FlyWeight Design Pattern :-

FlyWeight helps us to share same properties between different types of objects. This pattern helps reduce RAM usage and helps to save memory.

```java
public class Tank {
    public int x;
    public int y;
    public int currHealth;

    TankType type;
}
```

composition for TankType.

```java
public class TankType {
    String typeName;
    public int power;
    public int orrHealth;
    public ImageIcon image;

    public TankType(String typeName, int power, int orrHealth, ImageIcon image) {
        this.typeName = typeName;
        this.power = power;
        this.orrHealth = orrHealth;
        this.image = image;
    }
}
```

```java
public class TankTypeFactory {

    static HashMap<String,TankType> repo = new HashMap<>();

    public static TankType getTankType(String type) {
        TankType res = null;
        if(repo.containsKey(type)){
            res = repo.get(type);
        }
        else{
            switch (type) {
                case "typeA":
                    res = new TankType(typeName: "typeA",power: 10,orrHealth: 20,image: null);
                    break;

                case "typeB":
                    res = new TankType(typeName: "typeB",power: 40,orrHealth: 80,image: null);
                    break;

                case "typeC":
                    res = new TankType(typeName: "typeC",power: 700,orrHealth: 120,image: null);
                    break;

                default:
                    break;
            }
            repo.put(type, res);
        }

        return res;
    }
}
```

unique

stores object of each type

→ if object already created, returns it

else creates the objects, stores it in repo & then returns it.

```java
class Test {
    Run | Debug
    public static void main(String[] args) {
        Tank[] tanks = new Tank[100000];

        for(int i=0;i<20000;i++){
            tanks[i] = new Tank();
            tanks[i].type = TankTypeFactory.getTankType(type: "typeA");
        }

        for(int i=20000;i<40000;i++){
            tanks[i] = new Tank();
            tanks[i].type = TankTypeFactory.getTankType(type: "typeB");
        }

        for(int i=40000;i<100000;i++){
            tanks[i] = new Tank();
            tanks[i].type = TankTypeFactory.getTankType(type: "typeC");
        }
    }
}
```

All tanks Contain same object / same address of type A object stored in themselves.

Similarly only 1 TypeB object is shared in all the Tanks created from 20k to 40k.

Same 1 object of TypeC is shared in Tanks objects from 40k to 1 lakh tanks.

Prototype Returns a same property value object to the reference.
It gives a whole new object via deep copy.

```java
public interface Shape {
    Shape cloneShape();
}
```

```java
class Test {
    Run | Debug
    public static void main(String[] args) {
        Shape c1 = new Circle();
        Shape c2 = c1.cloneShape();
        //c2 is a whole different object with same property values as c1.
        //c2 and c1 are 2 different object as we return a deep copy of the object
        //in the cloeShape function.
    }
}
```

```java
public class Rectangle implements Shape {
    int t1x;
    int t1y;
    int width;
    int height;

    public Rectangle() {
        //makes hit to db and sets its properties.
    }

    private Rectangle(int t1x, int t1y, int width, int height) {
        this.t1x = t1x;
        this.t1y = t1y;
        this.width = width;
        this.height = height;
    }

    @Override
    public Shape cloneShape() {
        // Makes deep copy of current object and returns a new cloned object to the reference.
        return new Rectangle(this.t1x,this.t1y,this.width,this.height);
    }
}
```

(Same)

```java
public class Circle implements Shape {
    int x;
    int rad;

    public Circle() {
        //makes hit to db to set x and rad.
    }

    private Circle(int x, int rad) {
        this.x = x;
        this.rad = rad;
    }

    @Override
    public Shape cloneShape() {
        // Makes deep copy of current object and returns a new cloned object to the reference.
        return new Circle(this.x,this.rad);
    }
}
```

returns new circle object with same properties as the object on which (.) dot operator is applied.

```java
public class Rectangle implements Shape {
    int tlx;
    int tly;
    int width;
    int height;
    ArrayList<Integer> list = new ArrayList<>();

    public Rectangle() {
        // makes hit to the db to create
        list.add(10);
        list.add(20);
        list.add(30);

    }

    private Rectangle(int tlx, int tly, int width, int height, ArrayList<Integer> list) {
        this.tlx = tlx;
        this.tly = tly;
        this.width = width;
        this.height = height;
        this.list = new ArrayList<>();
        for(int val: list) {
            this.list.add(val);
        }
    }
```

deepcopy

```java
private Rectangle(int tlx, int tly, int width, int height, ArrayList<Integer> list) {
    this.tlx = tlx;
    this.tly = tly;
    this.width = width;
    this.height = height;
    this.list = list
}
```

Shallow copy