Factory Pattern

Lecture-3

Factory Pattern

The factory method is a creational design pattern, i.e., related to object creation. In the Factory pattern, we create objects without exposing the creation logic to the client and the client uses the same common interface to create a new type of object

Factory Pattern Implementation

- The idea is to use a static member-function (static factory method) that creates & returns instances, hiding the details of class modules from the user.
- A factory pattern is one of the core design principles to create an object, allowing clients to create objects of a library in a way such that it doesn’t have a tight coupling with the class hierarchy of the library.

Code Example – without pattern

```java
// Library Class/Interface
interface Vehicle {
    void printVehicle();
}

class TwoWheeler extends Vehicle {
    public void printVehicle() {
        System.out.println("I am two wheeler");
    }
}

class ThreeWheeler extends Vehicle {
    public void printVehicle() {
        System.out.println("I am three wheeler");
    }
}
```

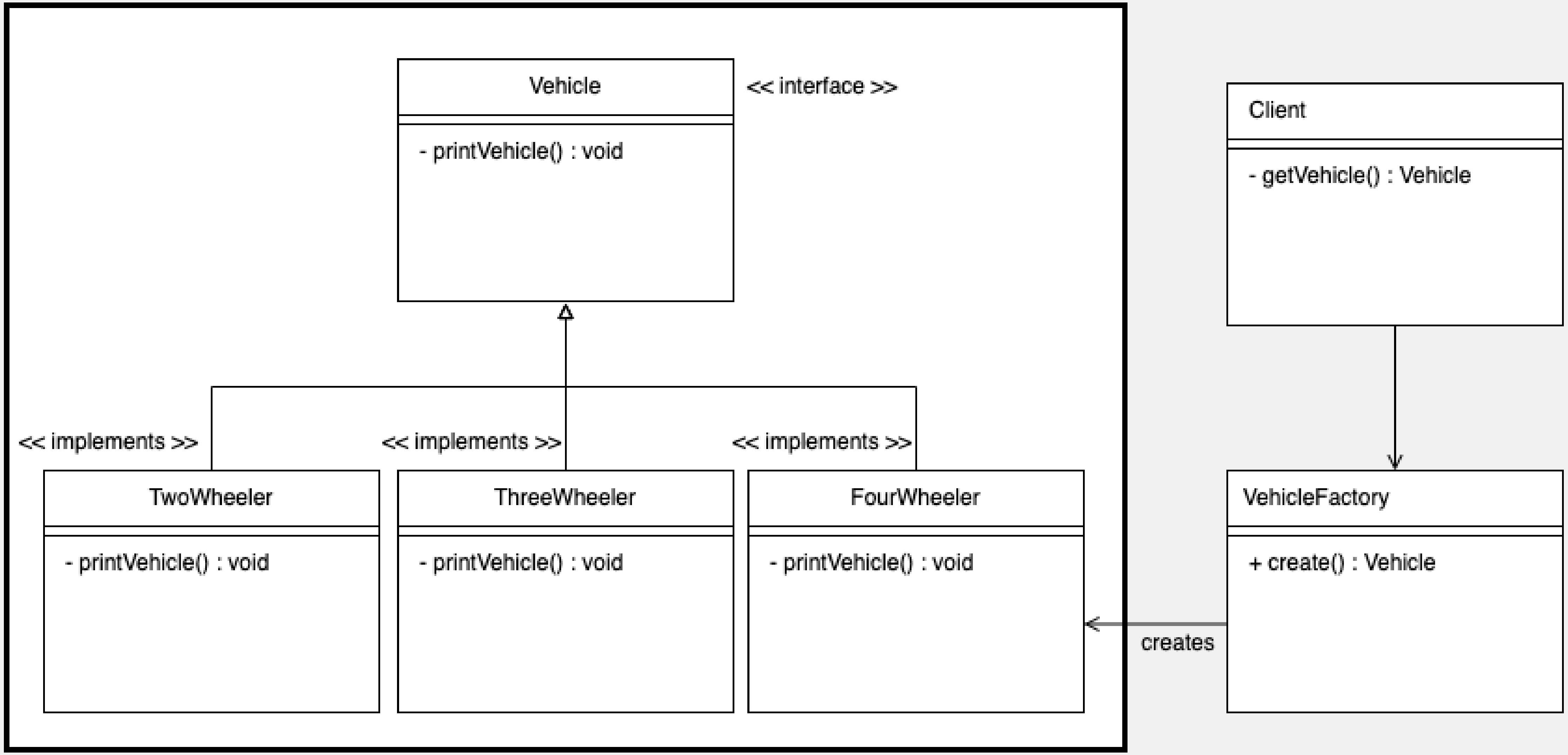
```java
// Client Code
class Client {
     private Vehicle vehicle;

     Client(VehicleType type) {

        if (VehicleType.VT_TwoWheeler.compareTo(type) == 0){
             vehicle = new TwoWheeler();
         } else if (VehicleType.VT_ThreeWheeler.compareTo(type) == 0) {
             vehicle = new ThreeWheeler();
         }
    }

    Vehicle getVehicle() {
        return vehicle;
    }
}
```

Code Example – with pattern

```
// Library Class/Interface
interface Vehicle {
    void printVehicle();
}

class VehicleFactory {
    // Factory method to create objects of different types.
    // Change is required only in this function to create a new object type
     public static Vehicle Create(VehicleType type){
         if (VehicleType.VT_TwoWheeler.compareTo(type) == 0){
             return new TwoWheeler();
         } else if (VehicleType.VT_ThreeWheeler.compareTo(type) == 0) {
             return new ThreeWheeler();
         } else if (VehicleType.VT_FourWheeler.compareTo(type) == 0) {
             return new FourWheeler();
         }
         return null;
     }
}

class TwoWheeler extends Vehicle {
    public void printVehicle() {
        System.out.println("I am two wheeler");
    }
}

class ThreeWheeler extends Vehicle {
    public void printVehicle() {
        System.out.println("I am three wheeler");
    }
}

class FourWheeler extends Vehicle {
    public void printVehicle() {
        System.out.println("I am four wheeler");
    }
}
```

```
// Client class
class Client {

    private final Vehicle pVehicle;

    // Client doesn't explicitly create objects
    // but passes type to factory method "Create()"
    Client(VehicleType type) {
        pVehicle = VehicleFactory.Create(type);
    }

    Vehicle getVehicle() {
        return pVehicle;
    }
}

// Driver Program
public class GFG {
    public static void main(String[] args) {
        Client client = new Client(VehicleType.VT_TwoWheeler);
        Vehicle vehicle = client.getVehicle();
        vehicle.printVehicle();
    }
}
```

Factory Pattern - Diagram



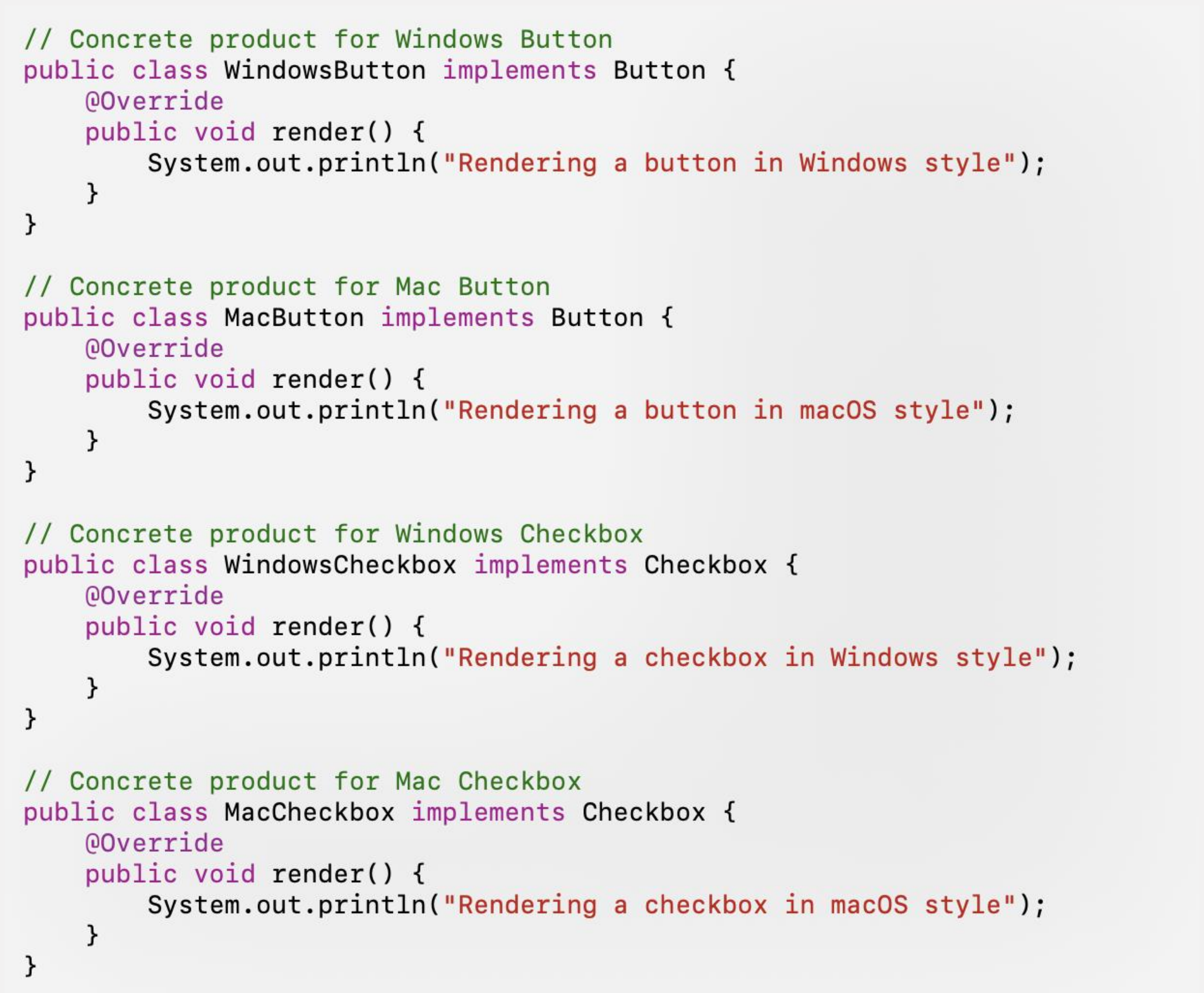
Abstract Factory Code Example

```java
// Abstract product for Button
public interface Button {
    void render();
}

// Abstract product for Checkbox
public interface Checkbox {
    void render();
}
```

```java
// Abstract Factory
public interface GUIFactory {
    Button createButton();
    Checkbox createCheckbox();
}
```

```java
// Concrete Factory for Windows
public class WindowsFactory implements GUIFactory {
    @Override
    public Button createButton() {
        return new WindowsButton();
    }

    @Override
    public Checkbox createCheckbox() {
        return new WindowsCheckbox();
    }
}

// Concrete Factory for macOS
public class MacFactory implements GUIFactory {
    @Override
    public Button createButton() {
        return new MacButton();
    }

    @Override
    public Checkbox createCheckbox() {
        return new MacCheckbox();
    }
}
```

Abstract Factory Code Example



```java
// Concrete product for Windows Button
public class WindowsButton implements Button {
    @Override
    public void render() {
        System.out.println("Rendering a button in Windows style");
    }
}

// Concrete product for Mac Button
public class MacButton implements Button {
    @Override
    public void render() {
        System.out.println("Rendering a button in macOS style");
    }
}

// Concrete product for Windows Checkbox
public class WindowsCheckbox implements Checkbox {
    @Override
    public void render() {
        System.out.println("Rendering a checkbox in Windows style");
    }
}

// Concrete product for Mac Checkbox
public class MacCheckbox implements Checkbox {
    @Override
    public void render() {
        System.out.println("Rendering a checkbox in macOS style");
    }
}
```

Abstract Factory Code Example

```java
public class Application {
    private Button button;
    private Checkbox checkbox;

    public Application(GUIFactory factory) {
        button = factory.createButton();
        checkbox = factory.createCheckbox();
    }

    public void renderUI() {
        button.render();
        checkbox.render();
    }
}

public class Main {
    public static void main(String[] args) {
        // Simulate OS detection
        String os = "Windows";  // This could be detected dynamically in a real-world scenario
        GUIFactory factory;

        if (os.equals("Windows")) {
            factory = new WindowsFactory();
        } else {
            factory = new MacFactory();
        }

        Application app = new Application(factory);
        app.renderUI();
    }
}
```

Abstract Factory Class Diagram

```
                      +--------------------------+
                      |       <<interface>>      |
                      |        GUIFactory        |
                      +--------------------------+
                      | + createButton(): Button |
                      | + createCheckbox(): Checkbox |
                      +--------------------------+
                                  ^
                                  |
             +--------------------+--------------------+
             |                                         |
+--------------------------+            +--------------------------+
| <<concrete class>>       |             | <<concrete class>>       |
|    WindowsFactory        |             |    MacFactory            |
+--------------------------+            +--------------------------+
| + createButton(): Button |             | + createButton(): Button |
| + createCheckbox(): Checkbox |         | + createCheckbox(): Checkbox |
+--------------------------+            +--------------------------+
                                  ^
                                  |
           +------------------------------------+------------------------------------+
           |                                    |                                    |
+---------------------------+      +---------------------------+      +---------------------------+
| <<interface>>              |       | <<interface>>              |       | <<interface>>              |
|         Button             |       |       Checkbox             |       | <<concrete class>>         |
+---------------------------+      +---------------------------+      |        MacCheckbox        |
| + render(): void           |       | + render(): void           |       +--------------------------+
+---------------------------+      +---------------------------+      | + render(): void          |
           ^                                    ^                      +--------------------------+
           |                                    |                                    |
+---------------------------+      +---------------------------+      +---------------------------+
| <<concrete class>>         |       | <<concrete class>>         |       | <<concrete class>>         |
|    WindowsButton           |       |    WindowsCheckbox         |       |        MacButton           |
+---------------------------+      +---------------------------+      +---------------------------+
| + render(): void           |       | + render(): void           |       | + render(): void           |
+---------------------------+      +---------------------------+      +---------------------------+
```