**Flyweight Design Pattern**

The **Flyweight Design Pattern** is a structural pattern that optimizes memory usage by sharing common data among similar objects. This approach is particularly beneficial when dealing with a large number of objects that share common attributes, as it reduces memory consumption and enhances performance.

**Key Components:**

1. **Flyweight**: Defines the interface for objects that can be shared.
2. **ConcreteFlyweight**: Implements the Flyweight interface and stores the intrinsic state (shared data).
3. **FlyweightFactory**: Manages the creation and reuse of Flyweight objects, ensuring that shared instances are used.
4. **Client**: Maintains references to Flyweight objects and provides the extrinsic state (unique data).

**Example Implementation in Java:**

// Flyweight interface  
public interface Shape {  
 void draw(int x, int y, int width, int height, Color color);  
}  
// ConcreteFlyweight  
public class Line implements Shape {  
 @Override  
 public void draw(int x, int y, int width, int height, Color color) {  
 // Drawing logic for Line  
 }  
}  
// ConcreteFlyweight  
public class Oval implements Shape {  
 private boolean fill;  
  
 public Oval(boolean fill) {  
 this.fill = fill;  
 }  
 @Override  
 public void draw(int x, int y, int width, int height, Color color) {  
 // Drawing logic for Oval  
 }  
}  
// FlyweightFactory  
public class ShapeFactory {  
 private static final Map<ShapeType, Shape> *shapes* = new HashMap<>();  
 public static Shape getShape(ShapeType type) {  
 Shape shape = *shapes*.get(type);  
 if (shape == null) {  
 switch (type) {  
 case *OVAL\_FILL*:  
 shape = new Oval(true);  
 break;  
 case *OVAL\_NOFILL*:  
 shape = new Oval(false);  
 break;  
 case *LINE*:  
 shape = new Line();  
 break;  
 }  
 *shapes*.put(type, shape);  
 }  
 return shape;  
 }  
 public enum ShapeType {  
 *OVAL\_FILL*, *OVAL\_NOFILL*, *LINE*;  
 }  
}  
// Client code  
public class DrawingClient {  
 public static void main(String[] args) {  
 Shape line = ShapeFactory.*getShape*(ShapeFactory.ShapeType.*LINE*);  
 line.draw(10, 20, 30, 40, Color.RED);  
 Shape oval = ShapeFactory.*getShape*(ShapeFactory.ShapeType.*OVAL\_FILL*);  
 oval.draw(50, 60, 70, 80, Color.GREEN);  
 }  
}

In this example, Shape is the Flyweight interface, and Line and Oval are ConcreteFlyweights that implement this interface. The ShapeFactory manages the creation and reuse of these objects, ensuring that shared instances are used. The DrawingClient class acts as the client, utilizing the Flyweight objects and providing the necessary extrinsic state during operations.

For a more detailed explanation and additional examples, you can refer to the [Flyweight Design Pattern tutorial on GeeksforGeeks.](https://www.geeksforgeeks.org/flyweight-design-pattern/" \t "_new)