Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJ19ITC802 **DATE: 15/02/2025**

COURSE NAME: Design Patterns Laboratory **CLASS:** BE - IT

EXPERIMENT NO. 3

CO/LO: Identify and apply the most suitable design pattern to address a given application design problem.

AIM: Draw class diagram for Flyweight Pattern and implement the same to demonstrate the working of Counter Strike Game (Instead of creating objects for each player, use Flyweight Pattern to create only 2 objects one for Terrorists class and other for Counter Terrorists class, and reuse them)

DESCRIPTION:

Flyweight pattern is one of the structural design patterns as this pattern provides ways to decrease object count thus improving application required object's structure. Flyweight pattern is used when we need to create many similar objects.

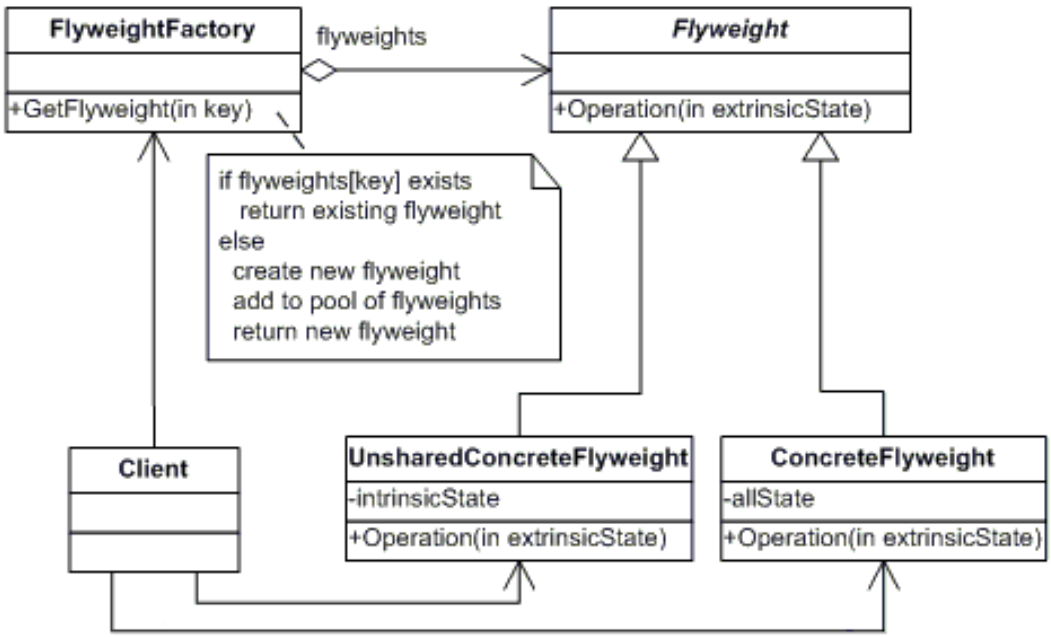
SOURCE CODE:

```
import java.util.HashMap;
import java.util.Map;

// 1. Flyweight Interface
interface Flyweight {
    void operation(ExtrinsicState extrinsicState);
}

// 2. Concrete Flyweight
class ConcreteFlyweight implements Flyweight {
    private String intrinsicState; // Shared state

    public ConcreteFlyweight(String intrinsicState) {
        this.intrinsicState = intrinsicState;
    }
```

```
    @Override
    public void operation(ExtrinsicState extrinsicState) {
        System.out.println("ConcreteFlyweight: Intrinsic State = " + intrinsicState +
                ", Extrinsic State = " + extrinsicState.getState());
    }

    public String getIntrinsicState() {
        return intrinsicState;
    }
}

// 3. Unshared Concrete Flyweight (Not always needed, but shown for completeness)
class UnsharedConcreteFlyweight implements Flyweight {
    private String allState; // Not shared

    public UnsharedConcreteFlyweight(String allState) {
        this.allState = allState;
    }

    @Override
    public void operation(ExtrinsicState extrinsicState) {
        System.out.println("UnsharedConcreteFlyweight: All State = " + allState +
                ", Extrinsic State = " + extrinsicState.getState());
    }
}

// 4. Flyweight Factory
class FlyweightFactory {
    private Map<String, Flyweight> flyweights = new HashMap<>();

    public Flyweight getFlyweight(String key) {
        if (flyweights.containsKey(key)) {
```

```
            return flyweights.get(key);
        } else {
            Flyweight flyweight = new ConcreteFlyweight(key);
            flyweights.put(key, flyweight);
            return flyweight;
        }
    }
}

// 5. Extrinsic State (Passed to the operation)
class ExtrinsicState {
    private String state;

    public ExtrinsicState(String state) {
        this.state = state;
    }

    public String getState() {
        return state;
    }
}

// 6. Client
public class Client {
    public static void main(String[] args) {
        FlyweightFactory factory = new FlyweightFactory();

        // Get flyweights with different intrinsic states
        Flyweight flyweight1 = factory.getFlyweight("State A");
        Flyweight flyweight2 = factory.getFlyweight("State B");
        Flyweight flyweight3 = factory.getFlyweight("State A"); // Same as flyweight1

        // Create extrinsic states
```

```
        ExtrinsicState extrinsic1 = new ExtrinsicState("Value 1");
        ExtrinsicState extrinsic2 = new ExtrinsicState("Value 2");

        // Operate on the flyweights with different extrinsic states
        flyweight1.operation(extrinsic1);
        flyweight2.operation(extrinsic2);
        flyweight3.operation(extrinsic1); // Same intrinsic state, different extrinsic state

        // Example of UnsharedConcreteFlyweight (if needed)
        UnsharedConcreteFlyweight unsharedFlyweight = new
UnsharedConcreteFlyweight("Unique State");
        unsharedFlyweight.operation(extrinsic1);
    }
}
```



Class Diagram for Flyweight Pattern

OUTPUT

```
ConcreteFlyweight: Intrinsic State = State A, Extrinsic State = Value 1
ConcreteFlyweight: Intrinsic State = State B, Extrinsic State = Value 2
ConcreteFlyweight: Intrinsic State = State A, Extrinsic State = Value 1
UnsharedConcreteFlyweight: All State = Unique State, Extrinsic State = Value 1
```

CONCLUSION:

The Flyweight pattern, depicted in the class diagram, is a structural design pattern that aims to minimize memory usage or computational expenses by sharing as much as possible with similar objects. It achieves this by [1] distinguishing between intrinsic (shared) state and extrinsic (unshared) state. The FlyweightFactory is responsible for managing a pool of Flyweight objects. When a client requests a Flyweight, the factory checks if one with the required intrinsic state already exists. If so, it returns the existing instance; otherwise, it creates a new one, adds it to the pool, and returns it. ConcreteFlyweight classes implement the Flyweight interface, storing the intrinsic state. Extrinsic state is passed to the Flyweight's operation method, allowing the same Flyweight instance to be used in different contexts. The UnsharedConcreteFlyweight represents Flyweights that cannot be shared. This pattern is particularly useful when dealing with a large number of objects that share common properties.

REFERENCES:

Foundational Texts (The Classics):

- **Design Patterns: Elements of Reusable Object-Oriented Software** by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides [1] (Addison-Wesley). This is the original "Gang of Four" book that introduced design patterns to the software engineering world. The Flyweight pattern is described in detail here. This is *the* essential reference.

Online Resources (Explanations and Examples):

- **Refactoring.guru:** (refactoring.guru/design-patterns/flyweight) This website provides clear explanations and examples of various design patterns, including Flyweight. It's a great place to get a good understanding of the pattern.
- **Sourcemaking.com:** (sourcemaking.com/design_patterns/flyweight) Another website with explanations and code examples for design patterns.
- **Wikipedia:** (en.wikipedia.org/wiki/Flyweight_pattern) Wikipedia's entry on the Flyweight pattern can provide a good overview.

Java-Specific Resources:

- **Effective Java** by Joshua Bloch (Addison-Wesley). While not solely focused on design patterns, this book offers excellent advice on Java programming practices, including how to

write efficient and maintainable code, which is highly relevant when implementing design patterns.

When Choosing a Reference:

- **Level of Detail:** The Gang of Four book is the most comprehensive but might be more detail than you need initially. Websites like Refactoring.guru are excellent for quick understanding.
- **Language:** If you're working in a specific language (like Java), look for examples and explanations in that language.
- **Context:** Consider the context of your work. If it's for a formal paper or project, citing the original "Gang of Four" book is essential. For quick learning, online resources might be sufficient.