22BCP469D ï Aryan Randeriya

20CP210P ï Design Patterns Lab

Introduction

A design pattern is an all-purpose solution to frequently occurring issues in software design. It is a reusable solution that can be employed to effectively address a particular set of issues. Design patterns in programming provide a structured method to handle recurrent problems, thereby enhancing the flexibility, maintainability, and reusability of the code. These patterns are beneficial because they have been tried and tested, proving successful in resolving related issues in the past.

Advantages of using design patterns in programming

- **Reusability:** Design patterns are reusable solutions that can be applied to different problems in various contexts.
- **Maintainability:** Design patterns contribute to making code more modular, facilitating easier maintenance and updates.
- **Flexibility:** Design patterns provide greater flexibility in the code, making it easier to incorporate changes without disrupting the entire system.
- **Scalability:** Design patterns aid in ensuring that the code is scalable, capable of handling increased demands as the system grows.
- **Reliability:** Design patterns have undergone testing and proven effectiveness in solving common problems, making them more reliable than ad hoc solutions.

Creational Patterns

In software engineering, creational design patterns are design patterns that deal with object creation mechanisms, trying to create objects in a manner suitable to the situation. The basic form of object creation could result in design problems or in added complexity to the design. Creational design patterns solve this problem by somehow controlling this object creation.

1. Factory Design Pattern

The Factory Design Pattern is a creational design pattern that provides an interface for creating objects in a superclass but allows subclasses to alter the type of objects that will be created. It falls under the category of creational design patterns, which deal with the process of object creation. The main goal of the Factory Design Pattern is to promote code flexibility, separation of concerns, and maintainability by delegating the responsibility of object creation to dedicated factory classes.

Here are the key components and concepts of the Factory Design Pattern:

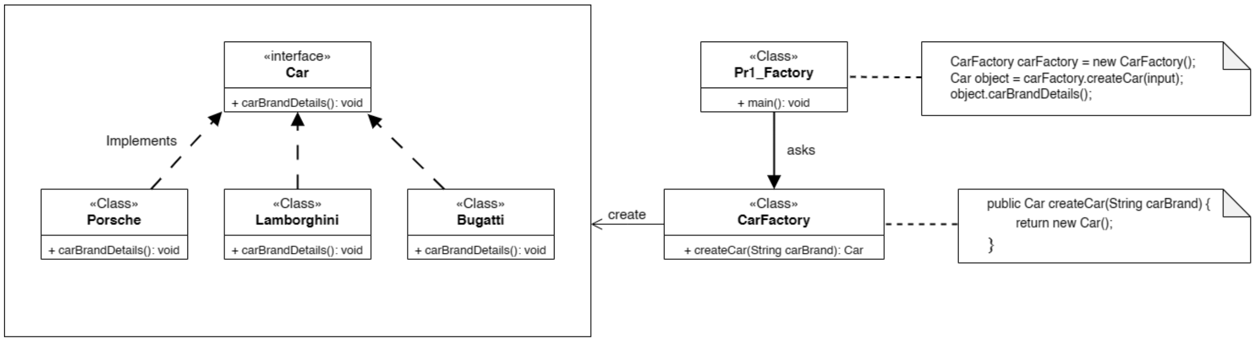
- **Interface or Abstract Class:** The pattern starts with an interface or an abstract class that declares the creation method(s). This interface serves as a blueprint for the concrete classes that will be created.

- **Concrete Classes (Products):** Concrete classes implement the interface or extend the abstract class, defining the actual behavior of the objects being created. These classes represent the different variations or types of objects that the factory can produce.

- **Factory Class:** The factory class is responsible for creating instances of concrete classes based on certain conditions or parameters. It encapsulates the decision-making process of which concrete class to instantiate. The client code interacts with the factory rather than creating objects directly.

- **Client:** The client code is the part of the application that requests the creation of objects from the factory. It relies on the factory to provide instances of the desired class without being concerned about the specific implementation details of the objects.

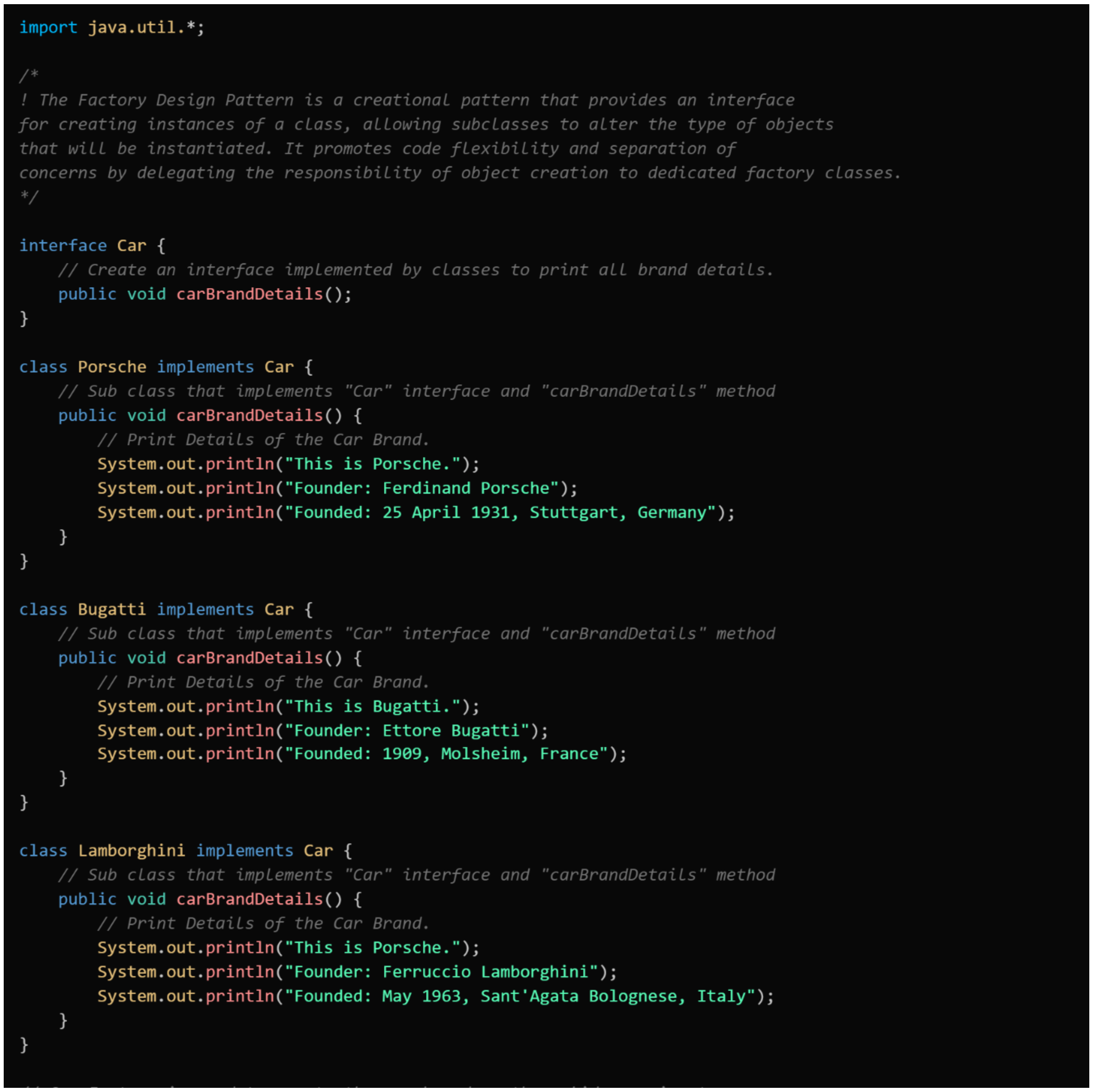
UML Class Diagram:


«interface»
Car
+ carBrandDetails(): void
Implements
«Class»
Porsche
+ carBrandDetails(): void
«Class»
Lamborghini
+ carBrandDetails(): void
«Class»
Bugatti
+ carBrandDetails(): void
create
«Class»
Pr1_Factory
+ main(): void
asks
«Class»
CarFactory
+ createCar(String carBrand): Car

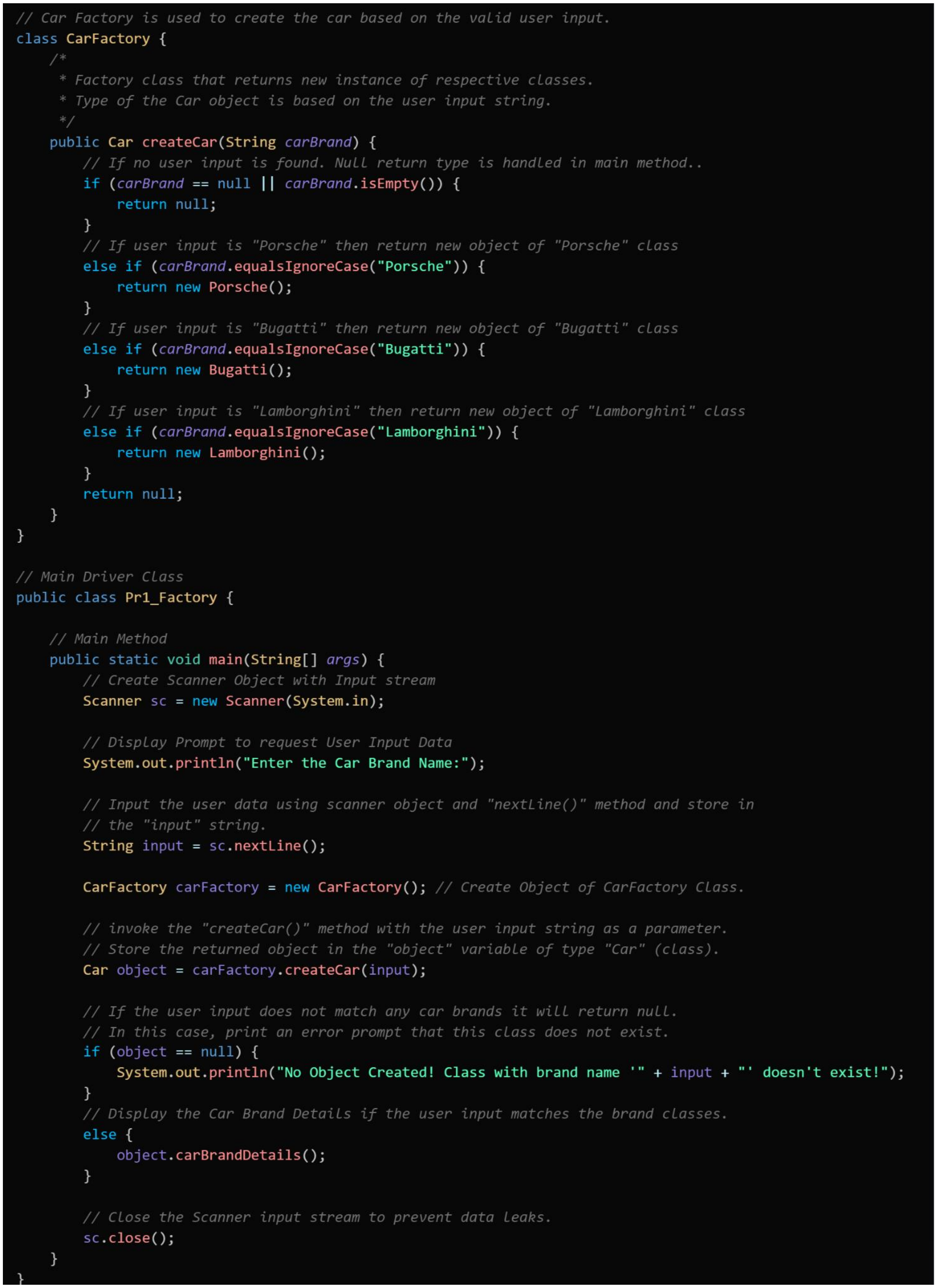
```
CarFactory carFactory = new CarFactory();
Car object = carFactory.createCar(input);
object.carBrandDetails();
```

```
public Car createCar(String carBrand) {
        return new Car();
}
```

Implementation:

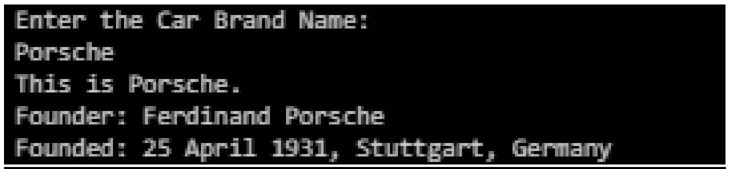


```java
import java.util.*;

/*
! The Factory Design Pattern is a creational pattern that provides an interface
for creating instances of a class, allowing subclasses to alter the type of objects
that will be instantiated. It promotes code flexibility and separation of
concerns by delegating the responsibility of object creation to dedicated factory classes.
*/

interface Car {
    // Create an interface implemented by classes to print all brand details.
    public void carBrandDetails();
}

class Porsche implements Car {
    // Sub class that implements "Car" interface and "carBrandDetails" method
    public void carBrandDetails() {
        // Print Details of the Car Brand.
        System.out.println("This is Porsche.");
        System.out.println("Founder: Ferdinand Porsche");
        System.out.println("Founded: 25 April 1931, Stuttgart, Germany");
    }
}

class Bugatti implements Car {
    // Sub class that implements "Car" interface and "carBrandDetails" method
    public void carBrandDetails() {
        // Print Details of the Car Brand.
        System.out.println("This is Bugatti.");
        System.out.println("Founder: Ettore Bugatti");
        System.out.println("Founded: 1909, Molsheim, France");
    }
}

class Lamborghini implements Car {
    // Sub class that implements "Car" interface and "carBrandDetails" method
    public void carBrandDetails() {
        // Print Details of the Car Brand.
        System.out.println("This is Porsche.");
        System.out.println("Founder: Ferruccio Lamborghini");
        System.out.println("Founded: May 1963, Sant'Agata Bolognese, Italy");
    }
}
```

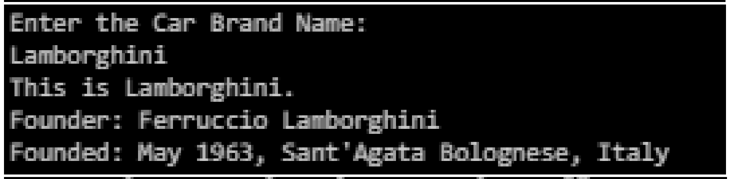


```java
// Car Factory is used to create the car based on the valid user input.
class CarFactory {
    /*
     * Factory class that returns new instance of respective classes.
     * Type of the Car object is based on the user input string.
     */
    public Car createCar(String carBrand) {
        // If no user input is found. Null return type is handled in main method..
        if (carBrand == null || carBrand.isEmpty()) {
            return null;
        }
        // If user input is "Porsche" then return new object of "Porsche" class
        else if (carBrand.equalsIgnoreCase("Porsche")) {
            return new Porsche();
        }
        // If user input is "Bugatti" then return new object of "Bugatti" class
        else if (carBrand.equalsIgnoreCase("Bugatti")) {
            return new Bugatti();
        }
        // If user input is "Lamborghini" then return new object of "Lamborghini" class
        else if (carBrand.equalsIgnoreCase("Lamborghini")) {
            return new Lamborghini();
        }
        return null;
    }
}

// Main Driver Class
public class Pr1_Factory {

    // Main Method
    public static void main(String[] args) {
        // Create Scanner Object with Input stream
        Scanner sc = new Scanner(System.in);

        // Display Prompt to request User Input Data
        System.out.println("Enter the Car Brand Name:");

        // Input the user data using scanner object and "nextLine()" method and store in
        // the "input" string.
        String input = sc.nextLine();

        CarFactory carFactory = new CarFactory(); // Create Object of CarFactory Class.

        // invoke the "createCar()" method with the user input string as a parameter.
        // Store the returned object in the "object" variable of type "Car" (class).
        Car object = carFactory.createCar(input);

        // If the user input does not match any car brands it will return null.
        // In this case, print an error prompt that this class does not exist.
        if (object == null) {
            System.out.println("No Object Created! Class with brand name '" + input + "' doesn't exist!");
        }
        // Display the Car Brand Details if the user input matches the brand classes.
        else {
            object.carBrandDetails();
        }

        // Close the Scanner input stream to prevent data leaks.
        sc.close();
    }
}
```

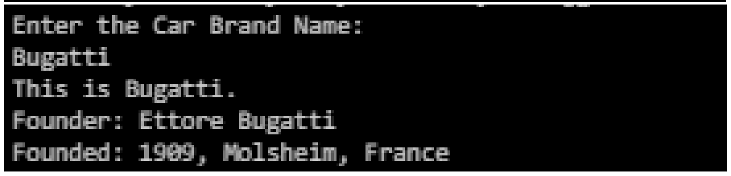
Output:



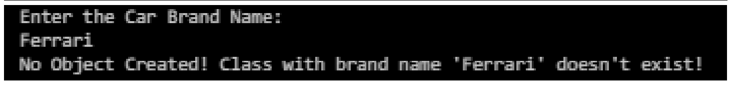
```
Enter the Car Brand Name:
Porsche
This is Porsche.
Founder: Ferdinand Porsche
Founded: 25 April 1931, Stuttgart, Germany
```



```
Enter the Car Brand Name:
Lamborghini
This is Lamborghini.
Founder: Ferruccio Lamborghini
Founded: May 1963, Sant'Agata Bolognese, Italy
```



```
Enter the Car Brand Name:
Bugatti
This is Bugatti.
Founder: Ettore Bugatti
Founded: 1909, Molsheim, France
```



```
Enter the Car Brand Name:
Ferrari
No Object Created! Class with brand name 'Ferrari' doesn't exist!
```