# Programming Test Results (With Test Cases)

## Result Summary

| Field | Value |
|---|---|
| **Test ID** | 40150 |
| **Student ID** | 29195 |
| **Programs (with test cases)** | 3 |
| **Total Test Cases** | 9 |
| **Test Cases Passed** | 9 |
| **Fully Passed Programs** | 3 |
| **Partially Passed Programs** | 0 |
| **Failed Programs** | 0 |
| **Overall % (with test cases)** | 100.00% |
| **Grade** | Outstanding |

## Programs With Test Cases

| # | Program Name | Total TC | Passed | Success Rate | Score /10 | Submitted At | Attempts |
|---|---|---|---|---|---|---|---|
| 1 | Customer and Orders Management | 3 | 3 | 100.0% | 10 | 19/11/2025, 13:33:17 | 0 |
| 2 | Car Has An Engine | 3 | 3 | 100.0% | 10 | 19/11/2025, 12:28:28 | 0 |
| 3 | Modeling Employee and Address Details | 3 | 3 | 100.0% | 10 | 19/11/2025, 12:03:58 | 0 |

## Program Details (With Test Cases)

## Program 1: Customer and Orders Management

| | | |
|---|---|---|
| **Languages:** | Java | |
| **Score (010):** | 10 / 10 | |
| **Test Case Summary:** | Total: 3 | Passed: 3 |
| | Failed: 0 | Success: 100.0% |
| **Attempts:** | 0 | |
| **Submitted At:** | 19/11/2025, 13:33:17 | |

**Description:**

Task Description :-
-------------------
Create a program that manages customers and their orders using Aggregation.

Create a BLC class Order.

A class to represent an order with details like order ID, product name, and price.

Field Declarations :-
------------------------
```
private String orderId;     -> Stores the order ID
private String itemName;     -> Stores the name of the product ordered
private double price;        -> Stores the price of the product
```

Constructor Declaration :-
---------------------------
-> public Order(String orderId, String itemName, double price) A constructor to initialize the order ID, item name, and price.

getter() Methods Declaration :-
--------------------------------
-> public String getOrderId();
-> public String getProductName();
-> public double getPrice();

toString() Method Declaration :-
---------------------------------
@Override
public String toString();
A method to return a string representation of the order.

Create a class called Customer.

A class to represent a customer with details

Field Declarations :-
------------------------
-> private String name; // Stores the customer's name
-> private String email; // Stores the customer's email
-> private String address; //Stroes the customer's address
-> private long mobileNo; //Stroes the customer's mobile number
-> private Order order; // Stores the order object

Constructor Declaration :-
---------------------------
->Take a parameterized constructor to initialize all the fields.

toString Method Declaration :-
------------------------------
@Override
public String toString();
A method to return a string representation of the customer, including order details.

Create an ELC class Zomoto class with the following tasks:

-> Create a main method.
-> Create an Order object.
-> Create a Customer object that includes the Order.
-> Display the customer information.

**Constraints:** -

**Sample Input:** O101 Pizza 499.50 Rahul Sharma rahul@gmail.com Delhi 9876543210

**Sample Output:** Customer Details: Name: Rahul Sharma Email: rahul@gmail.com Address: Delhi Mobile: 9876543210 Order Details: Order ID: O101 Product Name: Pizza Price: 499.5

**Explanation:** -

## Solution Code

```
import java.util.*;
public class Zomato{
    public static void main(String [] args){
    Scanner sc = new Scanner(System.in);

   String id = sc.nextLine();
```

```
    String itemName = sc.nextLine();
    double price = Double.parseDouble(sc.nextLine());

    Order d = new Order(id,itemName,price);

    String name = sc.nextLine();
    String email = sc.nextLine();
    String address = sc.nextLine();
    long mobNo = Long.parseLong(sc.nextLine());
    Customer c = new  Customer(name,email,address,mobNo,d);
System.out.println(c.toString());

    }

}

class Order{
    private String orderId;
    private String itemName;
    private double productPrice;

    Order(String orderId,String itemName,double productPrice){
        this.orderId = orderId;
        this.itemName = itemName;
        this.productPrice = productPrice;

    }
    public String getOrderId(){
        return this.orderId;
    }

      public String getItemName(){
        return this.itemName;
      }
      public double getPrice(){
        return this.productPrice;
      }
      public String toString(){
        return "Order Details:"+"\n"+"Order ID: "+this.orderId+"\n"+"Product Name: 
"+this.itemName+"\n"+"Price: "+this.productPrice;
      }
}
```

```
class Customer{
    private String name;
    private String email;
    private String address;
    private long mobileNo;
    private Order order;

    Customer(String name,String email,String address,long mobileNo,Order order){
        this.name = name;
        this.email=email;
        this.address=address;
        this.mobileNo = mobileNo;
        this.order = order;
    }
    public String toString(){
        return "Customer Details:"+"\n"+"Name: "+this.name+"\n"+"Email:
"+this.email+"\nAddress: "+this.address+"\nMobile:
"+this.mobileNo+"\n\n"+order.toString();
    }


}
```

## Program 2: Car Has An Engine

| | | |
|---|---|---|
| **Languages:** | Java | |
| **Score (010):** | 10 / 10 | |
| **Test Case Summary:** | Total: 3 | Passed: 3 |
| | Failed: 0 | Success: 100.0% |
| **Attempts:** | 0 | |
| **Submitted At:** | 19/11/2025, 12:28:28 | |

**Description:**

Create a Car class that contains an Engine object as a composition. The Engine class should store details about the engine, while the Car class should include details about the car along with validation checks for numeric values.

Create a BLC class Engine.

A class to represent an engine with a model and capacity attributes.

Field Declarations :-
-------------------------

private String model;         > Stores the model of the engine.
private int engineCapacity;     > Stores the capacity of the engine (must be positive).

Constructor Declaration :-
---------------------------
public Engine(String model, int engineCapacity) -> Initializes the engine model and capacity.

Note:-
------
If engineCapacity <= 0, print "Error Invalid Input" and terminate object creation.

Getter Methods :-
------------------
-> public String getModel()
-> public int getEngineCapacity()
Returns model and capacity.

toString() method :-
--------------------
@Override
public String toString() -> Returns a string representation of the engine in the format :-

Engine Model: [model], Engine Capacity: [capacity]cc

Create another BLC class called Car.

A class to represent a car that contains an engine.

variable Declarations :-
-------------------------
private String make;     > Stores the car's make.
private String model;     > Stores the car's model.
private int year;     > Stores the car's manufacturing year (must be positive).
private final Engine engine;     > Stores the engine object.

Constructor Declaration :-
---------------------------
public Car(String make, String model, int year) -> Initializes make, model, and year.

Use composition logic to create Engine object.

Note :-

-------

If year <= 0, print "Error Invalid Input" and terminate object creation.

toString() method :-

--------------------

@Override

public String toString() -> Returns a string representation of the car in the format :-

Create an ELC class CompositionDemo with main method to test this application.

**Constraints:** -

**Sample Input:** TurboDiesel 2200 Mahindra XUV700 2022

**Sample Output:** Engine Model: TurboDiesel, Engine Capacity: 2200cc Car Make: Mahindra, Car Model: XUV700, Year: 2022 Engine Details -> Engine Model: TurboDiesel, Engine Capacity: 2200cc

**Explanation:** -

## Solution Code

```
import java.util.*;
public class CompositionDemo{
    public static void main(String [] args){
        Scanner sc = new Scanner (System.in);
          String engModel = sc.nextLine();
          int engCap =Integer.parseInt(sc.nextLine());

          String carMke = sc.nextLine();
          String carmod = sc.nextLine();
          int year = Integer.parseInt(sc.nextLine());
          Car c = new Car(engModel,engCap,carMke,carmod,year);
          System.out.println(c.toString());
    }
}
class Engine{
    private String model;
    private int ingineCapacity;
    Engine(String model,int ingineCapacity){
        this.model= model;
        this.ingineCapacity = ingineCapacity;
        if(ingineCapacity<=0){
            System.out.println("Error Invalid Input");
            System.exit(0);
        }
    }
    public String getModel(){
        return this.model;

    }
    public int getIEngineCapacity(){
        return this.ingineCapacity;
    }

    public String toString(){
        return "Engine Details -> Engine Model: "+this.getModel()+", Engine Capacity: 
"+this.getIEngineCapacity()+"cc";
    }

}
class Car {
    private String carMake;
    private String carModel;
    private int year;
    private Engine engine;
  Car(String model ,int capacity,String carMake,String carModel,int year){
    this.carMake = carMake;
    this.carModel = carModel;
    this.year = year;
    if(year<=0){
        System.out.println("Error Invalid Input");
        System.exit(0);
    }
    this.engine = new Engine(model,capacity);
  }
  public String toString(){
    return "Car Make: "+this.carMake+", Car Model: "+this.carModel+", Year: 
"+this.year+"\n"+engine.toString();
  }

}
```

## Program 3: Modeling Employee and Address Details

|  |  |
|---|---|
| **Languages:** | Java |
| **Score (010):** | 10 / 10 |

|  |  |  |
|---|---|---|
| **Test Case Summary:** | Total: 3 | Passed: 3 |
|  | Failed: 0 | Success: 100.0% |

**Attempts:** 0

**Submitted At:** 19/11/2025, 12:03:58

**Description:**

Create a BLC class Address

Fields :-
------------
Implement a class with private fields for street and city.

Constructor :-
--------------
Provide a constructor to initialize these fields.

getter() & toString() method :-
-------------------------------
Implement getter methods and a toString() method to display the address.

Create another BLC class Person.

Fields :-
--------
Implement a class with private fields for name and an Address object.

Constructor :-
--------------
Provide a constructor to initialize name and address.

method getPersonDetails()
--------------------------
Implement a method to display the person's name and address.

Create an ELC class called Test

Implement a main method :-
---------------------------
-> Instantiate an Address object with sample data.
-> Instantiate a Person object using the Address object created earlier.
-> Print the person's details using the method in the Person class.

| | |
|---|---|
| **Constraints:** | - |
| **Sample Input:** | MG Road Bengaluru Rahul Sharma |
| **Sample Output:** | Person Name: Rahul Sharma Address: Street - MG Road, City - Bengaluru |
| **Explanation:** | - |

## Solution Code

```
import java.util.*;
public class Test{
public static void main(String [] args){
    Scanner sc = new Scanner(System.in);

     String street = sc.nextLine();
     String city = sc.nextLine();
     String name = sc.nextLine();

     Address  ad = new Address(street,city);

     Person p = new Person(name,ad);
     System.out.println(p.getPerson());
}
}
class Address{
    private String street;
    private String city;
    Address(String street,String city){
        this.street = street;
        this.city= city;
    }
    public void setStreet(String street){
        this.street=street;
    }
    public  String getStreet(){
       return this.street;
    }
    public void setCity(String  city){
        this.city=city;
    }
    public String getCity(){
        return this.city;
    }
    public String toString(){
        return "Address: Street - "+this.street+", City - "+this.city;
    }
}

class Person{
    private String name;
    private Address address;

     Person(String name,Address address){
        this.name=name;
        this.address= address;
     }
     public String getPerson(){
        return "Person Name: "+this.name+"\n"+""+address.toString();
     }
}
```