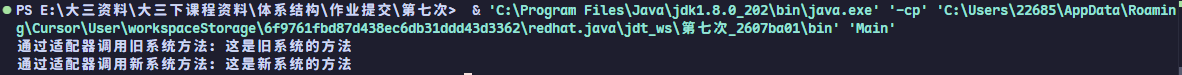
**第七次作业**



**双向适配器​**​是一种设计模式中的适配器，允许两个不兼容的接口​**​互相协作​**​。它不仅可以将一个接口转换为另一个，还能反向转换，使得两种接口可以透明地交互。

**核心实现方式**

1. ​**​双向适配器类​**​同时实现​**​两个目标接口​**​。
2. ​**​内部持有两个接口的实例​**​或直接实现转换逻辑。
3. ​**​在方法调用时，将请求转发或转换​**​到对应的接口方法。

**调用结果**：

**Code:**

*// 旧系统接口*

interface *OldSystem* {

*String* *oldMethod*();

}

*// 新系统接口*

interface *NewSystem* {

*String* *newMethod*();

}

*// 旧系统具体实现*

class *ConcreteOldSystem* implements *OldSystem* {

    @Override

    public *String* *oldMethod*() {

        return "这是旧系统的方法";

    }

}

*// 新系统具体实现*

class *ConcreteNewSystem* implements *NewSystem* {

    @Override

    public *String* *newMethod*() {

        return "这是新系统的方法";

    }

}

*// 双向适配器*

class *BidirectionalAdapter* implements *OldSystem*, *NewSystem* {

    private *OldSystem* oldSystem;

    private *NewSystem* newSystem;

    public *BidirectionalAdapter*(*OldSystem* *oldSystem*, *NewSystem* *newSystem*) {

        this.oldSystem = *oldSystem*;

        this.newSystem = *newSystem*;

    }

    @Override

    public *String* *oldMethod*() {

        if (oldSystem != null) {

            return oldSystem.*oldMethod*();

        }

        return "适配器调用旧系统方法";

    }

    @Override

    public *String* *newMethod*() {

        if (newSystem != null) {

            return newSystem.*newMethod*();

        }

        return "适配器调用新系统方法";

    }

}

*// 主类*

public class *Main* {

    public static void *main*(*String*[] *args*) {

*// 创建具体系统实例*

*OldSystem* oldSystem = new *ConcreteOldSystem*();

*NewSystem* newSystem = new *ConcreteNewSystem*();

*// 创建双向适配器*

*BidirectionalAdapter* adapter = new *BidirectionalAdapter*(oldSystem, newSystem);

*// 通过适配器调用旧系统方法*

*System*.out.*println*("通过适配器调用旧系统方法: " + adapter.*oldMethod*());

*// 通过适配器调用新系统方法*

*System*.out.*println*("通过适配器调用新系统方法: " + adapter.*newMethod*());

    }

}