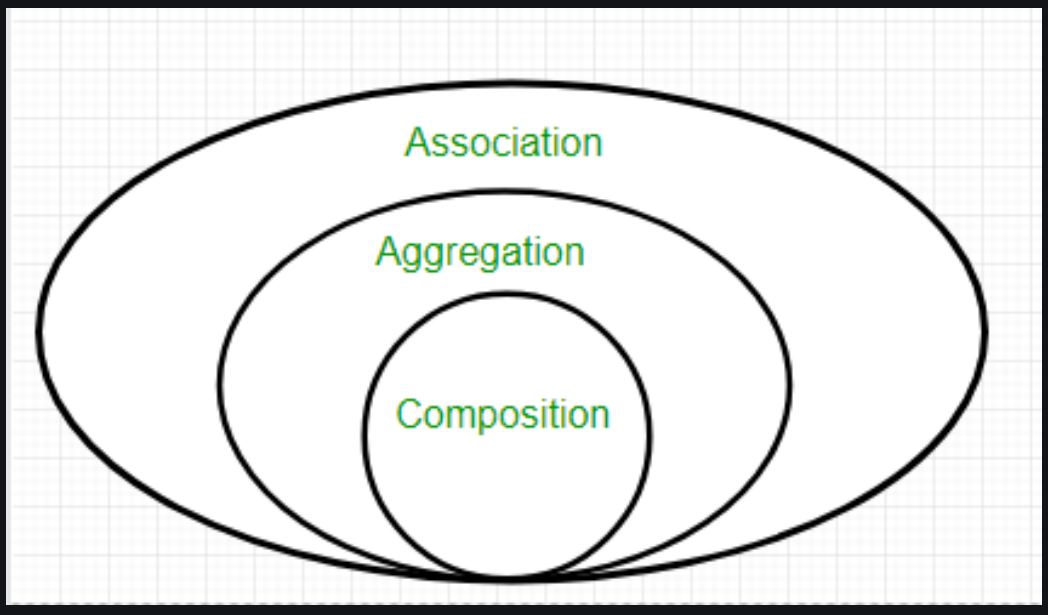
Associação, Composição e Agregação em Java

Nível de dificuldade: Médio • Última atualização: 04 de abril de 2022

Associação é uma relação entre duas classes separadas que se estabelece através 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.



Exemplo:

Java

```
// Java Program to illustrate the
// Concept of Association
```

Comece sua jornada de codificação agora!

Conecte-se

Registro

```
// Bank class
class Bank {

    // Attributes of bank
    private String name;

    // Constructor of this class
    Bank(String name)
    {
        // this keyword refers to current instance itself
        this.name = name;
    }

    // Method of Bank class
    public String getBankName()
    {
        // Returning name of bank
        return this.name;
    }
}

// Class 2
// Employee class
class Employee {
    // Attributes of employee
    private String name;
    // Employee name
    Employee(String name)
    {
        // This keyword refers to current instance itself
        this.name = name;
    }

    // Method of Employee class
    public String getEmployeeName()
    {
        // returning the name of employee
        return this.name;
    }
}

// Class 3
// Association between both the
// classes in main method
class GFG {
```

Comece sua jornada de codificação agora!
Conecte-se
Registro

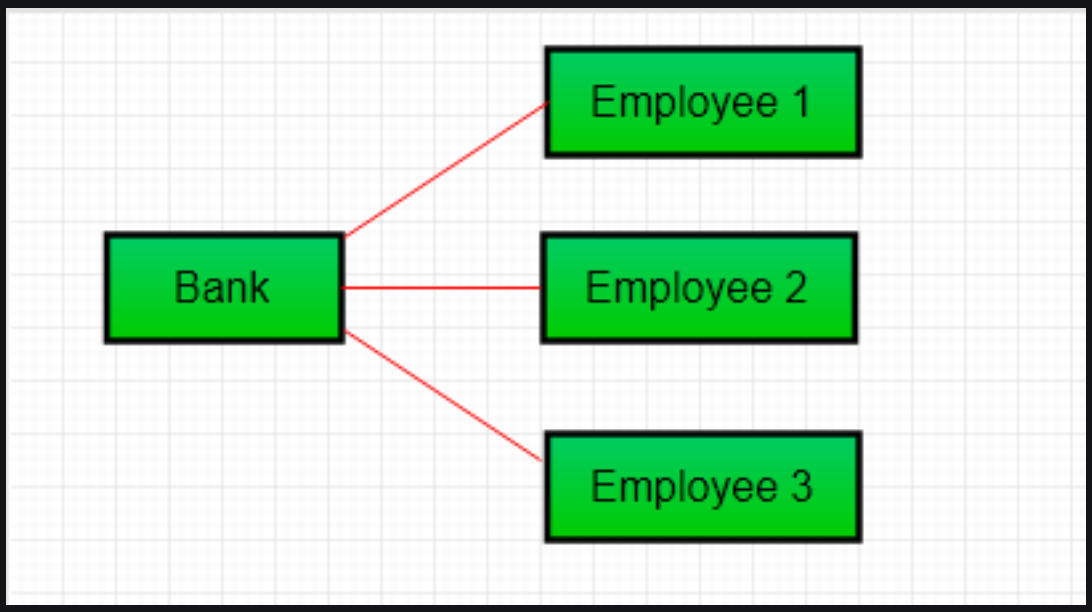
```
        // Creating objects of bank and Employee class
        Bank bank = new Bank("ICICI");
        Employee emp = new Employee("Ridhi");

        // Print and display name and
        // corresponding bank of employee
        System.out.println(emp.getEmployeeName()
                           + " is employee of "
                           + bank.getBankName());
    }
}
```

Resultado

```
Ridhi é funcionário da ICICI
```

Saída Explicação: No exemplo acima, duas classes separadas Bank e Employee são associadas por meio de seus objetos. O banco pode ter muitos funcionários, então é um relacionamento de um para muitos.



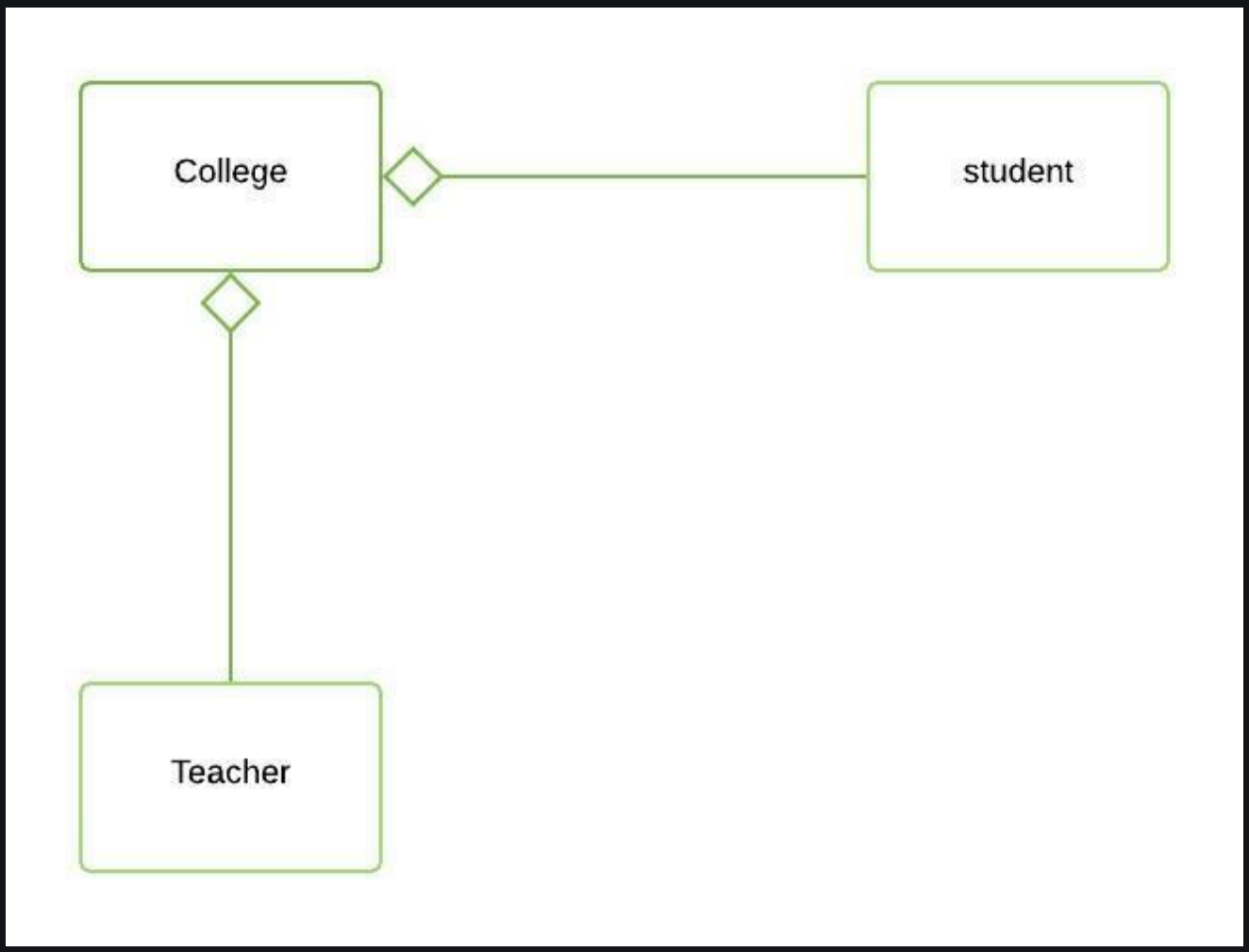
Agregação

É uma forma especial de Associação onde:

Comece sua jornada de codificação agora!
Conecte-se
Registro

natureza unidirecional.

- Na agregação, **ambas as entradas podem sobreviver individualmente** , o que significa que terminar uma entidade não afetará a outra entidade.



Agregação

Exemplo

Java

```
// Java program to illustrate Concept of Aggregation

// Importing required classes
import java.io.*;
import java.util.*;

// Class 1
```

Comece sua jornada de codificação agora!
Conecte-se
Registro

```
    String name;
    int id;
    String dept;

    // Constructor of student class
    Student(String name, int id, String dept)
    {

        // This keyword refers to current instance itself
        this.name = name;
        this.id = id;
        this.dept = dept;
    }
```



```
// It is associated with student class through its Objects
class Department {
    // Attributes of Department class
    String name;
    private List<Student> students;
    Department(String name, List<Student> students)
    {
        // this keyword refers to current instance itself
        this.name = name;
        this.students = students;
    }

    // Method of Department class
    public List<Student> getStudents()
    {
        // Returning list of user defined type
        // Student type
        return students;
    }
}

// Class 3
//  Institute class contains list of Department
// Objects. It is associated with Department
// class through its Objects
class Institute {

    // Attributes of Institute
    String instituteName;
```

Comece sua jornada de codificação agora!
Conecte-se
Registro

```
    {
        // This keyword refers to current instance itself
        this.instituteName = instituteName;
        this.departments = departments;
    }

    // Method of Institute class
    // Counting 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;
    }
}

// Class 4
// main class
class GFG {

    // main driver method
    public static void main(String[] args)
    {
        // Creating object of Student class inside main()
        Student s1 = new Student("Mia", 1, "CSE");
        Student s2 = new Student("Priya", 2, "CSE");
        Student s3 = new Student("John", 1, "EE");
        Student s4 = new Student("Rahul", 2, "EE");

        // Creating a List of CSE Students
        List<Student> cse_students = new ArrayList<Student>();

        // Adding CSE students
        cse_students.add(s1);
        cse_students.add(s2);
```

Comece sua jornada de codificação agora!
Conecte-se
Registro

```
        // Adding EE students
        ee_students.add(s3);
        ee_students.add(s4);

        // Creating objects of EE and CSE class inside
        // main()
        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);

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

        // Display message for better readability
        System.out.print("Total students in institute: ");

        // Calling method to get total number of students
        // in institute and printing on console
        System.out.print(institute.getTotalStudentsInInstitute());
    }
}
```

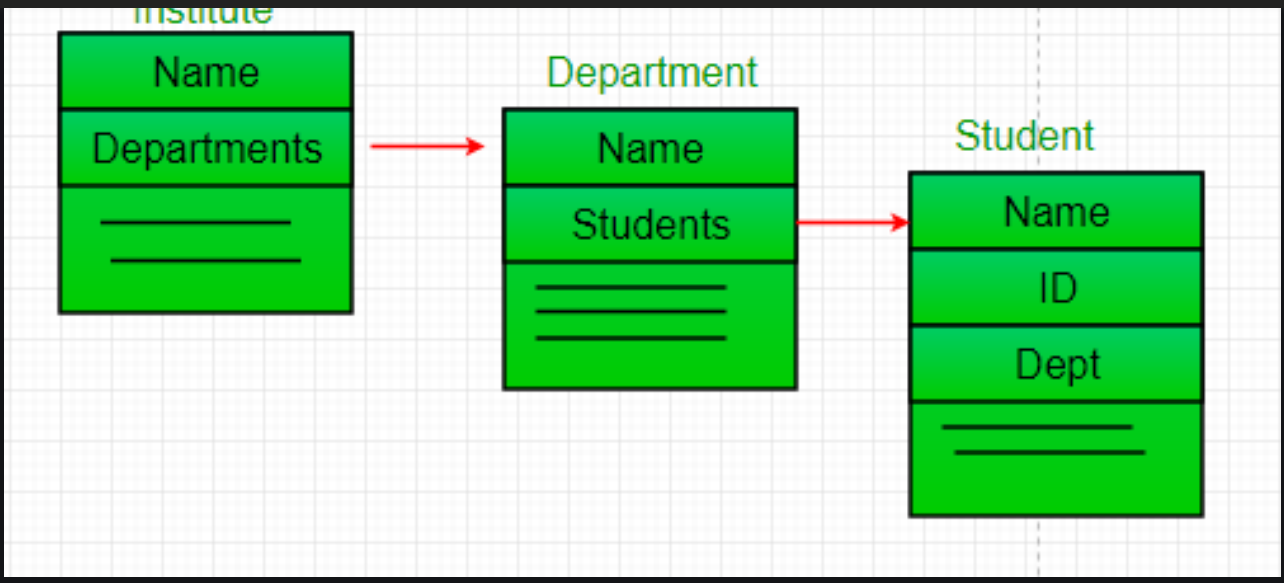
Resultado

```
Total de alunos no instituto: 4
```

Explicação da saída: Neste exemplo, há um Instituto que não possui. de departamentos como CSE, EE. Cada 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 Department. 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 (ou seja, Lista de Objetos) da classe Aluno, o que significa que está associada à classe Aluno por meio de seu(s) Objeto(s).

presenta um relacionamento **Tem-A** . No exemplo acima: Aluno **tem um** nome. Aluno **Tem-A** ID. Aluno **Tem-A** Dept. Departamento **Tem-A** Alunos conforme ilustrado na mídia

Comece sua jornada de codificação agora!
Conecte-se
Registro

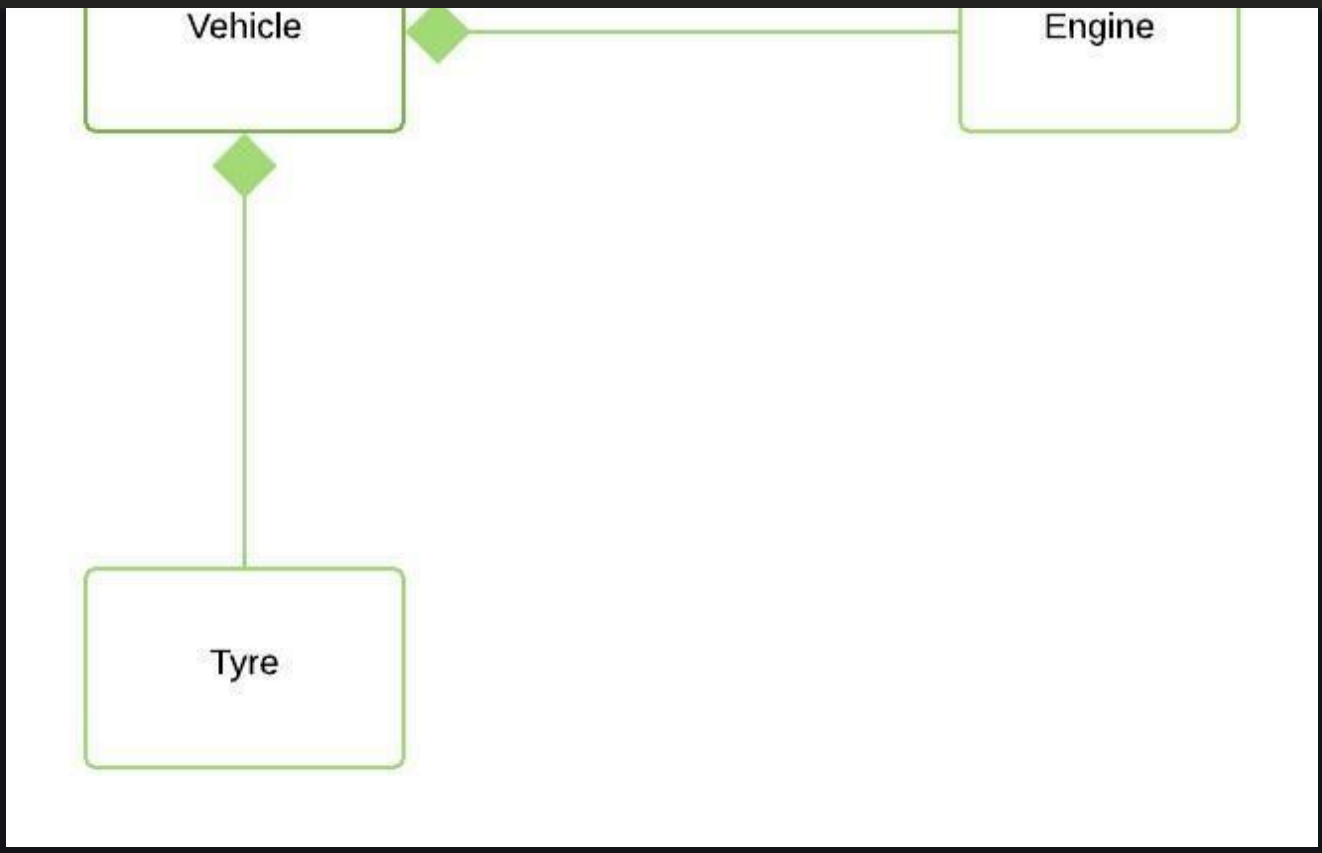


Quando usamos Agregação ??

A reutilização de código é melhor alcançada por agregação.

Conceito 3: **Composição**

Comece sua jornada de codificação agora!
Conecte-se
Registro



Composição

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

- Representa **parte do** relacionamento.
- 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.

Biblioteca de **exemplo**

Java

```
// Java program to illustrate
// the concept of Composition

// Importing required classes
import java.io.*;
import java.util.*;
```

Comece sua jornada de codificação agora!
Conecte-se
Registro

```
    // Attributes of book
    public String title;
    public String author;

    // Constructor of Book class
    Book(String title, String author)
    {

        // This keyword refers to current instance itself
        this.title = title;
        this.author = author;
    }
}

// Class 2
class Library {

    // Reference to refer to list of books
    private final List<Book> books;

    // Library class contains list of books
    Library(List<Book> books)
    {

        // Referring to same book as
        // this keyword refers to same instance itself
        this.books = books;
    }

    // Method
    // To get total number of books in library
    public List<Book> getTotalBooksInLibrary()
    {

        return books;
    }
}

// Class 3
// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {
```

Comece sua jornada de codificação agora!
Conecte-se
Registro

```
            = 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 number of books
        List<Book> books = new ArrayList<Book>();

        // Adding books
        // using add() method
        books.add(b1);
        books.add(b2);
        books.add(b3);

        Library library = new Library(books);

        // Calling method to get total books in library
        // and storing it in list of user0defined type -
        // Books
        List<Book> bks = library.getTotalBooksInLibrary();

        // Iterating over books using for each loop
        for (Book bk : bks) {

            // Printing the title and author name of book on
            // console
            System.out.println("Title : " + bk.title
                               + " and "
                               + " Author : " + bk.author);
        }
    }
}
```

Resultado

```
Título: EffectiveJ Java e Autor: Joshua Bloch
Título: Pensando em Java e Autor: Bruce Eckel
Título: Java: A Referência Completa e Autor: Herbert Schildt
```

Output explanation: In the above example, a library can have no. of **books** on the same or different subjects. So, If Library gets destroyed then All books within that particular

Comece sua jornada de codificação agora!
Conecte-se
Registro

Aggregation vs Composition

1. Dependency: Aggregation implies a relationship where the child **can exist independently** of the parent. For example, Bank and Employee, delete the Bank and the Employee still exist. whereas Composition implies a relationship where the child **cannot exist independent** of the parent. Example: Human and heart, heart don't exist separate to a Human

2. Type of Relationship: Aggregation relation is **"has-a"** and composition is **"part-of"** relation.

3. Type of association: Composition is a **strong** Association whereas Aggregation is a **weak** Association.

Example:

Java

```
// Java Program to Illustrate Difference between
// Aggregation and Composition

// Importing I/O classes
import java.io.*;

// Class 1
// Engine class which will
// be used by car. so 'Car'
// class will have a field
// of Engine type.
class Engine {

    // Method to starting an engine
    public void work()
    {

        // Print statement whenever this method is called
        System.out.println(
            "Engine of car has been started ");
    }
}

// Class 2
```

Comece sua jornada de codificação agora!
Conecte-se
Registro

```
    // it needs to have an engine.

    // Composition
    private final Engine engine;

    // Note: Uncommented part refers to Aggregation
    // private Engine engine;

    // Constructor of this class
    Car(Engine engine)
    {

        // This keywords refers to same instance
        this.engine = engine;
    }

    // Method
    // Car start moving by starting engine
    public void move()
    {

        // if(engine != null)
        {
            // Calling method for working of engine
            engine.work();

            // Print statement
            System.out.println("Car is moving ");
        }
    }
}

// Class 3
// Main class
class GFG {

    // Main driver method
    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 argument
```

Comece sua jornada de codificação agora!
Conecte-se
Registro

```
        // move() method inside main()
        car.move();
    }
}
```

Output

```
Engine of car has been started
Car is moving
```

In case of aggregation, the Car also performs its functions through an Engine. but the Engine is not always an internal part of the Car. An engine can be swapped out or even can be removed from the car. That's why we make The Engine type field non-final.

This article is contributed by **Nitsdheerendra**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

♡ Like

Previous Next

▲