Flyweight pattern is used to reduce number of objects created when there is need to create a lot of Objects of a class. The primary aim is to reduce memory usage (RAM).

It works by storing created objects [1] and reusing the objects.

Uses a **HashMap** to store references to objects created

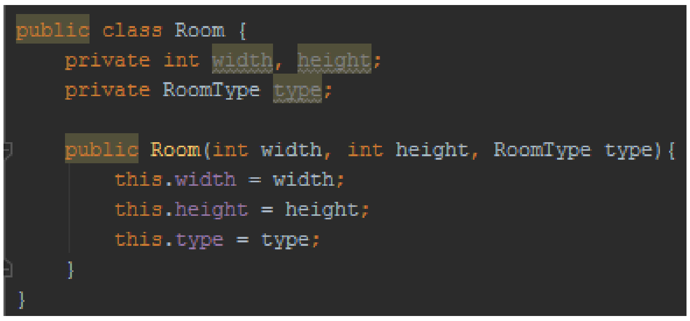
The flyweight object is made up of two properties → *Intrinsic* and *Extrinsic*

Intrinsic properties make the object unique [2].

Extrinsic properties are set by client code and used to perform different operations. [2].

Implementation [1]

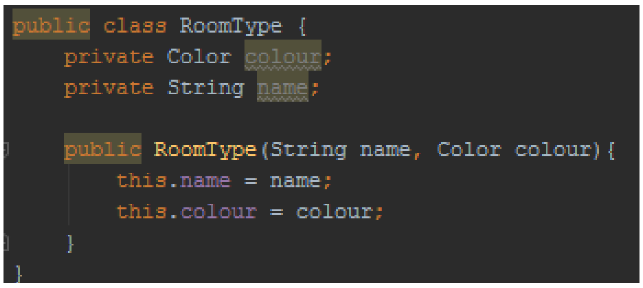
- Interface: define operations client code can perform on the flyweight object
- One or more concrete implementations of the interface
- Factory to handle object instantiation and caching (store to hashmap)



```java
public class Room {
    private int width, height;
    private RoomType type;

    public Room(int width, int height, RoomType type){
        this.width = width;
        this.height = height;
        this.type = type;
    }
}
```

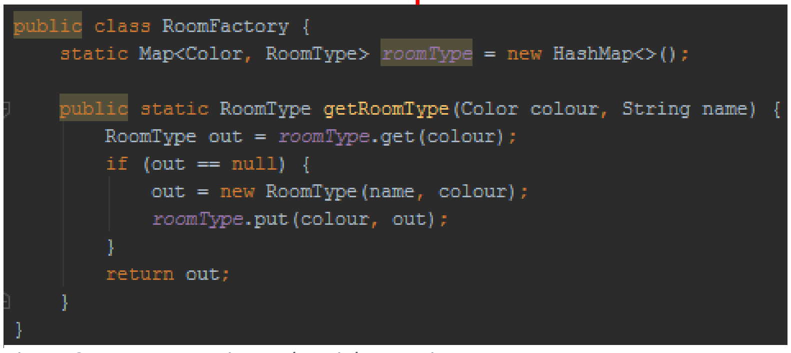
Figure 1 Room.java: State unique for each room



```java
public class RoomType {
    private Color colour;
    private String name;

    public RoomType(String name, Color colour){
        this.name = name;
        this.colour = colour;
    }
}
```

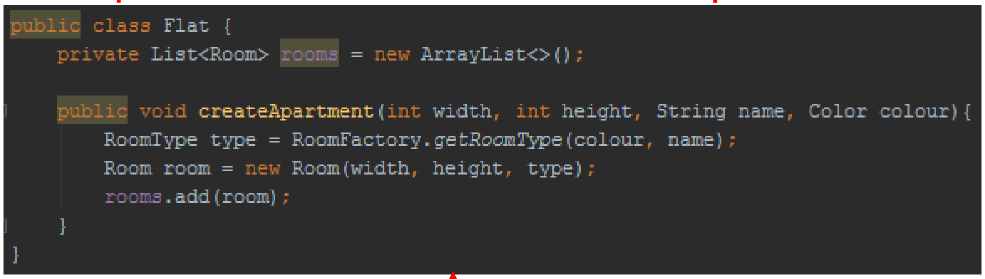
Figure 2 RoomType.java: Shared by several rooms



```java
public class RoomFactory {
    static Map<Color, RoomType> roomType = new HashMap<>();

    public static RoomType getRoomType(Color colour, String name) {
        RoomType out = roomType.get(colour);
        if (out == null) {
            out = new RoomType(name, colour);
            roomType.put(colour, out);
        }
        return out;
    }
}
```

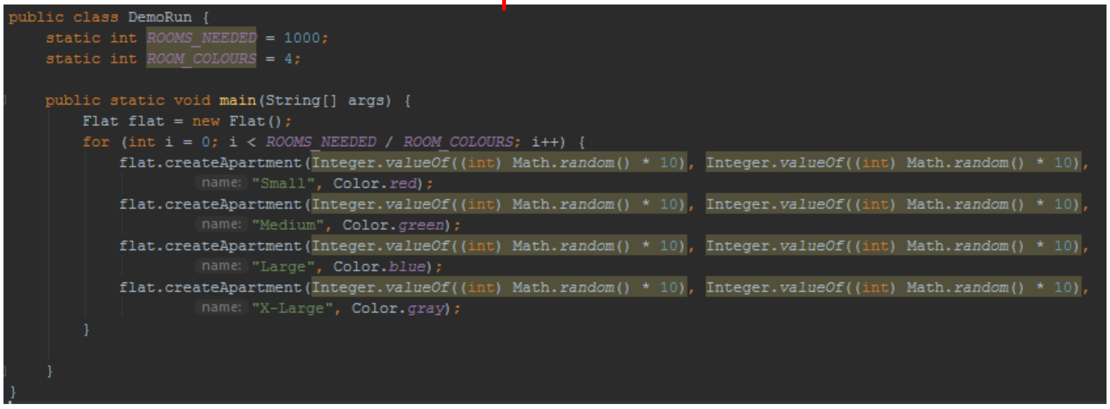
Figure 3 RoomFactory.java: Flyweight creation



```java
public class Flat {
    private List<Room> rooms = new ArrayList<>();

    public void createApartment(int width, int height, String name, Color colour){
        RoomType type = RoomFactory.getRoomType(colour, name);
        Room room = new Room(width, height, type);
        rooms.add(room);
    }
}
```

Figure 3 Flat.java: The apartment to create



```java
public class DemoRun {
    static int ROOMS_NEEDED = 1000;
    static int ROOM_COLOURS = 4;

    public static void main(String[] args) {
        Flat flat = new Flat();
        for (int i = 0; i < ROOMS_NEEDED / ROOM_COLOURS; i++) {
            flat.createApartment(Integer.valueOf((int) Math.random() * 10), Integer.valueOf((int) Math.random() * 10),
                    name: "Small", Color.red);
            flat.createApartment(Integer.valueOf((int) Math.random() * 10), Integer.valueOf((int) Math.random() * 10),
                    name: "Medium", Color.green);
            flat.createApartment(Integer.valueOf((int) Math.random() * 10), Integer.valueOf((int) Math.random() * 10),
                    name: "Large", Color.blue);
            flat.createApartment(Integer.valueOf((int) Math.random() * 10), Integer.valueOf((int) Math.random() * 10),
                    name: "X-Large", Color.gray);
        }

    }
}
```

Figure 5 Client Code