软件体系结构

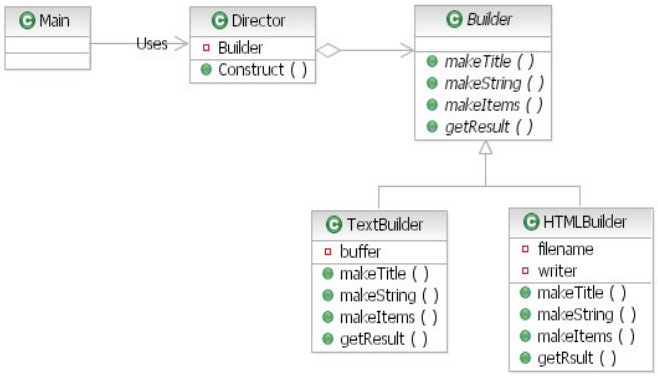
《软件体系结构作业十二》

学　号 22920212204396

姓　名 黄子安

2024 年 5 月 3 日

一、修改本例，增加一个新的 concrete 的 Builder



新增一个 concreteBuilder 用于生成 markdown 格式的文本，首先新建一个 MarkdownBuilder，继承 Builder 并实现其中的各个方法用于文本的创建

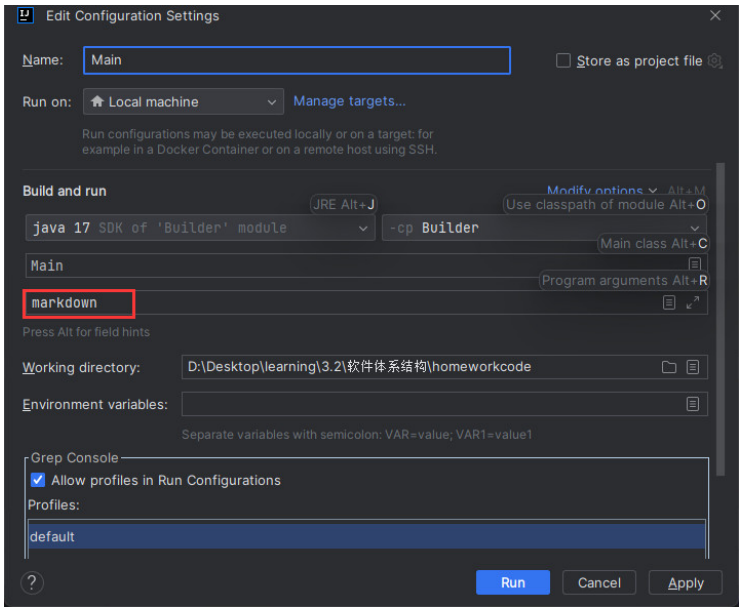
```
package builder;

import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class MarkdownBuilder extends Builder{
    private String filename;                            // 产生的文件名
    private PrintWriter writer;                         // 写入到文件的PrintWriter
    public void makeTitle(String title) {               // Markdown文件的标题
        filename = title + ".md";                       // 根据标题决定文件名
        try {
            writer = new PrintWriter(new FileWriter(filename));  // 建立PrintWriter
        } catch (IOException e) {
            e.printStackTrace();
        }
        writer.println("# " + title);                      // 输出标题
    }
    public void makeString(String str) {                // Markdown文件的字串
        writer.println(str);
    }
    public void makeItems(String[] items) {             // HTML文件的项目
        for (String item : items) {
            writer.println("* " + item);
        }
        writer.println("");
    }
    public Object getResult() {                         // 完成的文件
        writer.close();                                 // 关闭文件
        return filename;                                // 返回文件名
    }
}
```

其中根据 Markdown 的语法规则输出标题和对应的列表选项

之后修改 main 函数，增加生成 Markdown 的命令行参数选项，当传入的参数为 markdown 的时候生成对应的 markdown 文件

```
public class Main {
    public static void main(String[] args) {
        if (args.length != 1) {
            //System.out.println(args.length);
            usage();
            System.exit(0);
        }
        if (args[0].equals("plain")) {
            Director director = new Director(new TextBuilder());
            String result = (String)director.construct();
            System.out.println(result);
        } else if (args[0].equals("html")) {
            Director director = new Director(new HTMLBuilder());
            String filename = (String)director.construct();
            System.out.println("已产生" + filename + "。");
        } else if (args[0].equals("markdown")) {
            Director director = new Director(new MarkdownBuilder());
            String filename = (String)director.construct();
            System.out.println("已产生" + filename + "。");
        } else {
            usage();
            System.exit(0);
        }
    }
    public static void usage() {
        System.out.println("Usage: java Main plain  产生一般格式的文件");
        System.out.println("Usage: java Main html   产生HTML格式的文件");
        System.out.println("Usage: java Main markdown   产生Markdown格式的文件");
    }
}
```

配置运行时的命令行参数，在 IDEA 中可以在运行选项中进行设定



最后运行代码，输出一个 markdown 文件

```
D:\Java17\bin\java.exe "-javaagent:D:\IdeaU\IntelliJ IDEA 2023.1.3\lib\idea_rt.jar=61160:D:\
已产生Greeting.md。

Process finished with exit code 0
```

使用 typora 打开后如下图所示，可以发现 Builder 模式实现循序渐进组合较复杂的对象实例，组合过程的详细处理则隐藏在 Director 里，每个 ConcreteBuilder 包含了创建和装配一个特定产品的所有代码，这些代码只需要写一次；然后不同的 Director 可以在相同部件集合的基础上构造不同的 Product

Greeting

从早上到白天結束

- 早安。
- 午安。

到了晚上

- 晚安。
- 祝你有个好梦。
- 再见。

此外其实感觉用的更多的是简化版的 Builder 模式，通过 lombok 提供的@Builder 注解可以更为方便的构造对象，在构造参数较多且参数有些是可选的时候使用起来更加方便，正好作业写到 Builder 模式顺便了解了下详细的过程，做一个记录

首先新建一个内部 Builder 类，复制构造参数，在其中创建 set 方法，**每次调用返回对应的实例**，之后将原来类的构造函数设为私用并且参数为 builder

```
public class Computer {
    private final String cpu;//必须
    private final String ram;//必须
    private final int usbCount;//可选
    private final String keyboard;//可选
    private final String display;//可选

    private Computer(Builder builder){
        this.cpu=builder.cpu;
        this.ram=builder.ram;
        this.usbCount=builder.usbCount;
        this.keyboard=builder.keyboard;
        this.display=builder.display;
    }
    public static class Builder{
        private String cpu;//必须
        private String ram;//必须
        private int usbCount;//可选
        private String keyboard;//可选
        private String display;//可选

        public Builder(String cup,String ram){
            this.cpu=cup;
            this.ram=ram;
        }

        public Builder setUsbCount(int usbCount) {
            this.usbCount = usbCount;
            return this;
        }
        public Builder setKeyboard(String keyboard) {
            this.keyboard = keyboard;
            return this;
        }
        public Builder setDisplay(String display) {
            this.display = display;
            return this;
        }
        public Computer build(){
            return new Computer(this);
        }
    }
}
```

最后就可以通过链式编程方便的实现构造过程，且可以控制参数个数，这就和 lombok 提供的@Builder 注解功能基本一致，使用起来十分顺畅

```
Computer computer=new Computer.Builder("因特尔","三星")
                .setDisplay("三星24寸")
                .setKeyboard("罗技")
                .setUsbCount(2)
                .build();
```