《软件体系结构与设计模式》

作业报告

作业名称：作业 2 建造者模式分析与应用

授课教师：毛斐巧

报告人：郑彦薇 学号：2020151022 班级：软件工程 01 班

报告提交时间：2023/5/27

成　绩：

1.作业内容与要求：

计算机组装工厂可以将 CPU、内存、硬盘、主机、显示器等硬件设备组装在一起，构成一台完整的计算机，且构成的计算机可以是笔记本，也可以是台式机，还可以是不提供显示器的服务器主机。对于用户而言，无须关心计算机的组成设备和组装过程，工厂返回给用户的是完整的计算机对象。简述你所理解的建造者模式，并使用建造者模式实现计算机组装过程，要求绘制类图并给出代码框架。

2.解答报告正文

2.1. 什么是建造者模式

建造者模式也称生成器模式，是最复杂的创建型模式。它将一个复杂对象的构建与它的表示分离，使得同样的构造过程可以创建不同的表示。建造者模式关注该复杂对象是如何一步步创建而成的，对于客户端而言，无需直到复杂对象的内部组成部分与装配方式，只需要知道创造者的类型即可。

2.2. 计算机组装过程的实现

（1）问题分析：

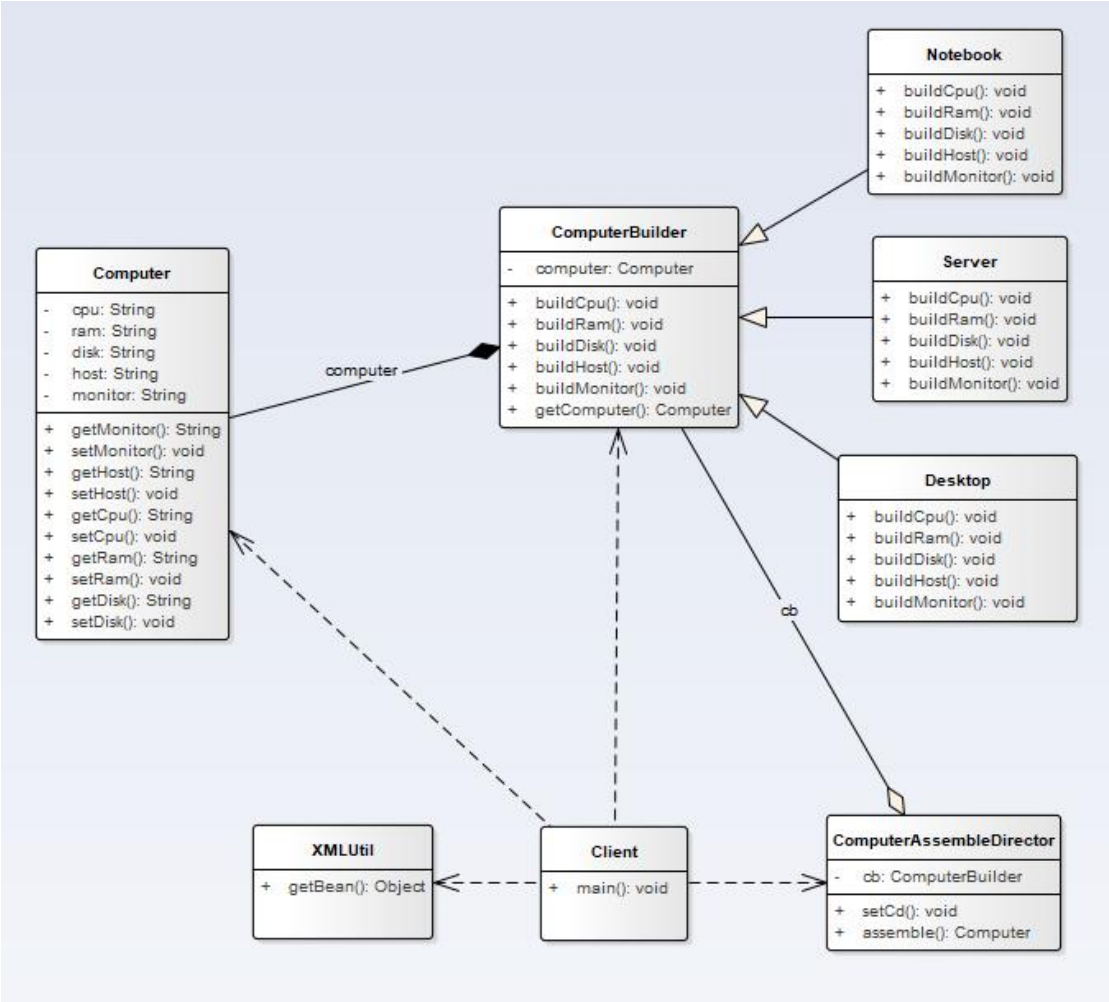
对于计算机组装工厂，其需进行的工作是构建出一个复杂的计算机对象给用户。而对于用户而言，只需要知道返回的计算机对象是什么，对于组装过程并不需要知道。因此可以使用建造者模式实现。

要使用建造者模式实现该过程，需要定义并实现以下几个类：

- 抽象建造者类 ComputerBuilder：为创建产品对象的各个部件指定抽象接口。在该抽象类中需要声明两类方法——用于创建复杂对象的各个部件的 build 方法以及用于返回复杂对象的 getComputer 方法。
- 具体建造者类 Notebook、Desktop 和 Server：实现 ComputerBuilder 接口，实现各个部件的构造和装配方法如 buildCpu、buildDisk 等方法，定义并明确它所创建的复杂对象，也可以提供一个方法返回创建好的对象。
- 复合产品类 Computer：产品是被构建的复杂对象，包含多个组成部件。比如在计算机的组装过程中需要 CPU、内存、硬盘等组成部件，则在该产品类中定义这些组件的 set 和 get 方法。
- 指挥者类 ComputerAssembleDirector：负责安排复杂对象的建造次序。指挥者和抽象建造者之间存在关联关系，需要在指挥者类的 construct 方法中调用建造者对象的部件构造和装配方法，完成对复杂对象的建造。
- 客户类 Client：指定具体的建造者和指挥者，为指挥者类中的具体建造者对象赋值然后通过指挥者将组装好的计算机对象返回给用户。

另外，还可以定义辅助类 XMLUtil，实现通过 xml 文件来制造不同的计算机。xml 文件中，设置属性表示要建造的计算机，可以通过更改 xml 文件中的属性，观察用户类是否能获取到不同计算机及其组装。

（2）类图：



（3）代码框架：

抽象建造者类 ComputerBuilder：

```
public abstract class ComputerBuilder {
    protected Computer computer = new Computer();
    public abstract void buildCpu();
    public abstract void buildRam();
    public abstract void buildDisk();
    public abstract void buildHost();
    public abstract void buildMonitor();

    public Computer getComputer() {
        return computer;
    }
}
```

具体建造者类 Notebook（其他两个类同理）：

```
public class Notebook extends ComputerBuilder{
    public void buildCpu(){
        computer.setCpu("Notebook's Cpu");
    }
```

```
    public void buildRam(){
        computer.setRam("Notebook's 内存");
    }
    public void buildDisk(){
        computer.setDisk("Notebook's 硬盘");
    }
    public void buildHost(){
        computer.setHost("Notebook's 主机");
    }
    public void buildMonitor(){
        computer.setMonitor("Notebook's 显示器");
    }
}
```

对于具体建造者类 Server，因为它没有显示器，所以 buildMonitor 设置为空：

```
public void buildMonitor(){
    //服务器没有主机，所以为空方法
}
```

复合产品类 Computer：

```
public class Computer {
    private String cpu;
    private String ram;//内存
    private String disk;//硬盘
    private String host;//主机
    private String monitor;//显示器

    public String getMonitor() {
        //如果为空，则抛出异常，交给调用者处理
        if (monitor == null) {
            throw new RuntimeException("服务器没有显示器");
        }
        return monitor;
    }
    public void setMonitor(String monitor) {
        this.monitor = monitor;
    }
    public String getCpu() {
        return cpu;
    }
    public void setCpu(String cpu) {
        this.cpu = cpu;
    }
    //...其他组件 set 和 get 方法同理
}
```

指挥者类 ComputerAssembleDirector：

```
public class ComputerAssembleDirector {
    ComputerBuilder cb;
    public void setCb(ComputerBuilder cb){
        this.cb = cb;
    }
    public Computer assemble(){
        cb.buildCpu();
        cb.buildRam();
        cb.buildDisk();
        cb.buildHost();
        cb.buildMonitor();
        return cb.getComputer();
    }
}
```

客户类 Client：

```
public class Client{
    public static void main(String[] args){
        ComputerBuilder cb = (ComputerBuilder) XMLUtil.getBean();
        ComputerAssembleDirector director = new ComputerAssembleDirector();
        director.setCb(cb);
        Computer computer = director.assemble();
        System.out.println("计算机组成");
        System.out.println(computer.getCpu());
        System.out.println(computer.getRam());
        System.out.println(computer.getDisk());
        System.out.println(computer.getHost());
        try{
            System.out.println(computer.getMonitor());
        }catch (Exception e){
            System.out.println(e.getMessage());
        }
    }
}
```

Xml 文件以及辅助类 XMLUtil：

```
<?xml version="1.0" encoding="UTF-8" ?>
<config>
    <className>Notebook</className>
</config>
```

```
public class XMLUtil {
    //从 xml 配置文件中提取具体类类名，并返回一个实例对象
    public static Object getBean(){
        try{
            DocumentBuilderFactory dFactory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = dFactory.newDocumentBuilder();
```

```
            Document doc;
            //从项目根目录开始读取
            doc = builder.parse(new File("src//config.xml"));
            //获取包含类名的文本节点
            NodeList nl = doc.getElementsByTagName("className");
            Node classNode = nl.item(0).getFirstChild();
            String cName = classNode.getNodeValue();
            //通过类名生成实例对象并返回
            Class c = Class.forName(cName);
            Object obj = c.newInstance();
            return obj;

        }catch(Exception e){
            e.printStackTrace();
            return null;
        }
    }
}
```

运行客户端 Client，可以得到结果如下：

```
计算机组成
Notebook's Cpu
Notebook's 内存
Notebook's 硬盘
Notebook's 主机
Notebook's 显示器

进程已结束,退出代码0
```

若修改 xml 文件中的 className 为 Server，可以得到运行结果为：

```
计算机组成
Server's Cpu
Server's 内存
Server's 硬盘
Server's 主机
服务器没有显示器

进程已结束,退出代码0
```