软件构造大作业

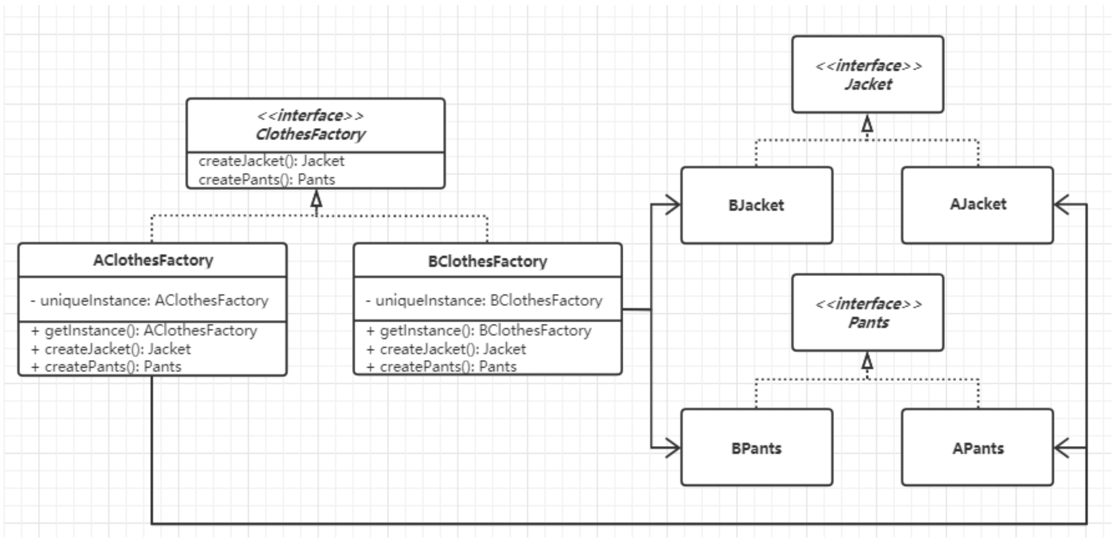
1. 综合利用抽象工厂模式和单件模式，模拟实现服装工厂生产衣服的过程，具体内容如下：现有两种服装品牌的工厂 A 和 B，品牌 A 和品牌 B 的工厂都可以生产夹克和裤子，利用抽象工厂模式实现生产各种品牌的夹克和裤子的代码(即打印“生产 A 品牌夹克”之类的句子即可)，并且利用单件模式，将品牌 A 和品牌 B 的工厂实现为单件。

试完成以下工作：

（1） 画出 UML 类图

（2） 给出核心实现代码

(1)



(2)

ClothesFactory.java

```
/**

 * 抽象工厂

 */

public interface ClothesFactory {

   // 生产夹克

   Jacket createJacket();


   // 生产裤子

   Pants createPants();
}
```

AClothesFactory.java

```
/**

 * 品牌A 生产服装的工厂，可以生产夹克和裤子，使用单件模式
 */

public class AClothesFactory implements ClothesFactory {
    private static AClothesFactory uniqueInstance;

    private AClothesFactory() {

        System.out.println("初始化A 工厂实例...");

    }


    // synchronized 避免多线程问题

    public static synchronized AClothesFactory getInstance() {
        if (uniqueInstance == null) {
            uniqueInstance = new AClothesFactory();

            System.out.println("实例化A 工厂成功");

        } else {

            System.out.println("A 工厂实例已存在");

        }
        return uniqueInstance;
    }

    @Override
    public Jacket createJacket() {
        return new AJacket();
    }

    @Override
    public Pants createPants() {
        return new APants();
    }
}
```

BClothesFactory.java

```
/**

 * 品牌B 生产服装的工厂，可以生产夹克和裤子，使用单件模式

 */

public class BClothesFactory implements ClothesFactory {
    private static BClothesFactory uniqueInstance = null;

```

```java
    private BClothesFactory() {

        System.out.println("初始化B 工厂实例...");

    }

    public static synchronized BClothesFactory getInstance() {
        if (uniqueInstance == null) {
            uniqueInstance = new BClothesFactory();

            System.out.println("实例化B 工厂成功");

        } else {

            System.out.println("B 工厂实例已存在");

        }
        return uniqueInstance;
    }

    @Override
    public Jacket createJacket() {
        return new BJacket();
    }

    @Override
    public Pants createPants() {
        return new BPants();
    }
}
```

Jacket.java

```java
/**

 * 夹克

 */

public interface Jacket {


}
```

Pants.java

```java
/**

 * 裤子

 */

```

```java
public interface Pants {

}
```

AJacket.java

```java
public class AJacket implements Jacket {

    public AJacket() {

        System.out.println("---A 品牌夹克---");

    }
}
```

APants.java

```java
public class APants implements Pants {
    public APants() {

        System.out.println("---A 品牌裤子---");

    }
}
```

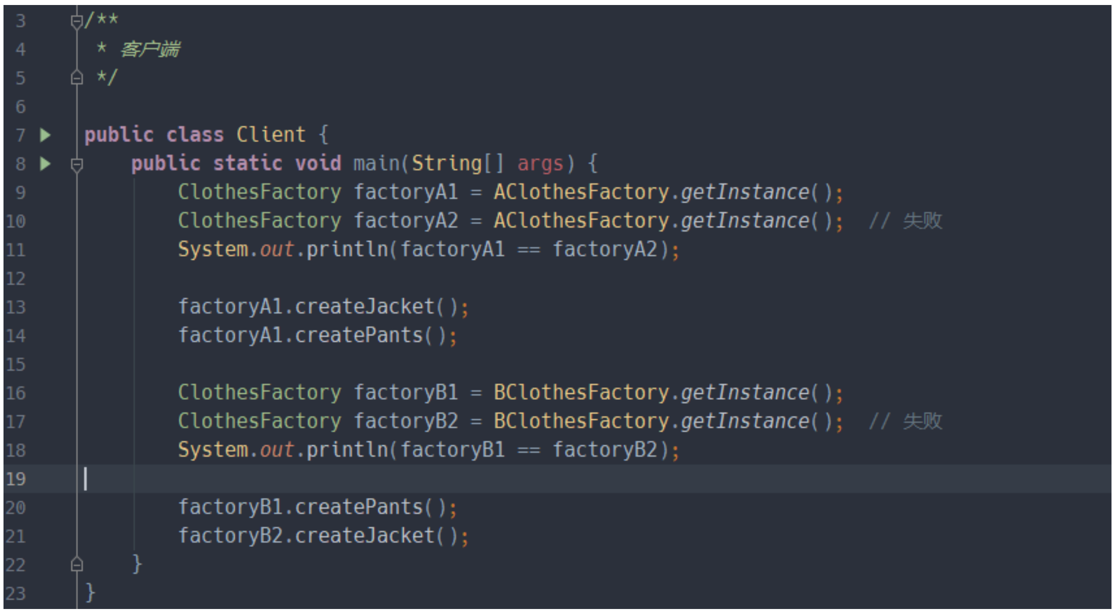
BJacket.java

```java
public class BJacket implements Jacket {
    public BJacket() {

        System.out.println("---B 品牌夹克---");

    }
}
```

BPants.java

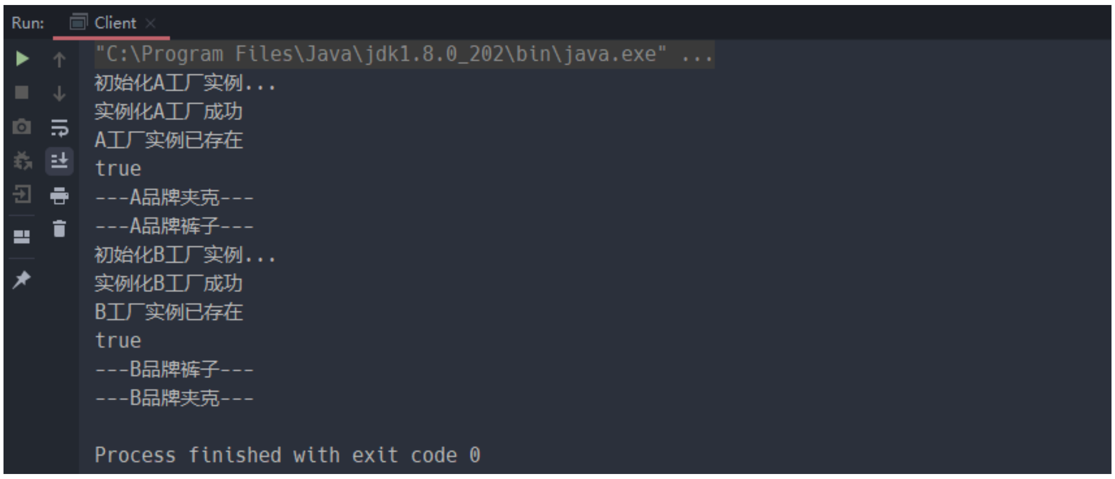
```java
public class BPants implements Pants {
    public BPants() {

        System.out.println("---B 品牌裤子---");

    }
}
```

客户端：



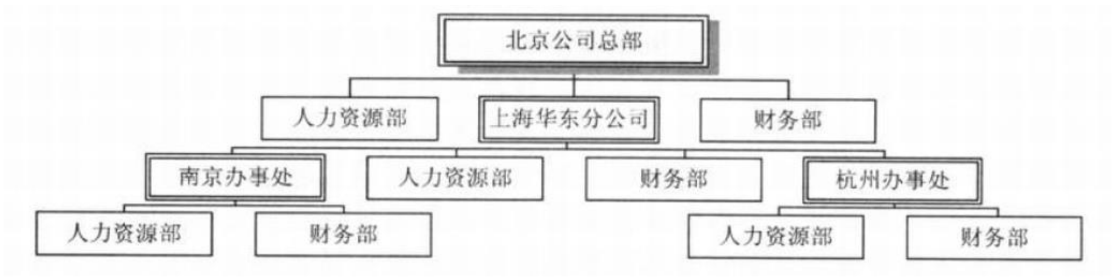
```java
/**
 * 客户端
 */

public class Client {
    public static void main(String[] args) {
        ClothesFactory factoryA1 = AClothesFactory.getInstance();
        ClothesFactory factoryA2 = AClothesFactory.getInstance();  // 失败
        System.out.println(factoryA1 == factoryA2);

        factoryA1.createJacket();
        factoryA1.createPants();

        ClothesFactory factoryB1 = BClothesFactory.getInstance();
        ClothesFactory factoryB2 = BClothesFactory.getInstance();  // 失败
        System.out.println(factoryB1 == factoryB2);

        factoryB1.createPants();
        factoryB2.createJacket();
    }
}
```

结果：



```
"C:\Program Files\Java\jdk1.8.0_202\bin\java.exe" ...
初始化A工厂实例...
实例化A工厂成功
A工厂实例已存在
true
---A品牌夹克---
---A品牌裤子---
初始化B工厂实例...
实例化B工厂成功
B工厂实例已存在
true
---B品牌裤子---
---B品牌夹克---

Process finished with exit code 0
```

2. 根据下图，利用组合模式构建该公司的结构。



即总公司下设上海华东分公司，华东分公司下设南京办事处和杭州办事处。其中，各级分公司均设有人力资源部和财务部。

利用组合模式构建好该结构后，在其树状结构上利用职责链模式处理各地员工加薪请求。

例如，一南京员工提出加薪请求，如果加薪不超过 1000 元，南京办事处即可批准；如果超过 1000 元，则上报华东分公司，如果 2000 元以内，则批准；超过则上报北京总公司，如果

不超过 3000 元，则总公司批准，否则拒绝。

CompanyComponent.java

```java
public abstract class CompanyComponent {
    protected String name;
    protected int minSalary;
    protected int maxSalary;
    protected CompanyComponent successor;

    public void add(CompanyComponent companyComponent) {
        throw new UnsupportedOperationException();
    }

    public void remove(CompanyComponent companyComponent) {
        throw new UnsupportedOperationException();
    }

    public CompanyComponent getChild(int i) {
        throw new UnsupportedOperationException();
    }

    public String getName() {
        throw new UnsupportedOperationException();
    }

    public void display(int depth) {
        throw new UnsupportedOperationException();
    }

    public void setSalaryRange(int minSalary, int maxSalary) {
        this.minSalary = minSalary;
        this.maxSalary = maxSalary;
    }

    public void setSuccessor(CompanyComponent successor) {
        this.successor = successor;
    }

    public void handleRequest(int request) {
        throw new UnsupportedOperationException();
    }
}
```

Company.java

```java
import java.util.ArrayList;

/**
 * 总公司、子公司、办事处
 * Composite
 */

public class Company extends CompanyComponent {

    // 子公司或办事处可以有孩子结点
    ArrayList<CompanyComponent> companyComponents = new
ArrayList<>();
    String name;

    public Company(String name) {
        this.name = name;
    }

    @Override
    public void add(CompanyComponent companyComponent) {
        companyComponents.add(companyComponent);
    }

    @Override
    public void remove(CompanyComponent companyComponent) {
        companyComponents.remove(companyComponent);
    }

    @Override
    public CompanyComponent getChild(int i) {
        return companyComponents.get(i);
    }

    @Override
    public String getName() {
        return name;
    }

    @Override
    public void display(int depth) {
        for (int i = 0; i < depth; i++) {
            System.out.print("-");
        }
        System.out.println(" " + name);

        // 打印孩子结点

        for (CompanyComponent companyComponent : companyComponents) {
            companyComponent.display(depth + 1);
        }
    }

    @Override
    public void setSalaryRange(int minSalary, int maxSalary) {
        super.setSalaryRange(minSalary, maxSalary);
    }

    @Override
    public void setSuccessor(CompanyComponent successor) {
        super.setSuccessor(successor);
    }

    @Override
    public void handleRequest(int request) {
        if (request >= minSalary && request <= maxSalary) {

            System.out.println(name + "批准请求：加薪" + request);

        } else if (successor != null) {

            System.out.println(name + "无权处理请求，上报" +

successor.getName());
            successor.handleRequest(request);
        } else {

            System.out.println(name + "拒绝请求，未批准加薪" + request);

        }
    }
}
```

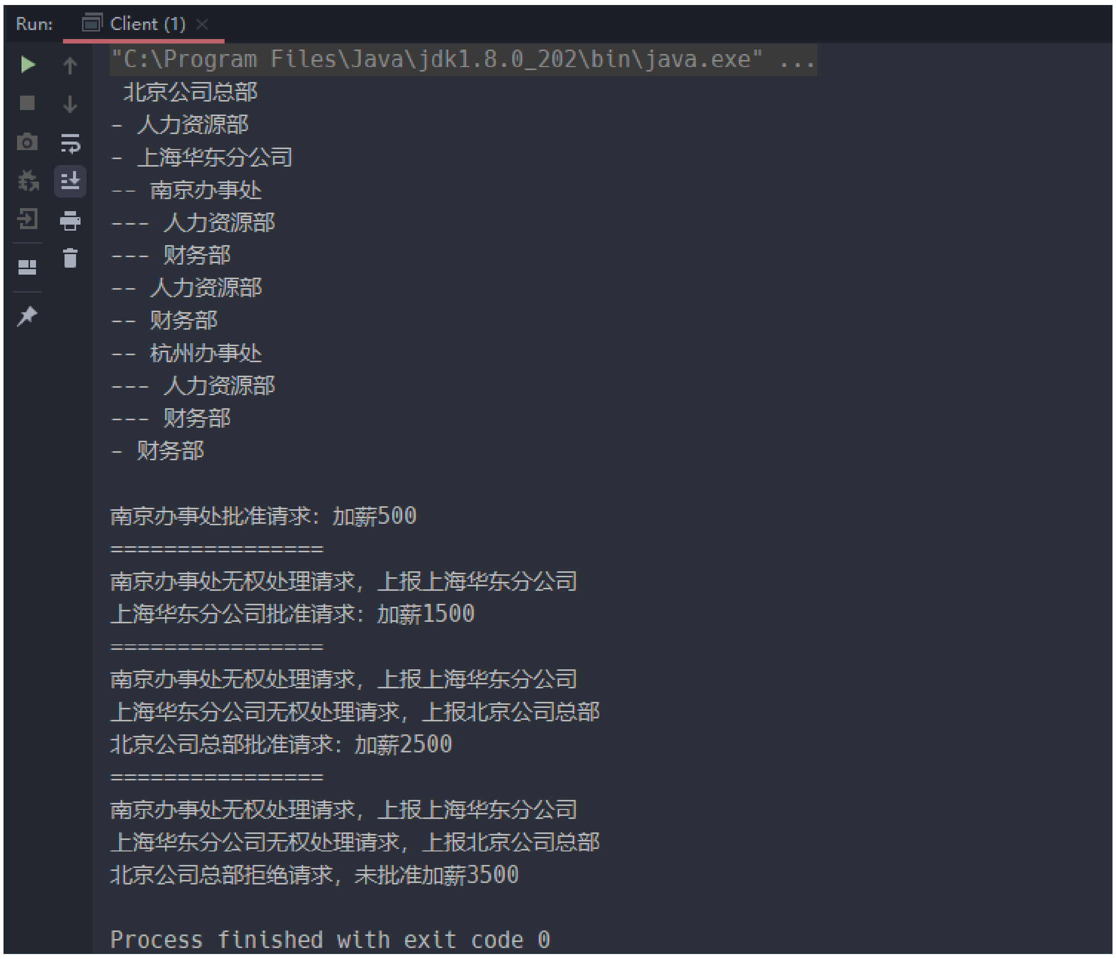
Department.java

```java
/**

 * 部门

 * Leaf
 */

public class Department extends CompanyComponent {
    private String name;
    public Department(String name) {
        this.name = name;
    }

    @Override
    public String getName() {
        return name;
    }

    @Override
    public void display(int depth) {
        for (int i = 0; i < depth; i++) {
            System.out.print("-");
        }
        System.out.println(" " + name);
    }
}
```

客户端：

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

        Company headOffice = new Company("北京公司总部");

        Company shanghaiBranch = new Company("上海华东分公司");


        headOffice.add(new Department("人力资源部"));

        headOffice.add(shanghaiBranch);

        headOffice.add(new Company("财务部"));


        Company nanjingOffice = new Company("南京办事处");

        Company hangzhouOffice = new Company("杭州办事处");


        shanghaiBranch.add(nanjingOffice);

        shanghaiBranch.add(new Department("人力资源部"));

        shanghaiBranch.add(new Company("财务部"));

        shanghaiBranch.add(hangzhouOffice);
        shanghaiBranch.setSuccessor(headOffice);


        nanjingOffice.add(new Department("人力资源部"));

        nanjingOffice.add(new Company("财务部"));

        nanjingOffice.setSuccessor(shanghaiBranch);


        hangzhouOffice.add(new Department("人力资源部"));

        hangzhouOffice.add(new Company("财务部"));

        hangzhouOffice.setSuccessor(shanghaiBranch);

        nanjingOffice.setSalaryRange(0, 1000);
        hangzhouOffice.setSalaryRange(0, 1000);

        shanghaiBranch.setSalaryRange(1001, 2000);
        headOffice.setSalaryRange(2001, 3000);


        // 打印公司组织结构

        headOffice.display(0);

        System.out.println();

        // 南京员工提出加薪 500 请求，南京办事处可批准

        nanjingOffice.handleRequest(500);
        System.out.println("================");

        // 南京员工提出加薪 1500 请求，需上报华东分公司

        nanjingOffice.handleRequest(1500);
        System.out.println("================");

        // 南京员工提出加薪 2500 请求，需上报北京总公司

        nanjingOffice.handleRequest(2500);
        System.out.println("================");

        // 南京员工提出加薪 3500 请求，总公司拒绝

        nanjingOffice.handleRequest(3500);
    }
}
```

结果：



```
"C:\Program Files\Java\jdk1.8.0_202\bin\java.exe" ...
 北京公司总部
- 人力资源部
- 上海华东分公司
-- 南京办事处
--- 人力资源部
--- 财务部
-- 人力资源部
-- 财务部
-- 杭州办事处
--- 人力资源部
--- 财务部
- 财务部

南京办事处批准请求：加薪500
================
南京办事处无权处理请求，上报上海华东分公司
上海华东分公司批准请求：加薪1500
================
南京办事处无权处理请求，上报上海华东分公司
上海华东分公司无权处理请求，上报北京公司总部
北京公司总部批准请求：加薪2500
================
南京办事处无权处理请求，上报上海华东分公司
上海华东分公司无权处理请求，上报北京公司总部
北京公司总部拒绝请求，未批准加薪3500

Process finished with exit code 0
```

3. 利用**观察者模式**和**中介者模式**，实现一个二元函数的函数计算器。该计算器含有两个数据类：

```
class A{                    class B{
    int x;                      int y;
}                           }
```

分别存储整型变量 x 和 y。需要实现三个函数计算器类，分别计算 f1(x,y)=x+y, f2(x,y)=x-y 和 f3(x)=2x。

要求:(1)当且仅当类 A 和类 B 都发布通知时，3 个函数计算器才相应地计算出 3 个新的值。

(2)类 A、类 B 与三个函数计算器类尽可能解耦。

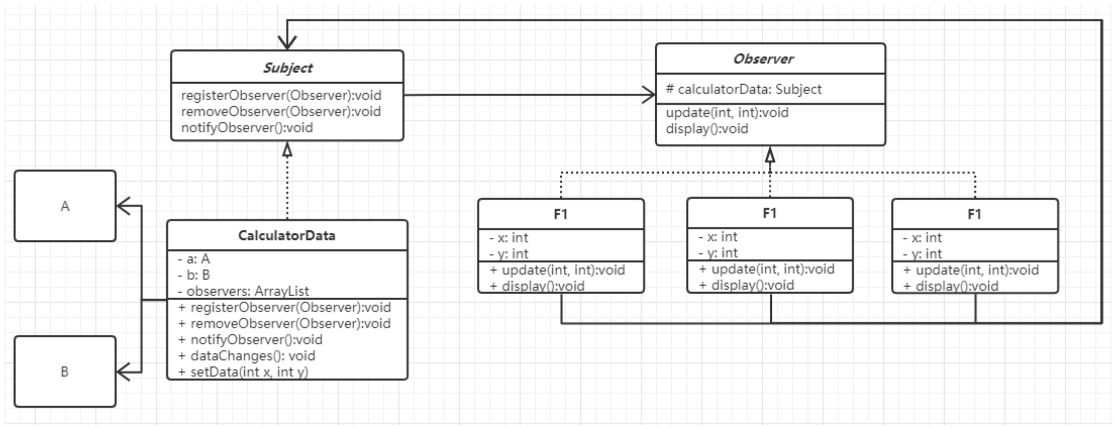
(3)可以动态地增加和减少函数计算器。

试完成以下工作：

（1） 画出 UML 类图

（2） 通过修改类 A 和类 B，以及增加适当的类，实现其代码

（1）



（2）

Subject.java

```java
/**
 * Subject / Mediator
 */

public abstract class Subject {
    /**

     * 注册观察者

     * @param o 待注册的观察者对象

     */
    void registerObserver(Observer o){};

    /**

     * 移出观察者

     * @param o 待移出的观察者

     */
    void removeObserver(Observer o){};

    /**

     * 当主题状态改变时，通知所有观察者

     */
    void notifyObserver(){};
}
```

Observer.java

```
/**
 * Observer / Colleague
 */

public abstract class Observer {
    protected Subject calculatorData;

    public Observer(Subject calculatorData) {
        this.calculatorData = calculatorData;
    }

    void update(int x, int y){

    }

    void display(){

    }
}
```

CalcalatorData.java

```
import java.util.ArrayList;

/**

 * 计算器，作为中介者模式中的 ConcreteMediator

 */

public class CalculatorData extends Subject {
    private A a;
    private B b;
    private ArrayList<Observer> observers;

    public CalculatorData() {
        observers = new ArrayList<>();
    }

    @Override
    public void registerObserver(Observer o) {
```

```java
        observers.add(o);
    }

    @Override
    public void removeObserver(Observer o) {
        int i = observers.indexOf(o);
        if (i >= 0)
            observers.remove(o);
    }

    @Override
    public void notifyObserver() {
        for (Observer o : observers) {
            o.update(a.x, b.y);
        }
    }

    /**

     * 从客户端得到新数据时，通知观察者

     */
    public void dataChanged() {
        notifyObserver();
    }

    /**

     * 设置数据

     * @param x
     * @param y
     */
    public void setData(int x, int y) {
        a = new A(x);
        b = new B(y);
        dataChanged();
    }
}


// 数据类

class A {
    int x;

    public A(int x) {
        this.x = x;
    }
}

class B {
    int y;

    public B(int y) {
        this.y = y;
    }
}
```

F1.java

```java
/**

 * F1 计算器

 * f1(x, y) = x + y
 */

public class F1 extends Observer {
    private int x;
    private int y;

    public F1(Subject calculatorData) {
        super(calculatorData);
        calculatorData.registerObserver(this);
    }

    @Override
    public void update(int x, int y) {
        this.x = x;
        this.y = y;
        display();
    }

    @Override
    public void display() {
        System.out.println("f1(" + x + "," + y + ") = " + x + " + " +
y + " = " + (x + y));
    }

}
```

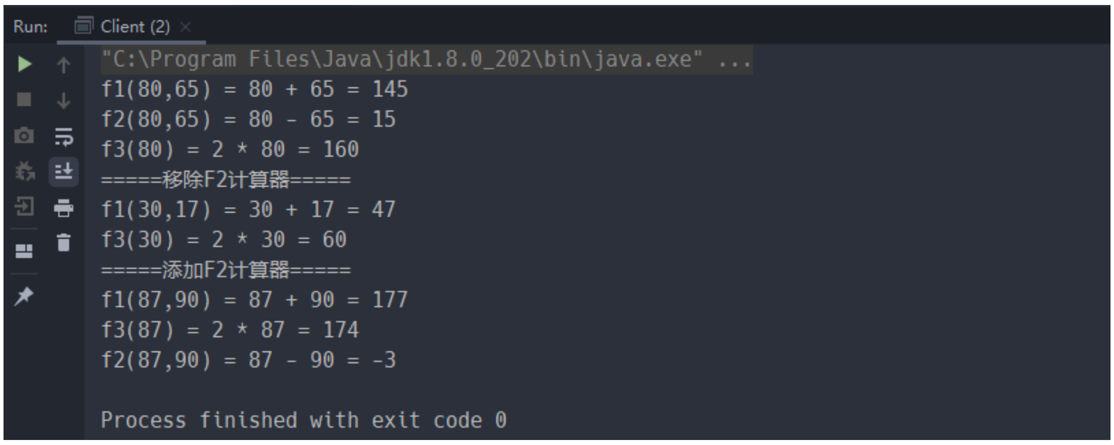
F2.java

```java
/**

 * F2 计算器

 * f2(x, y) = x - y
 */

public class F2 extends Observer {
    private int x;
    private int y;

    public F2(Subject calculatorData) {
        super(calculatorData);
        calculatorData.registerObserver(this);
    }

    @Override
    public void update(int x, int y) {
        this.x = x;
        this.y = y;
        display();
    }

    @Override
    public void display() {
        System.out.println("f2(" + x + "," + y + ") = " + x + " - " +
y + " = " + (x - y));
    }
}
```

F3.java

```java
/**

 * F3 计算器

 * f3(x) = 2 * x
 */

public class F3 extends Observer {
    private int x;

    public F3(Subject calculatorData) {
        super(calculatorData);
        calculatorData.registerObserver(this);
    }

    @Override
    public void update(int x, int y) {
        this.x = x;
        display();
    }

    @Override
    public void display() {
        System.out.println("f3(" + x + ") = 2 * " + x + " = "
+ (2 * x));
    }
}
```
客户端：
```java
public class Client {

    public static void main(String[] args) {
        CalculatorData data = new CalculatorData();

        F1 f1 = new F1(data);
        F2 f2 = new F2(data);
        F3 f3 = new F3(data);

        data.setData(80, 65);


        // 移除观察者 f2 函数计算器

        data.removeObserver(f2);

        System.out.println("=====移除 F2 计算器=====");

        data.setData(30, 17);


        // 重新注册 f2

        data.registerObserver(f2);

        System.out.println("=====添加 F2 计算器=====");

        data.setData(87, 90);
    }
}
```

结果：



```
"C:\Program Files\Java\jdk1.8.0_202\bin\java.exe" ...
f1(80,65) = 80 + 65 = 145
f2(80,65) = 80 - 65 = 15
f3(80) = 2 * 80 = 160
=====移除F2计算器=====
f1(30,17) = 30 + 17 = 47
f3(30) = 2 * 30 = 60
=====添加F2计算器=====
f1(87,90) = 87 + 90 = 177
f3(87) = 2 * 87 = 174
f2(87,90) = 87 - 90 = -3

Process finished with exit code 0
```