Flyweight（享元模式）

我们先来了解一下什么是享元，

享元可以简单理解为共享元素。

在我们生活中，例如共享充电宝，共享电动车，共享单车等等。

共享单车作为共享资源，所以既可以被张三骑，也可以被赵四骑。也就是说这个资源得到复用了。

那么这样做的最大好处是什么？

比如我们去租一辆车的时候就不用把它买下来再用了。

我们就可以用很小的代价得到我们想要的。

再往深处想，有这样一个共享资源后，再有很多人想要骑自行车都不用买了，做到了资源的节省。

再举个例子

假如现在我们使用某盘，下载一部电影，如果有成百上千人保存这个电影，某盘会为每个人单独保存一份吗？显然不会，某盘只需要有一份就够了， 这部影片就是一个共享的资源

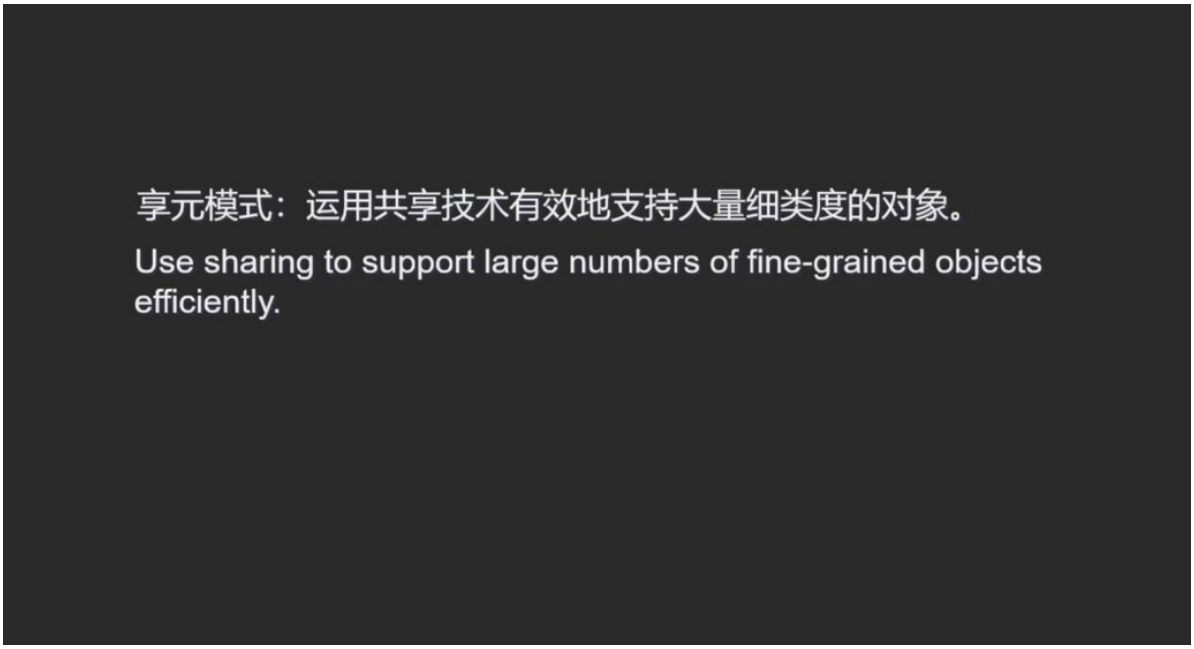
对于下载影片的张三和赵四他们所下载的内容都是一样的，不一样的是共享人和创建时间。

在享元模式中共享元素被分为两个状态,

- 内部状态，比如这部影片的内容，是完全可以被共享的。
- 还有一些不一样的东西，如分享人和创建时间是各不相同的在享元中称为外部状态。

而我们所需要共享的是固定的，不会随着环境的改变而改变。

我们来看这句话



这句话不太好理解，

享元模式最典型的应用就是池技术

如字符串常量池，线程池，数据库连接池等等。

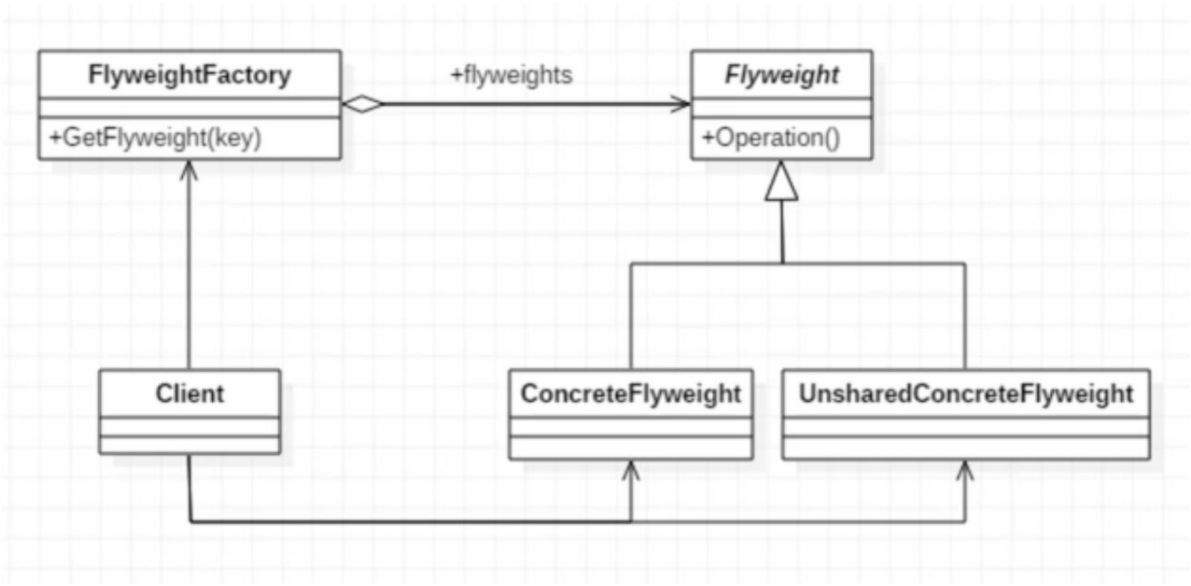
享元共享的资源一般都是不可变的，如String就是使用的享元模式。

```
class{
    String stu = "享元";
    String stu2 = "享元";

    System.out.println(stu==stu2);

    System.out.println(stu.equals(stu2));

}
```

因为声明后的字符串会存到常量池中，再声明相同的字符串时就会从常量池中获取，就不会创建这样一个新的重复的对象。

他们都是实现了对元素的有效利用，避免创建大量重复的对象，有效的利用资源。

享元模式一般由四部分组成



Flyweight,即抽象享元，它用来去定义一些享元对象。

ConcreteFlyweight，就是我们具体的享元对象，在实现它的时候要注意它与外部状态是没有任何关系的。

还有一个非常重要的角色，称为FlyweightFactory,享元工厂

它其实就是一个池容器，相对于我们的线程池、数据库连接池一样的容器，他是存放共享元素的池子。

下面我们来看一个实例代码：

定义一个共享单车的享元类。

```
abstract class BikeFlyWeight{
    //内部状态
    protected Integer state = 0;// 0是未使用,1是使用中

    //userName外部状态
    abstract void ride(String userName);

    abstract void back();

    public Integer getState() {
        return state;
    }
}
```

它可以传递外部状态来打印被谁骑。

还可以将自行车的状态返回。

再定义一个摩拜单车享元类，它继承自我们的单车享元

```
class MoBikeFlyWeight extends BikeFlyWeight{
    //定义新的内部状态，车架号
    private String bikeId;

    public MoBikeFlyWeight(String bikeId){
        this.bikeId = bikeId;
    }

    @Override
    void ride(String userName) {
        state = 1;
        System.out.println(userName +"骑" + bikeId
+"号自行车出行!");
    }

    @Overridevoid back() {
```

```
        state = 0;
    }
}
```

这就是我们定义好的具体的享元类。接下来我们要将享元实例化对象交给我们实例化工厂来进行管理。

这就是享元工厂，这里采用的是单例模式

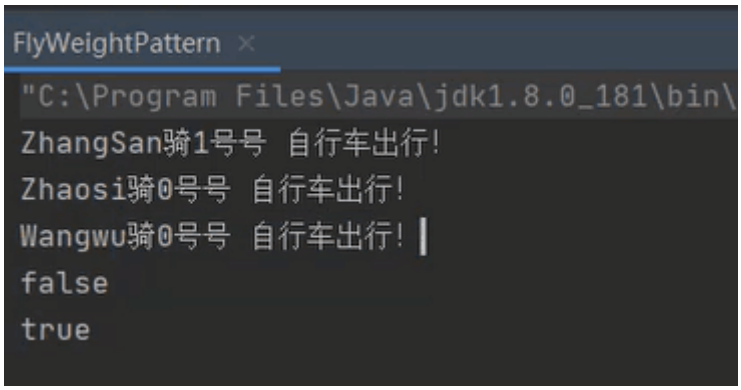
```
class BikeFlyWeightFactory {
    //这里我采用大了饿汉式创建单例模式，在类加载的时候立马去创建。
    private static BikeFlyWeightFactory instance = new BikeFlyWeightFactory();

    private Set<BikeFlyWeight> pool = new HashSet<>();
    //这里用的哈希set因为它是不可重复的。

    public static BikeFlyWeightFactory getInstance(){
        return instance;
    }

    private BikeFlyWeightFactoryO{
        for(int i = 0;i<2;i++){
            pool.add(new MoBikeFlyWeight( bikeId: i+"号"));
        }
    }

    public BikeFlyweight getBike(){
        for(BikeFlyWeight bike : pool){
            if(bike.getState() == 0){
                return bike;
```

```
            }
        }

        return null;
     }
}
```

这个是我们的用户实现类

```
public class FlyWeightPattern {
    public static void main(String[] args){

    BikeFlyWeight bike1 =
BikeFlyWeightFactory.getInstance().getBike();
    bike1.ride(" ZhangSan ");
    //bike1. back();

    BikeFlyWeight bike2 =
BikeFlyWeightFactory.getInstance().getBike();
    bike2.ride(" Zhaosi");
    bike2.back();

    BikeFlyWeight bike3 =
BikeFlyWeightFactory.getInstance().getBike();
    bike3.ride("Wangwu ");
    bike3.back();

        System.out.println(bike1 == bike2);
        System.out.println(bike2 == bike3);
    }
}
```

可以从代码中看到张三使用自行车后没有归还自行车，所以赵四和王五会骑同一辆自行车。



这样就做到了资源的复用，避免了重复对象的创建。

我们来分析一下享元的优缺点

他的优点很明显，使用池技术可以极大减少内存中使得相同或相似对象在内存中只保留一份。可以大大减少系统资源，提高系统性能。

享元模式的外部状态是相对独立的，不会影响内部状态，可以使得享元对象在不同环境中进行共享。

他的缺点也很明显，因为需要对内外部状态进行分离，所以需要较复杂的代码去实现。

虽然降低内存的使用，但是性能方面增加消耗。