实验6 职责代理

1 实验目的

1、理解代理模式的设计逻辑，熟悉动态代理的实现方法。

2 实验环境

开发环境：JDK 8.0（或更高版本）
开发工具：Eclipse
设计工具：StarUML（或PlantUML）

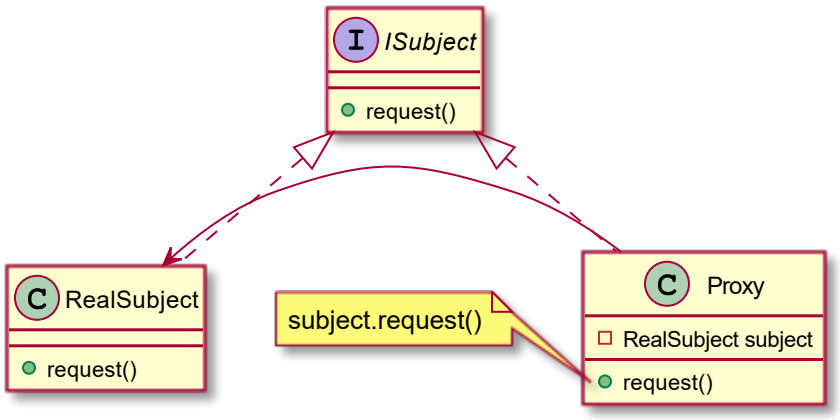
3 基础知识

3.1 代理模式

提供一个代理对象对目标对象的访问进行控制。

> Provide a surrogate or placeholder for another object to control access to it.

结构



参与者：

- 抽象主题（ISubject）：定义代理和真实主题共同接口。

- 真实主题（RealSubject）：定义被代理的真实对象。
- 代理（Proxy）：代理类，其内部持有真实主题引用，往往会在调用被代理对象方法前后增加一些功能。代理的功能一般分为：虚拟代理、保护代理、远程代理、智能指针等。

实现

```
public interface ISubject {
  void request();
}
public class RealSubject implements ISubject {

  @Override
  public void request() {
    System.out.println("处理");
  }

}
public class Proxy implements ISubject {
  RealSubject subject;

  public void setSubject(RealSubject subject) {
    this.subject = subject;
  }

  @Override
  public void request() {
    System.out.println("处理前");
    subject.request();
    System.out.println("处理后");
  }

}
public class Test {

  public static void main(String[] args) {
    Proxy proxy = new Proxy();
    RealSubject sub = new RealSubject();
    proxy.setSubject(sub);
    proxy.request();
  }

}
```

3.2 动态代理

静态代理类只能实现特定目标的访问控制，而某方面的处理可能要面向大多数业务，例如日志处理。如果为每个业务类设计一个静态的代理类显然不是一个可选的方案。动态代理技术能事先定义某方面的处理，在系统运行过程中针对某个特定业务类动态创建代理类，这也是面向切面编程（AOP）实现方式。JDK中的反射包中提供了Proxy类和InvocationHandler接口来实现动态代理。例如我们要对实现了以下两个接口的类进行动态代理：

```
public interface ISubjectA {
  void requestA();
}
public interface ISubjectB {
  void requestB();
}
```

基于InvocationHandler接口定义代理逻辑，基于Proxy动态生成代理类并创建代理对象

```
public class MyInvocationHandler implements InvocationHandler{
  Object subject;

  // 生成代理类，创建代理对象
  public Object bind(Object subject) {
    this.subject = subject;
    return Proxy.newProxyInstance(subject.getClass().getClassLoader(), subject.getClass().getInter
  }
  // 代理逻辑
  @Override
  public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
    System.out.println("执行前");
    method.invoke(subject, args);
    System.out.println("执行后");
    return null;
  }

}
```

动态代理业务测试

```
public class Test {

  public static void main(String[] args) {
    ISubjectA a = new SubjectA();
    ISubjectB b = new SubjectB();

    MyInvocationHandler i = new MyInvocationHandler();

    ISubjectA proxyA = (ISubjectA)(i.bind(a));
    proxyA.requestA();

    ISubjectB proxyB = (ISubjectB)(i.bind(b));
    proxyB.requestB();

  }

}
```

4 实验内容

4.1 带日志功能的计算器

需求描述

基于代理模式对实验1中的计算器进行改造，实现带日志功能的计算器。运行界面和日志文档格式如下：

- 程序运行界面

```
输入：982 + 902
结果1884.00
输入：900 - 88
结果812.00
输入：899 / 90
结果9.99
输入：96 * 55
结果5280.00
```

- 日志记录格式

```
[Tue Oct 18 16:01:50 CST 2022]判断操作符+是否符合当前计算单元：true
[Tue Oct 18 16:01:50 CST 2022]计算982.0+902.0的结果为：1884.0
[Tue Oct 18 16:01:54 CST 2022]判断操作符+是否符合当前计算单元：false
[Tue Oct 18 16:01:54 CST 2022]判断操作符-是否符合当前计算单元：true
[Tue Oct 18 16:01:54 CST 2022]计算900.0-88.0的结果为：812.0
[Tue Oct 18 16:01:58 CST 2022]判断操作符+是否符合当前计算单元：false
[Tue Oct 18 16:01:58 CST 2022]判断操作符-是否符合当前计算单元：false
[Tue Oct 18 16:01:58 CST 2022]判断操作符*是否符合当前计算单元：false
[Tue Oct 18 16:01:58 CST 2022]判断操作符/是否符合当前计算单元：true
[Tue Oct 18 16:01:58 CST 2022]计算899.0/90.0的结果为：9.988888888888889
[Tue Oct 18 16:02:02 CST 2022]判断操作符+是否符合当前计算单元：false
[Tue Oct 18 16:02:02 CST 2022]判断操作符-是否符合当前计算单元：false
[Tue Oct 18 16:02:02 CST 2022]判断操作符*是否符合当前计算单元：true
[Tue Oct 18 16:02:02 CST 2022]计算96.0*55.0的结果为：5280.0
```

实验提示

1、实验主程序与实验1近似，增加了代理对象的操作：

```
public class MainApp {

  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    CalcUnit[] calculator = {new AddUnit(), new SubUnit(), new MulUnit(), new DivUnit()};
    CalcUnitProxy proxy = new CalcUnitProxy();
    PrintWriter pw = null;
    try {
      pw = new PrintWriter(new File("D:/log.txt"));
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    }
    proxy.setLogWriter(pw);

    while(true) {
      System.out.print("输入: ");
      String line = input.nextLine();
      if(line.equals("exit"))
        break;
      String[] items = line.trim().split(" ");
      if(items.length != 3)
        continue;
      String operator = items[1];
      double x = Double.parseDouble(items[0]);
      double y = Double.parseDouble(items[2]);

      // 代理对象替代具体计算单元进行判断和计算
      for(CalcUnit cu : calculator) {
        proxy.setCalcUnit(cu);
        if(proxy.fit(operator)) {
          System.out.printf("结果%4.2f\n", proxy.calc(x, y));
          break;
        }
      }
      pw.flush();
    }
    pw.close();
    input.close();
  }
}
```

2、根据代理模式结构中代理类与目标接口之间的关系设计代理类（CalcUnitProxy），代理类为计算单元增加日志功能时需要用到PrintWriter对象对文件进行写操作，时间信息可以通过日历类获取。

```
// 获取当前时间字符串
GregorianCalendar.getInstance().getTime()

// PrintWriter提供按打印方式的写操作
PrintWriter.printf("%s", s)
PrintWriter.println(s)
```

3、具体的计算单元类对toString()函数进行重写，返回有用信息，例如AddUnit重写返回计算符号"+"

实验要求

- 完成程序，确保功能正常
- 采用StarUML等工具绘制正确的类图，简述设计逻辑

5 实验要求

5.1 实验评价

依据提交的实验报告进行评价，主要评价指标：

1、程序完整，执行正确。（40%）
2、设计逻辑合理，类图正确。（40%）
3、代码、文档格式规范。（20%）

5.2 实验报告

- 实验报告必须用指导老师给定的报告模板，并完成报告模板规定的内容
- 实验报告提交地址为：ftp://172.18.5.102，用户名和密码均为wangxiaomeng
- 文件命名规范：学号+名字，例如2022001晓明
- 实验报告提交方式请采用专门的FTP客户端，不要直接用Windows文件管理器方式上传，因为直接上传的方式不会反馈是否上传成功的信息。