

Flyweight Pattern - WarGame Example - Java Sourcecode

Java Source Code Example for the Flyweight Pattern - WarGame

Consider for example a game of war, were there is a large number of soldier objects; a soldier object maintain the graphical representation of a soldier, soldier behavior such as motion, and firing weapons, in addition soldier's health and location on the war terrain. The solution is to keep the common state of soldiers in a shared object

Java Source Code Example for the Flyweight Pattern - WarGame

Consider for example a game of war, were there is a large number of soldier objects; a soldier object maintain the graphical representation of a soldier, soldier behavior such as motion, and firing weapons, in addition soldier's health and location on the war terrain. The solution is to keep the common state of soldiers in a shared object

The war game example instantiates 5 Soldier clients, each client maintains its internal state which is extrinsic to the soldier flyweight. And Although 5 clients have been instantiated only one flyweight Soldier has been used.

The code below shows Soldier interface which is the flyweight interface

```
package flyweight;

/**
 * Flyweight Interface
 *
 */
public interface Soldier {

    /**
     * Move Soldier From Old Location to New Location
     * Note that soldier location is extrinsic
     * to the SoldierFlyweight Implementation
     * @param previousLocationX
     * @param previousLocationY
     * @param newLocationX
     * @param newLocationY
     */
    public void moveSoldier(int previousLocationX,
        int previousLocationY , int newLocationX ,int newLocationY);
}
```

The code below shows the SoldierImp which is the flyweight implementation

```
package flyweight;

public class SoldierImp implements Soldier {
```

```

/**
 * Intrinsic State maintained by flyweight implementation
 * Soldier Shape ( graphical representation)
 * how to display the soldier is up to the flyweight implementation
 */
private Object soldierGraphicalRepresentation;

/**
 * Note that this method accepts soldier location
 * Soldier Location is Extrinsic and no reference to previous location
 * or new location is maintained inside the flyweight implementation
 */
public void moveSoldier(int previousLocationX, int previousLocationY,
    int newLocationX, int newLocationY) {

    // delete soldier representation from previous location
    // then render soldier representation in new location
}
}

```

The code below shows the Soldier Factory from which we instantiate Soldiers Flyweight

```

package flyweight;

/**
 * Flyweight Factory
 */
public class SoldierFactory {

    /**
     * Pool for one soldier only
     * if there are more soldier types
     * this can be an array or list or better a HashMap
     */
    private static Soldier SOLDIER;

    /**
     * getFlyweight
     * @return
     */
    public static Soldier getSoldier(){

        // this is a singleton
        // if there is no soldier
        if(SOLDIER==null){

            // create the soldier
            SOLDIER = new SoldierImp();
        }

        // return the only soldier reference
        return SOLDIER;
    }
}

```

The code below shows SoldierClient implementation. The client uses the Flyweight soldier reference to perform its tasks

```

package flyweight;

```

```

/**
 * This is the "Heavyweight" soldier object
 * which is the client of the flyweight soldier
 * this object provides all soldier services and is used in the game
 */
public class SoldierClient {

    /**
     * Reference to the flyweight
     */
    private Soldier soldier = SoldierFactory.getSoldier();

    /**
     * this state is maintained by the client
     */
    private int currentLocationX = 0;

    /**
     * this state is maintained by the client
     */
    private int currentLocationY=0;

    public void moveSoldier(int newLocationX, int newLocationY){

        // here the actual rendering is handled by the flyweight object
        soldier.moveSoldier(currentLocationX,
            currentLocationY, newLocationX, newLocationY);

        // this object is responsible for maintaining the state
        // that is extrinsic to the flyweight
        currentLocationX = newLocationX;

        currentLocationY = newLocationY;
    }
}

```

The code below shows the war game implementation. Note that war game instantiates 5 Soldier clients, each client maintains its internal state which is extrinsic to the soldier flyweight. And Although 5 clients have been instantiated only one flyweight Soldier has been used.

```

package flyweight;

/**
 * Driver : War Game
 */
public class WarGame {

    public static void main(String[] args) {
        // start war

        // draw war terrain

        // create 5 soldiers:
        SoldierClient warSoldiers [] ={
            new SoldierClient(),
            new SoldierClient(),
            new SoldierClient(),
            new SoldierClient(),
            new SoldierClient()
        };
    }
}

```

```
// move each soldier to his location
// take user input to move each soldier
warSoldiers[0].moveSoldier(17, 2112);

// take user input to move each soldier
warSoldiers[1].moveSoldier(137, 112);

// note that there is only one SoldierImp ( flyweight Imp)
// for all the 5 soldiers
// Soldier Client size is small due to the small state it maintains
// SoliderImp size might be large or might be small
// however we saved memory costs of creating 5 Soldier representations
}
}
```