

```

Aggregation (Weak Reference):
  Aggregation is like another form of association between classes that represents a "HAS-A" relationship, but with a weaker association than composition.

  In aggregation, one contains an object of another class, but the contained object can exist independently of the container. If the container object is destroyed, the contained object can still exist.

package com.cnit.aggregation;

public class College {
    private String collegeName;
    private List<Student> studentList;
    public College(String collegeName, List<Student> studentList) {
        super();
        this.collegeName = collegeName;
        this.studentList = studentList;
    }
    @Override
    public String toString() {
        return "College [collegeName=" + collegeName + ", studentList=" + studentList + "]";
    }
}

package com.cnit.aggregation;

public class Student {
    private int studentId;
    private String studentName;
    private String studentAddress;
    private College college; //NOVA Relation
    public Student(int studentId, String studentName, String studentAddress, College college) {
        super();
        this.studentId = studentId;
        this.studentName = studentName;
        this.studentAddress = studentAddress;
        this.college = college;
    }
    @Override
    public String toString() {
        return "Student [studentId=" + studentId + ", studentName=" + studentName + ", studentAddress=" + studentAddress + ", college=" + college + "]";
    }
}

```

```

package com.ravi.aggregation;
public class Aggregation {
    void main()
    {
        College c1 = new College("WIT", "Vellore");
        Student s1 = new Student("Karthi", "Anvesh", c1);
        s1.print();
        Student s2 = new Student("Karthi", "A Nagar", c1);
        s2.print();
    }
}

Create a Shopping Hall application project by using Method Overriding Concept to dispaly different kinds of discount given to their customers like PriceCustomer and VPCustomer (No discount for General Customer)
Summary :
-----
[Super class : Customer]
-----  

    -> PriceCustomer, VPCustomer, PriceCustomer, VPCustomer  

    overridden the calculateBill() method to implement different discount rules as well as overriding the print() method for printing the sale.
Coding Requirements
-----
Create a new B2C Class Customer
Fields :  

    name : String  

    price : protected  

    total : double protected
use a parameterized constructor to initialize the Fields, In this constructor provide the name and price, if name is null or empty (see test cases for more details)
error message, If name is null or empty (see test cases for more details)
1) Method Name : calculateBill()
Parameter : One Parameter of type double var arg [Double... prices]
Access modifier : public
In this method using var arg receive item price, Give an error message and exit. If there is no item price, then calculate the total bill by adding all the item price received through var arg.
2) Method Name : printDetails()
Parameter : None
Return Type : void
Access modifier : public
In this method print customer name, total cost and no discount for general customer.
Create another B2C Class GeneralCustomer which is the sub class of Customer
Fields : no fields
Take a parameterized constructor to initialize super class properties.
Method :
1) Method Name : calculateBill()
Parameter : One Parameter of type double var arg [Double... prices]
Access modifier : public
In this method give 10% discount on total bill, calculate final Bill after discount amount and find result (See the test cases)
2) Method Name : printDetails()
Parameter : None
Return Type : void
Access modifier : public
In this method give 10% discount on total bill, calculate final Bill after discount amount and find result (See the test cases)

```