FlyWeight Pattern : stores list of objects , if you want to create a new object which is similar to object stored in flyweight, reuse from list of objects instead of creating new object.

Intrinsic : shareable

Extrinsic : non shareable

Flyweight pattern is primarily used to reduce the number of objects created and to decrease memory footprint and increase performance. This type of design pattern comes under structural pattern as this pattern provides ways to decrease object count thus improving the object structure of application.

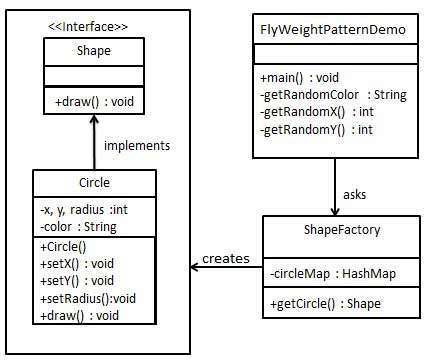
Flyweight pattern tries to reuse already existing similar kind objects by storing them and creates new object when no matching object is found. We will demonstrate this pattern by drawing 20 circles of different locations but we will create only 5 objects. Only 5 colors are available so color property is used to check already existing *Circle* objects.

## Implementation

We are going to create a *Shape* interface and concrete class *Circle* implementing the *Shape* interface. A factory class *ShapeFactory* is defined as a next step.

*ShapeFactory* has a *HashMap* of *Circle* having key as color of the *Circle* object. Whenever a request comes to create a circle of particular color to *ShapeFactory*, it checks the circle object in its *HashMap*, if object of *Circle* found, that object is returned otherwise a new object is created, stored in hashmap for future use, and returned to client.

*FlyWeightPatternDemo*, our demo class, will use *ShapeFactory* to get a *Shape* object. It will pass information (*red / green / blue/ black / white*) to *ShapeFactory* to get the circle of desired color it needs.



Note:

The Flyweight pattern describes how to share objects to allow their use at fine granularity without prohibitive cost. Each "flyweight" object is divided into two pieces: the state-dependent (**extrinsic**) part, and the state-independent (**intrinsic**) part. Intrinsic state is stored (shared) in the Flyweight object. Extrinsic state is stored or computed by client objects, and passed to the Flyweight when its operations are invoked.

import java.util.HashMap;

/\*

\* step 1 ; create an interface

\* Shape.java

\*/

interface Shape{

void draw();

}

/\*

\* step 2 : create concrete class implementing the same interface

\*

\*/

class Circle implements Shape{

//how to decide intrinsic or extrinsic : depends on the requirement

private String color;//intrinsic - can be reused

private int x;//extrinsic - cant be reused , obj may or may not be having same co ordinates

private int y;//extrinsic

private int radius;//in this example , radius considered as, intrinsic - reused, if all circles had different radius - extrinsic

public Circle(String color) {

this.color = color;

}

public void setX(int x) {

this.x = x;

}

public void setY(int y) {

this.y = y;

}

public void setRadius(int radius) {

this.radius = radius;

}

@Override

public void draw(){

System.out.println("Circle: draw() [ Color : " + color + ", x : " + x + ", y "+ y + ", radius :"+radius);

}

}

/\*

\* step 3

\* create a factory to generate object of concrete class based on giver information

\* ShapeFactory.java

\*/

class ShapeFactory{

private static final HashMap circleMap = new HashMap();

public static Shape getCircle(String color) {

Circle circle = (Circle)circleMap.get(color);

if(circle == null) {

circle = new Circle(color);

circleMap.put(color, circle);

System.out.println(" Creating circle of color : "+ color);

}

return circle;

}

}

/\*

\* step 4

\* use the factory to get object of concrete class by passing an information such as color.

\* FlyWeightDemo.java

\*/

public class FlyWeightDemo {

private static final String colors[] = { "Red", "Green", "Blue", "White", "Black" };

public static void main(String[] args) {

for(int i=0; i < 20; ++i) {

Circle circle = (Circle)ShapeFactory.getCircle(getRandomColor());

circle.setX(getRandomX());

circle.setY(getRandomY());

circle.setRadius(100);

circle.draw();

}

}

private static String getRandomColor() {

return colors[(int)(Math.random()\*colors.length)];

}

private static int getRandomX() {

return (int)(Math.random()\*100 );

}

private static int getRandomY() {

return (int)(Math.random()\*100);

}

}