```
namespace backendDay5
{
    0 references
    internal class Program
    {
        5 references
        abstract class Vehicle
        {
            2 references
            public string Manufacturer { get; set; }
            2 references
            public string Model { get; set; }
            2 references
            public int ProductionYear { get; set; }
            2 references
            public int MaxSeatingCapacity { get; set; }

            4 references
            public abstract void DescribePerformance();

            2 references
            public Vehicle(string manufacturer, string model, int productionYear, int maxSeatingCapacity)
            {
                if (productionYear < 1886 || productionYear > DateTime.Now.Year)
                {
                    throw new ArgumentException("Invalid production year. Cars didn't exist before 1886.");
                }

                if (maxSeatingCapacity <= 0)
                {
                    throw new ArgumentException("Seating capacity must be greater than zero.");
                }

                Manufacturer = manufacturer;
                Model = model;
                ProductionYear = productionYear;
                MaxSeatingCapacity = maxSeatingCapacity;
            }

            6 references
            public virtual void displayInfo()
            {
                Console.WriteLine($"Manufacturer: {Manufacturer}");
                Console.WriteLine($"Model: {Model}");
                Console.WriteLine($"Production Year: {ProductionYear}");
                Console.WriteLine($"Max Seating Capacity: {MaxSeatingCapacity}");
            }
        }
```

```
3 references
class Car : Vehicle
{
    2 references
    public int NumberOfDoors { get; set; }
    3 references
    public bool HasAutomaticTransmission { get; set; }
    3 references
    public int HorsePower { get; set; }

    1 reference
    public Car(string manufacturer, string model, int productionYear, int maxSeatingCapacity,
               int numberOfDoors, bool hasAutomaticTransmission, int horsePower)
        : base(manufacturer, model, productionYear, maxSeatingCapacity)
    {
        if (numberOfDoors <= 0)
        {
            throw new ArgumentException("Number of doors must be greater than 0.");
        }

        if (horsePower <= 0)
        {
            throw new ArgumentException("Horsepower must be a positive number.");
        }

        NumberOfDoors = numberOfDoors;
        HasAutomaticTransmission = hasAutomaticTransmission;
        HorsePower = horsePower;
    }

    2 references
    public override void DescribePerformance()
    {
        Console.WriteLine($"Performance: {HorsePower} HP, " +
                          (HasAutomaticTransmission ? "Automatic" : "Manual") + " Transmission");
    }

    4 references
    public override void displayInfo()
    {
        base.displayInfo();
        Console.WriteLine($"Number of Doors: {NumberOfDoors}");
        Console.WriteLine($"Transmission: {(HasAutomaticTransmission ? "Automatic" : "Manual")}");
        Console.WriteLine($"Horsepower: {HorsePower}");
        DescribePerformance();
    }
```

```
3 references
class Motorcycle : Vehicle
{
    3 references
    public int EngineCapacity { get; set; }
    3 references
    public bool HasSidecar { get; set; }

    1 reference
    public Motorcycle(string manufacturer, string model, int productionYear, int maxSeatingCapacity,
                      int engineCapacity, bool hasSidecar)
        : base(manufacturer, model, productionYear, maxSeatingCapacity)
    {
        if (engineCapacity < 50 || engineCapacity > 2000)
        {
            throw new ArgumentException("Engine capacity must be between 50cc and 2000cc.");
        }

        if (maxSeatingCapacity > 3)
        {
            throw new ArgumentException("Motorcycles typically can't seat more than 3 people.");
        }

        EngineCapacity = engineCapacity;
        HasSidecar = hasSidecar;
    }

    2 references
    public override void DescribePerformance()
    {
        Console.WriteLine($"Performance: {EngineCapacity}cc engine" +
                          (HasSidecar ? " with sidecar" : ""));
    }

    4 references
    public override void displayInfo()
    {
        base.displayInfo();
        Console.WriteLine($"Engine Capacity: {EngineCapacity}cc");
        Console.WriteLine($"Has Sidecar: {(HasSidecar ? "Yes" : "No")}");
        DescribePerformance();
    }
}
```

```
0 references
static void Main(string[] args)
{
    Console.WriteLine("=== VEHICLE ENTRY PROGRAM ===\n");

    // car
    Console.WriteLine("Enter Car Information:");
    string carManufacturer = ReadNonEmpty("Manufacturer");
    string carModel = ReadNonEmpty("Model");
    int carYear = ReadIntInRange("Production Year", 1886, DateTime.Now.Year);
    int carSeats = ReadIntGreaterThan("Max Seating Capacity", 0);
    int carDoors = ReadIntGreaterThan("Number of Doors", 0);
    bool carAuto = CheckBool("Has automatic transmission (true/false)");
    int carHP = ReadIntGreaterThan("Horsepower", 0);

    Car car = new Car(carManufacturer, carModel, carYear, carSeats, carDoors, carAuto, carHP);

    //rotoms
    Console.WriteLine("\nEnter Motorcycle Information:");
    string motoManufacturer = ReadNonEmpty("Manufacturer");
    string motoModel = ReadNonEmpty("Model");
    int motoYear = ReadIntInRange("Production Year", 1886, DateTime.Now.Year);
    int motoSeats = ReadIntInRange("Max Seating Capacity", 1, 3);
    int engineCC = ReadIntInRange("Engine Capacity (cc)", 50, 2000);
    bool hasSidecar = CheckBool("Has sidecar (true/false)");

    Motorcycle motorcycle = new Motorcycle(motoManufacturer, motoModel, motoYear, motoSeats, engineCC, hasSidecar);

    Console.WriteLine("\n=== CAR DETAILS ===");
    car.displayInfo();

    Console.WriteLine("\n=== MOTORCYCLE DETAILS ===");
    motorcycle.displayInfo();
}
```

```
4 references
static string ReadNonEmpty(string prompt)
{
    string input;
    do
    {
        Console.Write($"{prompt}: ");
        input = Console.ReadLine();
        if (string.IsNullOrWhiteSpace(input))
            Console.WriteLine("Input must not be empty.");
    } while (string.IsNullOrWhiteSpace(input));
    return input;
}

3 references
static int ReadIntGreaterThan(string prompt, int min)
{
    int value;
    bool valid;
    do
    {
        Console.Write($"{prompt}: ");
        valid = int.TryParse(Console.ReadLine(), out value) && value > min;
        if (!valid)
            Console.WriteLine($"Value must be greater than {min}.");
    } while (!valid);
    return value;
}

4 references
static int ReadIntInRange(string prompt, int min, int max)
{
    int value;
    bool valid;
    do
    {
        Console.Write($"{prompt} ({min}-{max}): ");
        valid = int.TryParse(Console.ReadLine(), out value) && value >= min && value <= max;
        if (!valid)
            Console.WriteLine($"Value must be between {min} and {max}.");
    } while (!valid);
    return value;
}
```

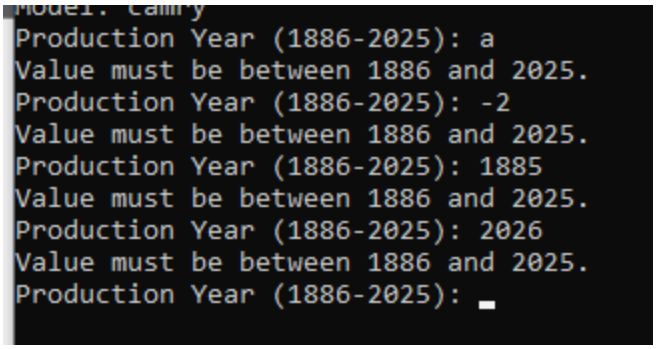
```
2 references
static bool CheckBool(string prompt)
{
    string input;
    do
    {
        Console.Write($"{prompt}: ");
        input = Console.ReadLine().ToLower();
        if (input == "true" || input == "yes") return true;
        if (input == "false" || input == "no") return false;
        Console.WriteLine("Enter 'true' or 'false'.");
    } while (true);
}
}
```

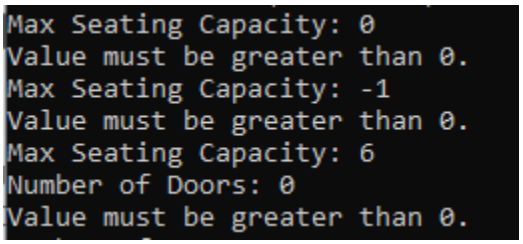
OUTPUTS

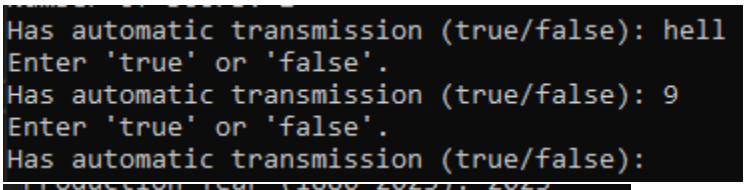
```
=== VEHICLE ENTRY PROGRAM ===

Enter Car Information:
Manufacturer: Toyota
Model: Camry
Production Year (1886-2025): 2022
Max Seating Capacity: 5
Number of Doors: 4
Has automatic transmission (true/false): true
Horsepower: 203

Enter Motorcycle Information:
Manufacturer: Honda
Model: CBR150R
Production Year (1886-2025): 2023
Max Seating Capacity (1-3): 2
Engine Capacity (cc) (50-2000): 150
Has sidecar (true/false): false

=== CAR DETAILS ===
Manufacturer: Toyota
Model: Camry
Production Year: 2022
Max Seating Capacity: 5
Number of Doors: 4
Transmission: Automatic
Horsepower: 203
Performance: 203 HP, Automatic Transmission

=== MOTORCYCLE DETAILS ===
Manufacturer: Honda
Model: CBR150R
Production Year: 2023
Max Seating Capacity: 2
Engine Capacity: 150cc
Has Sidecar: No
Performance: 150cc engine
```

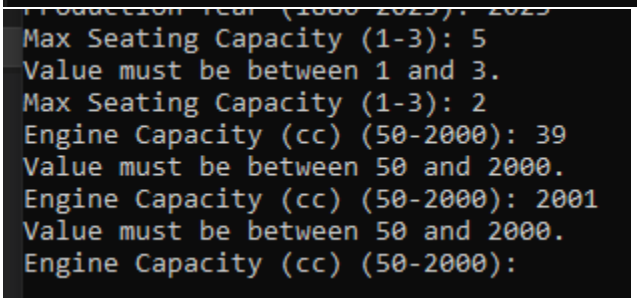
ERROR HANDLINGS

```
=== VEHICLE ENTRY PROGRAM ===

Enter Car Information:
Manufacturer:
Input must not be empty.
Manufacturer: Toyota
Model:
Input must not be empty.
Model:
```









EXPLANATION

In my program, I used inheritance by creating a base class called Vehicle, which holds general information like manufacturer, model, production year, and seating capacity. Then I created two child classes, Car and Motorcycle, that inherited from Vehicle and added their own specific features such as HorsePower or EngineCapacity. I also applied abstraction by making Vehicle an abstract class with the abstract method DescribePerformance(), which both subclasses had to define in their own way. This made it easier to describe performance differently for a car and a motorcycle.

Method overriding is shown through the displayInfo() method. The base class has the general version, while each subclass overrides it to include specific details and also calls the DescribePerformance() method. For validation, I used control flow with if statements and do-while loops to make sure input values were valid before creating any objects. I also made helper functions like ReadNonEmpty(), ReadIntInRange(), and CheckBool() to make input checking easier and avoid repeating code.

Constructors were used in each class to assign values to the properties when creating an object. I used : base(...) to send the general vehicle info to the parent constructor, which helped keep things clean. Each

vehicle’s info is displayed correctly because the displayInfo() method shows both the inherited and unique properties. I struggled a bit at first with using base and overriding methods, but this activity helped me understand how it all connects. I also enjoyed creating reusable functions to handle user input and validation.