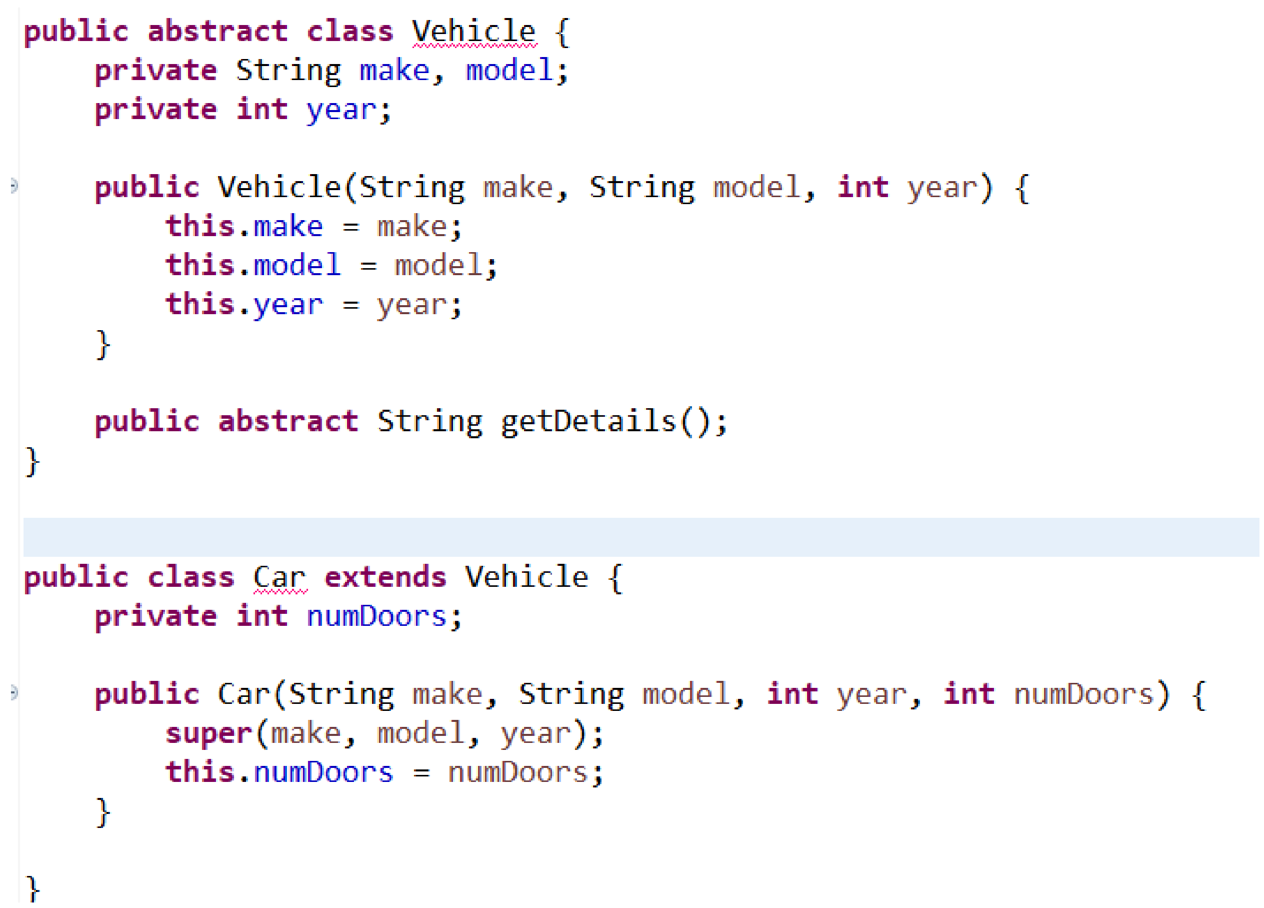
Credit Name: Chapter 8

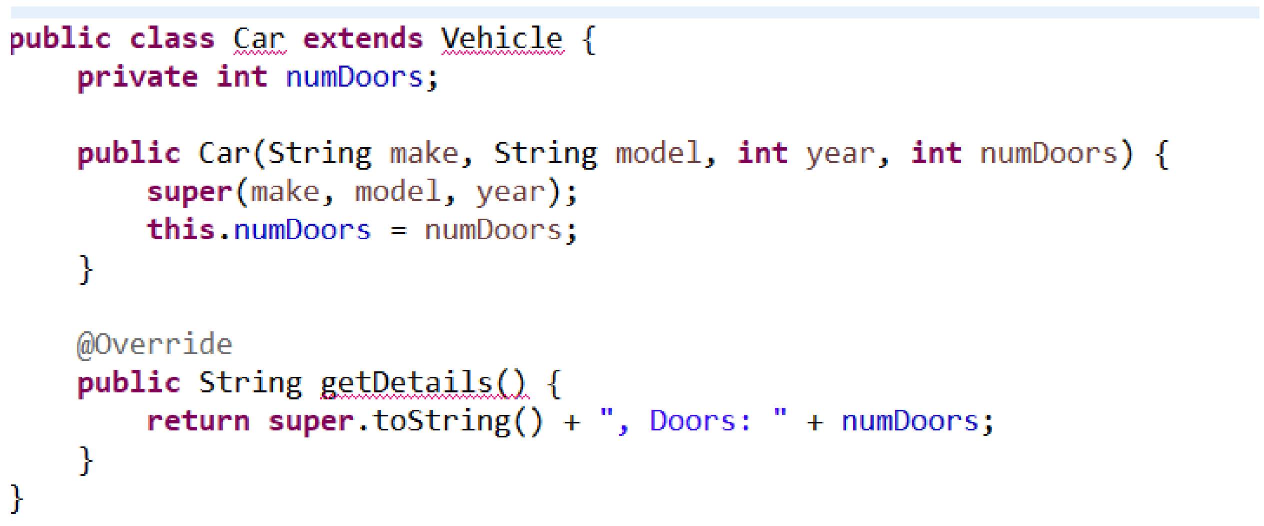
Assignment Name: Vehicle

Describe the errors you've encountered while working on this assignment. What caused the error and how do you overcome the error?

1. Missing Method Implementation



```java
public abstract class Vehicle {
    private String make, model;
    private int year;

    public Vehicle(String make, String model, int year) {
        this.make = make;
        this.model = model;
        this.year = year;
    }

    public abstract String getDetails();
}

public class Car extends Vehicle {
    private int numDoors;

    public Car(String make, String model, int year, int numDoors) {
        super(make, model, year);
        this.numDoors = numDoors;
    }

}
```



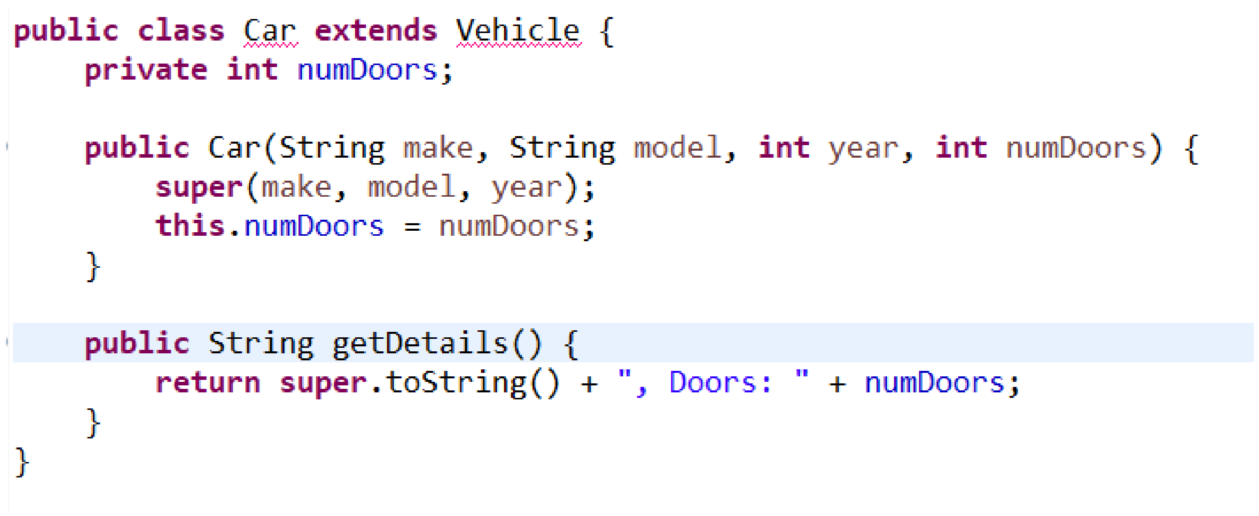
```java
public class Car extends Vehicle {
    private int numDoors;

    public Car(String make, String model, int year, int numDoors) {
        super(make, model, year);
        this.numDoors = numDoors;
    }

    @Override
    public String getDetails() {
        return super.toString() + ", Doors: " + numDoors;
    }
}
```

- Error: The subclasses (Car, Truck, Minivan) did not implement the getDetails() method from the abstract Vehicle class.

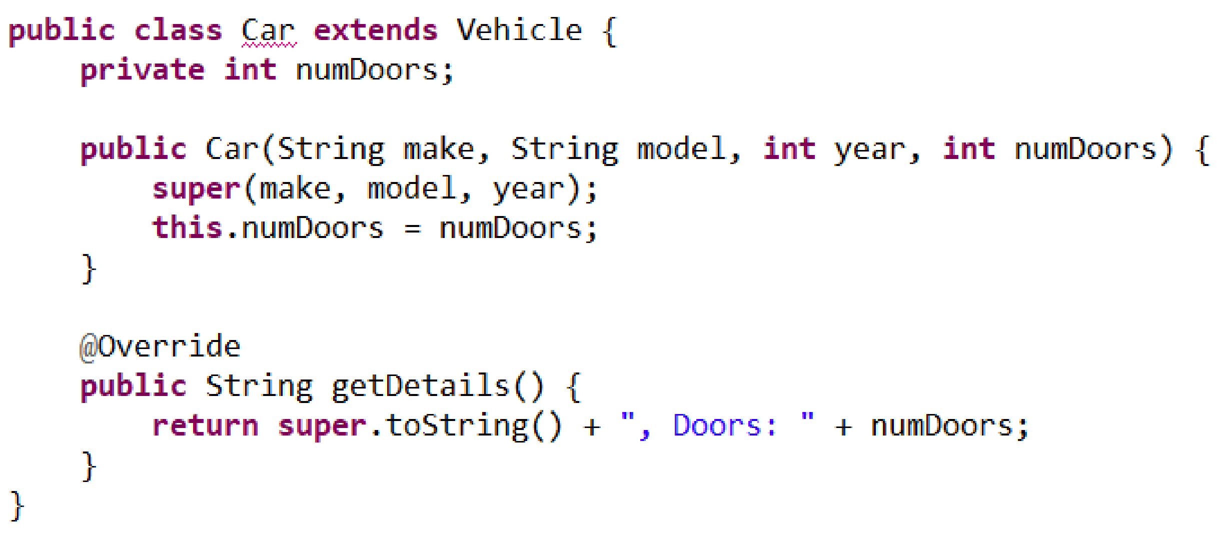
- Cause: Abstract methods in a base class must be implemented by all concrete subclasses.

- Fix: Added the getDetails() method to all subclasses with specific implementations for their respective attributes.

2. Forgetting the @Override Annotation



```java
public class Car extends Vehicle {
    private int numDoors;

    public Car(String make, String model, int year, int numDoors) {
        super(make, model, year);
        this.numDoors = numDoors;
    }

    public String getDetails() {
        return super.toString() + ", Doors: " + numDoors;
    }
}
```



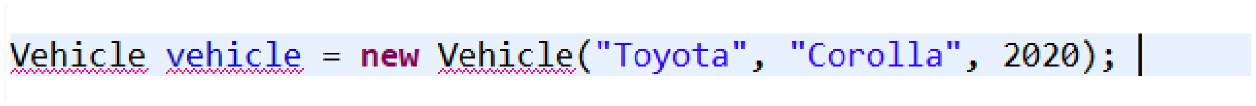
```java
public class Car extends Vehicle {
    private int numDoors;

    public Car(String make, String model, int year, int numDoors) {
        super(make, model, year);
        this.numDoors = numDoors;
    }

    @Override
    public String getDetails() {
        return super.toString() + ", Doors: " + numDoors;
    }
}
```

- Error: The getDetails() method in subclasses was implemented without the @Override annotation, causing potential method signature mismatches.

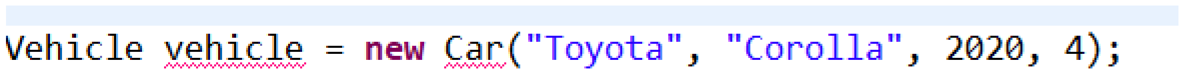
- Cause: Without @Override, the compiler does not verify if the method properly overrides an abstract method in the parent class.

- Fix: Added the @Override annotation to ensure the method matches the signature in the base class.

3. Instantiating an Abstract Class



```java
Vehicle vehicle = new Vehicle("Toyota", "Corolla", 2020);
```



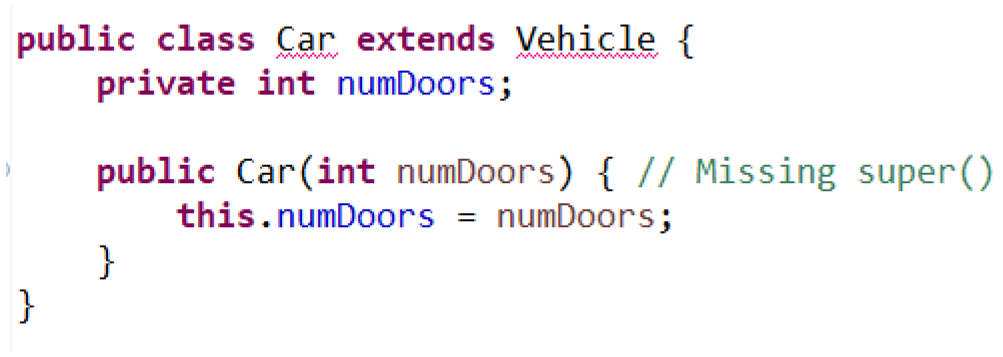
```java
Vehicle vehicle = new Car("Toyota", "Corolla", 2020, 4);
```

- Error: Attempting to create an instance of the abstract Vehicle class directly.

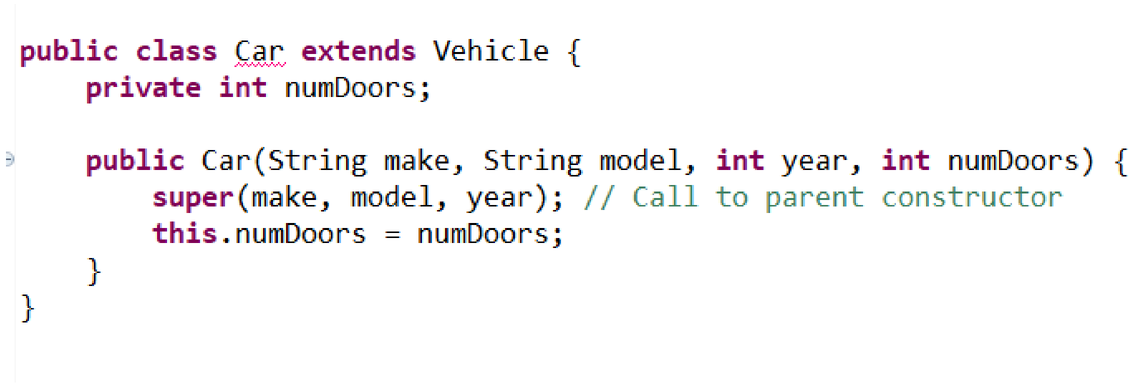
- Cause: Abstract classes cannot be instantiated.

- Fix: Created instances of concrete subclasses like Car , Truck, and Minivan instead.

4. Missing Constructor in Subclasses



```java
public class Car extends Vehicle {
    private int numDoors;

    public Car(int numDoors) { // Missing super()
        this.numDoors = numDoors;
    }
}
```



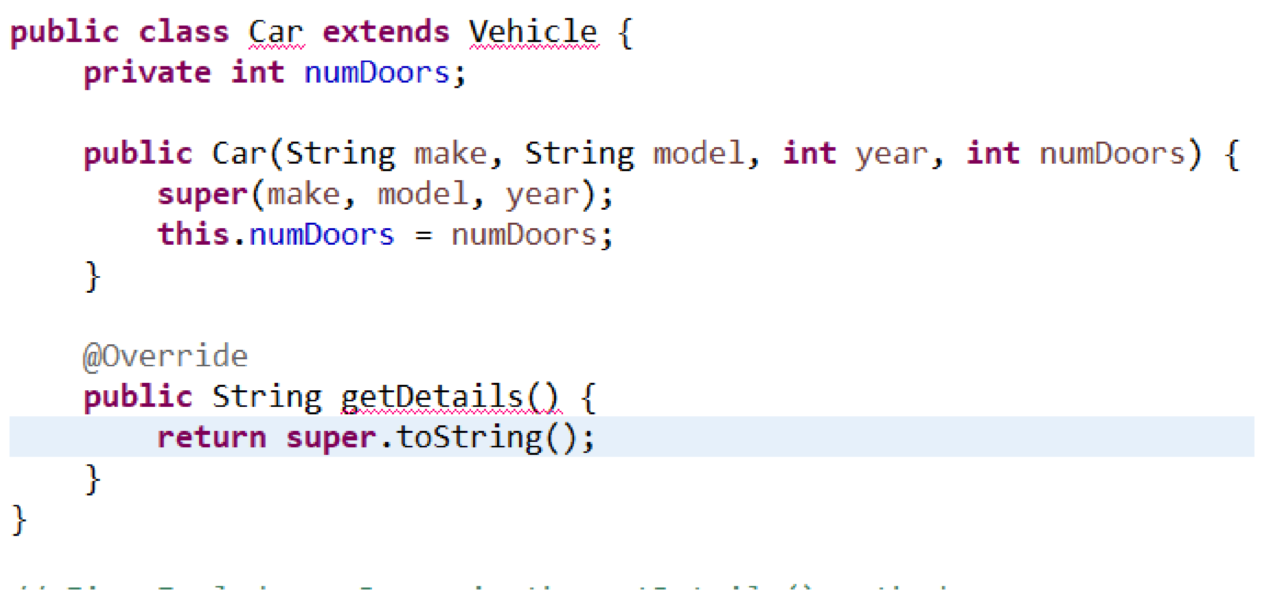
```java
public class Car extends Vehicle {
    private int numDoors;

    public Car(String make, String model, int year, int numDoors) {
        super(make, model, year); // Call to parent constructor
        this.numDoors = numDoors;
    }
}
```

- Error: The subclasses did not define constructors to match the `Vehicle` superclass constructor causing a compilation error.

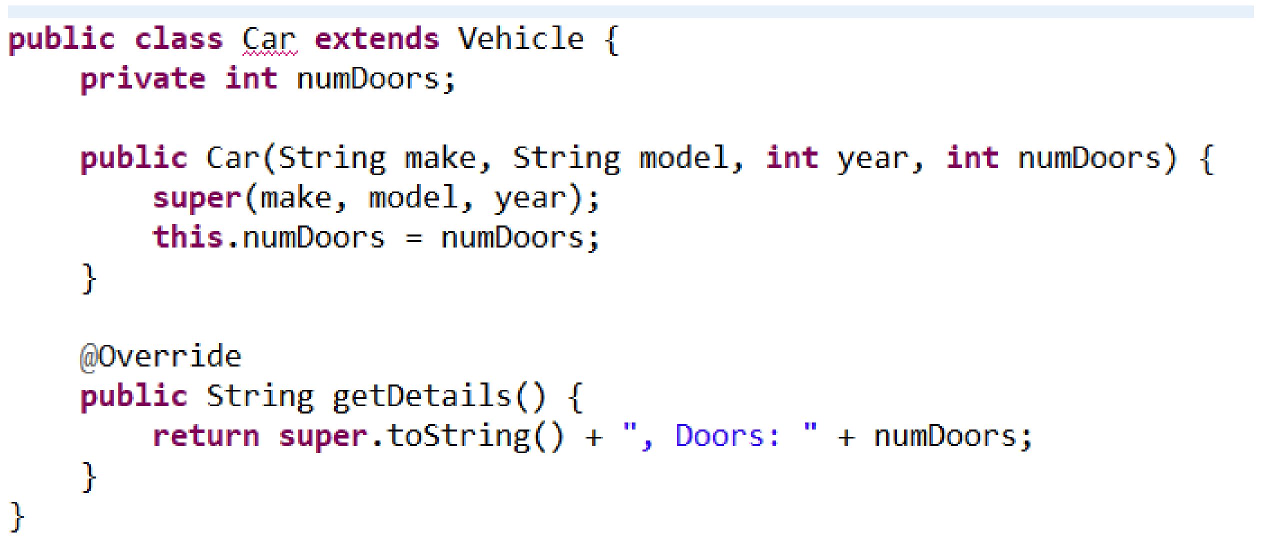
- Cause: Java requires subclasses to explicitly call the superclass constructor if it has parameters.

- Fix: Added constructors to each subclass and used super() to call the parent constructor.

5. Forgetting to Include Subclass-Specific Fields



```java
public class Car extends Vehicle {
    private int numDoors;

    public Car(String make, String model, int year, int numDoors) {
        super(make, model, year);
        this.numDoors = numDoors;
    }

    @Override
    public String getDetails() {
        return super.toString();
    }
}
```



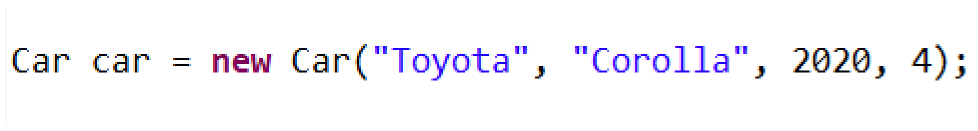
```java
public class Car extends Vehicle {
    private int numDoors;

    public Car(String make, String model, int year, int numDoors) {
        super(make, model, year);
        this.numDoors = numDoors;
    }

    @Override
    public String getDetails() {
        return super.toString() + ", Doors: " + numDoors;
    }
}
```

- Error:The getDetails()method in the subclasses did not include subclass-specific fields like numDoors for Car or payloadCapacity for Truck.

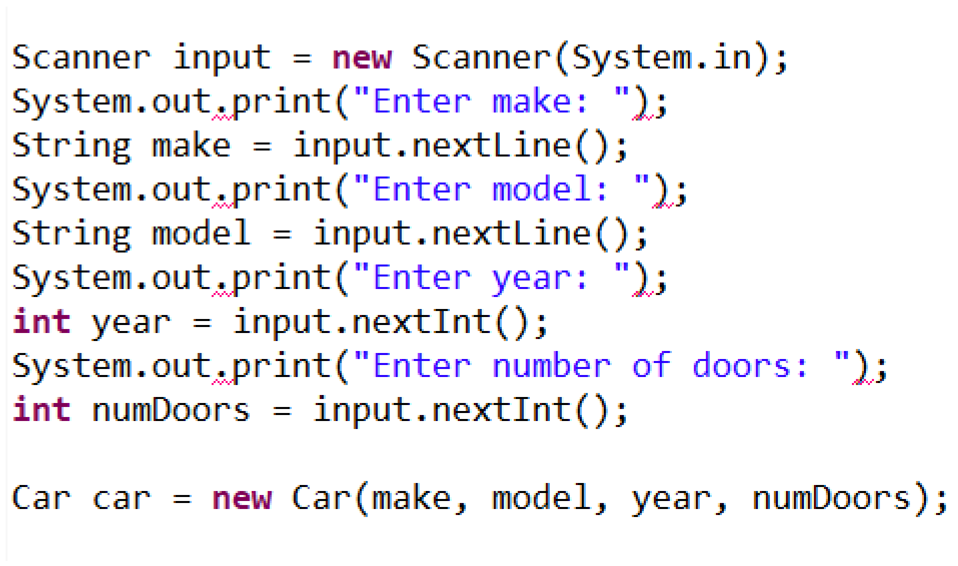
- Cause: Only the shared fields (make, model, year) from the base class were displayed.

- Fix:Updated the getDetails() method in each subclass to include their specific fields.

6. Hardcoding Vehicle Data



```
Car car = new Car("Toyota", "Corolla", 2020, 4);
```



```
Scanner input = new Scanner(System.in);
System.out.print("Enter make: ");
String make = input.nextLine();
System.out.print("Enter model: ");
String model = input.nextLine();
System.out.print("Enter year: ");
int year = input.nextInt();
System.out.print("Enter number of doors: ");
int numDoors = input.nextInt();

Car car = new Car(make, model, year, numDoors);
```

- Error: Vehicle details like make, model, and year were hardcoded, making the program less flexible.

- Cause: The program did not allow dynamic input for creating vehicles.

- Fix: Allow the user to input vehicle details dynamically using Scanner.