3. Prototype Design Pattern

The Prototype Design Pattern is a creational design pattern that deals with the creation of objects by cloning or copying an existing instance, known as the prototype. Instead of creating a new object by invoking a constructor, the prototype pattern involves creating new objects by copying an existing object, known as the prototype. This pattern is particularly useful when the cost of creating a new object is more expensive or complex than copying an existing one.

Here are the key components of the Prototype Design Pattern:

1. Prototype:

- The Prototype interface declares the method `clone`, which concrete prototypes must implement. This method is used for creating a copy of the object.

- In some cases, the Prototype interface may also include other methods for initializing or customizing the cloned object.

2. ConcretePrototype:

- ConcretePrototype classes implement the Prototype interface.

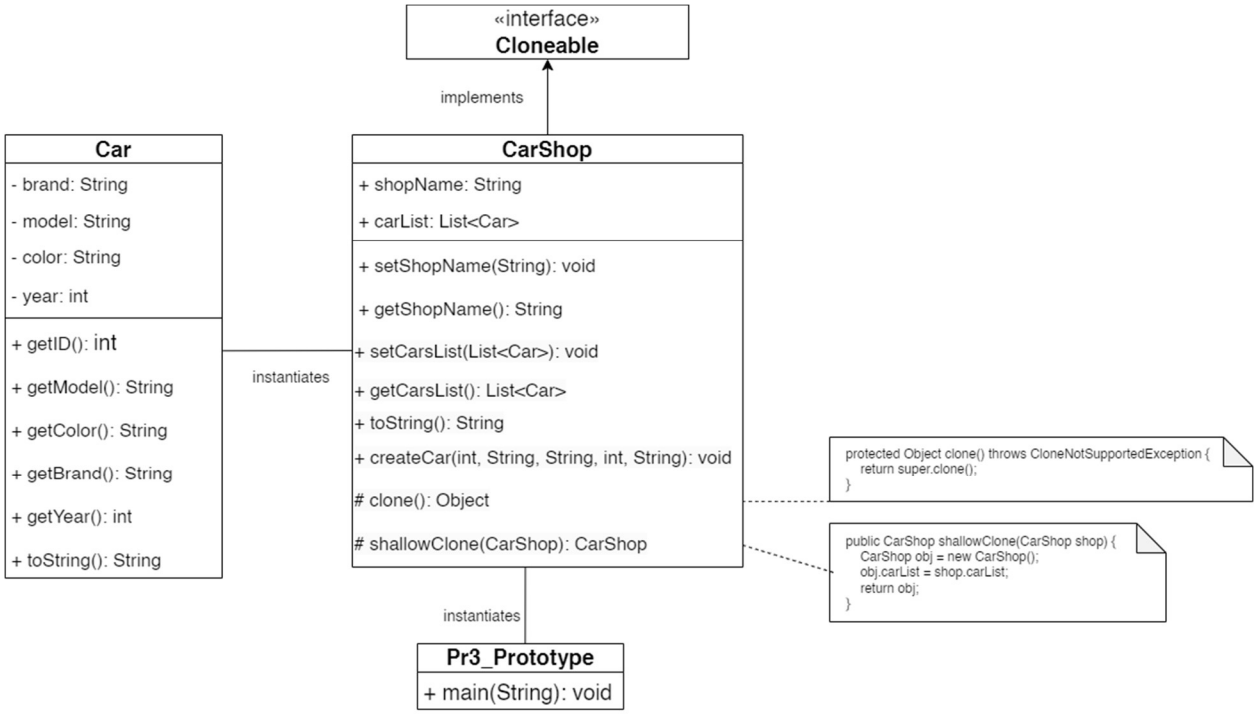
- They provide the implementation of the `clone` method, which performs a shallow or deep copy of the object, depending on the requirements.

3. Client:

- The Client is responsible for creating new objects by cloning the prototype.

- Instead of creating objects using a constructor, the client calls the `clone` method on the prototype.

UML Class Diagram:



Implementation:

```java
import java.util.ArrayList;
import java.util.List;

class Car {
    private String brand;
    private String model;
    private int year;
    private String color;
    private int id;

    // Constructor to initialize the Car with required attributes
    public Car(int id, String brand, String model, int year, String color) {
        this.id = id;
        this.brand = brand;
        this.model = model;
        this.year = year;
        this.color = color;
```

```
    }

    // Getter Method
    public int getID() {
        return id;
    }

    // Getter Method
    public String getBrand() {
        return brand;
    }

    // Getter Method
    public String getModel() {
        return model;
    }

    // Getter Method
    public int getYear() {
        return year;
    }

    // Getter Method
    public String getColor() {
        return color;
    }

    // The toString() method returns the String representation of the object.
    // In this implementation, the Java compiler internally invokes the
toString()
    // method and prints the attributes of the car accordingly.

    public String toString() {
        return "\nCar{" +
                "id='" + id + '\'' +
                "brand='" + brand + '\'' +
                ", model='" + model + '\'' +
                ", year=" + year +
                ", color='" + color + '\'' +
                "}";
    }

}

class CarShop implements Cloneable {
```

```
    private String shopName;

    // List to store the cars in the car shop
    List<Car> carList = new ArrayList<>();

    // Setter method to set the shop name
    public void setShopName(String shopName) {
        this.shopName = shopName;
    }

    // Getter method to fetch the shop name
    public String getShopName() {
        return shopName;
    }

    // Setter method to set the list of cars
    public void setCarsList(List<Car> carList) {
        this.carList = carList;
    }

    // Getter method to fetch the list of cars
    public List<Car> getCarsList() {
        return carList;
    }

    // Overridden method to providestring representation of the CarShop object
    public String toString() {
        return "CarShop{ Shop Name = " + shopName + ", " + "Cars = " + carList +
" }";
    }

    public void createCar(int id, String brand, String model, int year, String
color) {
        Car c = new Car(id, brand, model, year, color);
        getCarsList().add(c);
    }

    // Deep Clone
    protected Object clone() throws CloneNotSupportedException {
        return super.clone();
    }

    // Shallow Clone
    public CarShop shallowClone(CarShop shop) {
        CarShop obj = new CarShop(); // Create a new car shop object
```

```
        obj.carList = shop.carList; // Assign the reference of the old carshop to
the new carshop
        return obj;
    }

}

public class Pr3_Prototype {

    public static void main(String[] args) throws CloneNotSupportedException {
        CarShop car_shop1 = new CarShop();
        car_shop1.setShopName("Car Shop 1");
        car_shop1.createCar(1, "brandA", "modelA", 2023, "Black");
        car_shop1.createCar(2, "brandB", "modelB", 2023, "Black");

        // Deep cloning. Modified Results will not be reflected in book shop 2
        System.out.println("---------------------- Deep Clone (Simply copies
elements) ------------------ ");
        CarShop car_shop2 = (CarShop) car_shop1.clone();
        car_shop2.setShopName("Car Shop 2");
        System.out.println(car_shop1);
        System.out.println(car_shop2);

        // Shallow Clone. Modified Results will all be reflected.
        System.out.println("------------------ Shallow Clone (Changes are
reflected) ------------------ ");
        System.out.println("---------------------- Before Removing --------------
-------- ");
        System.out.println(car_shop1);

        // Shallow clone shop 1 and assign it to shop 3
        CarShop car_shop3 = car_shop1.shallowClone(car_shop1);
        // Remove the first element of shop1 to see if changes are reflected in
shop3
        car_shop1.getCarsList().remove(0);
        car_shop3.setShopName("Car Shop 3");

        System.out.println("---------------------- After Removing ---------------
------- ");
        System.out.println(car_shop1);
        System.out.println(car_shop3);
    }
}
```

Output:

```
---------------------- Deep Clone (Simply copies elements) ---------------
CarShop{ Shop Name = Car Shop 1, Cars = [
Car{id='1'brand='brandA', model='modelA', year=2023, color='Black'},
Car{id='2'brand='brandB', model='modelB', year=2023, color='Black'}] }
CarShop{ Shop Name = Car Shop 2, Cars = [
Car{id='1'brand='brandA', model='modelA', year=2023, color='Black'},
Car{id='2'brand='brandB', model='modelB', year=2023, color='Black'}] }
------------------- Shallow Clone (Changes are reflected) ----------------
---------------------- Before Removing ----------------------
CarShop{ Shop Name = Car Shop 1, Cars = [
Car{id='1'brand='brandA', model='modelA', year=2023, color='Black'},
Car{id='2'brand='brandB', model='modelB', year=2023, color='Black'}] }
---------------------- After Removing ----------------------
CarShop{ Shop Name = Car Shop 1, Cars = [
Car{id='2'brand='brandB', model='modelB', year=2023, color='Black'}] }
CarShop{ Shop Name = Car Shop 3, Cars = [
Car{id='2'brand='brandB', model='modelB', year=2023, color='Black'}] }
```