1~5：CCDDA　　6~10：ABBAA

软件设计与体系结构

注意事项：

1. 请考生按要求在试卷装订线内填写姓名、学号和年级专业。
2. 请仔细阅读各种题目的回答要求，在规定的位置填写答案。
3. 不要在试卷上乱写乱画，不要在装订线内填写无关的内容。
4. 满分 100 分，考试时间为 120 分钟。

题　号	一	二	三	四	五	总　分	统分人
得　分							

得　分	
评分人	

一、单项选择题（共 20 分,每小题 2 分）

1. 下列说法正确的是（　　）。

A.软件设计中要多用继承，少用组合。

B. 设计模式目的在于简化软件系统的设计。

C. 开放封闭原则的关键在于隔离变化， 对程序改动通过增加新代码进行， 而不是修改现有的代码。

D. 一个矩形类中有在界面上绘图的 draw()方法和计算面积的 getArea()方法，该矩形类的设计**不**违反单一职责原则。

2. 某程序中，鸟 Bird 类有飞 fly()方法，企鹅 Penguin 类从 Birde 类继承并重写 fly()方法抛出“企鹅不会飞”的异常信息，该做法违反了（　　）原则。

A. 单一职责　　B. 依赖倒置

C. 里氏替换　　D. 迪米特

3. 某文档编辑器打开文档时，由于从硬盘加载图片比较慢，为迅速打开文档，使用 ImageProxy 类的对象替代真实的图片，而后再由它从硬盘加载真实的图片， 这使用了（　　）设计模式。

A. 适配器　　B. 装饰器

C. 原型　　D. 代理

4. 某游戏兵营对象使用克隆的方式不停地创建战士对象，这使用了（　　）模式。

A. 建造者　　B. 工厂方法

C. 享元　　D. 原型

5. Java 类 HashSet 类实现了 Set 接口，但该类却是通过包装 HashMap 类的对象实现的， 即将 HashMap 转换成了 Set 的接口，这使用了（　　）设计模式。

A. 适配器　　B. 外观

C. 装饰器　　D. 代理

6.Andriod 开发中， 警告框 AltertDialog 由标题 title、内容 message、各种按钮等部件组成，创建 AlterDialog 时使用 AlterDialog.Builder 类通过 setTitle()、setMessage()、setPositiveButton()、setNegativeButton()等方法构造各个部件，然后使用 create()方法得到创建的 AltertDialog 对象，实现了复杂对象的构建和它的表示分离，这使用了（　　）设计模式。

A. 建造者　　B. 组合

C. 工厂方法　　D. 模版

7.有一编译子系统包含了 Scanner、Parser、ProgramNode 等类，由于大多数编译器用户不关心语法分析和代码生成等细节，因此提供了一个高层的接口 Complier 类，该类使编译子系统更加容易使用，这采用了（　　）模式。

A. 桥接　　B. 外观

C. 解释器　　D. 中介者

8.某一图形界面中， 窗口上的组件有依赖关系， 例如某个文本框为空时， 某个按钮不能使用；某个下拉框选中一项，选中内容需要出现在另一个文本框中等等， 因此单独封装了一个类负责控制和协调这些组件的交互，各组件不需要显式地相互引用，从而使其耦合松散，这是采用了（　　）设计模式。

A. 外观　　B. 中介者

C. 桥接　　D. 职责链

9.某一图形编辑器， 每次操作图形前都必须保存当前状态， 以便支持取消操作， 则可使用（　　） 设计模式。

A. 备忘录　　B. 迭代器

C. 命令　　D. 访问者

10. Java 语言为了节省程序内存提高程序性能， String 类设计中，开辟了一块 String 类型的常量池，String 赋值的时候，如果常量池中已经有了该字符串，则不会重新创建对象，而是直接将其指向常量池中的对象，这是采用了（　　）设计模式。

A. 享元　　B. 模版方法

C. 外观　　D. 原型

1、 依赖倒转
2、 模型、控制器
3、 访问者
4、 骨架、子类
5、 装饰器
6、 职责链
7、 Command、Receiver

得分	
评分人	

二、 填空题（共 10 分,每小题 1 分）

1._____原则：抽象不应该依赖细节，细节应该依赖抽象。

2. MVC 是指_____、视图、 _____ 。

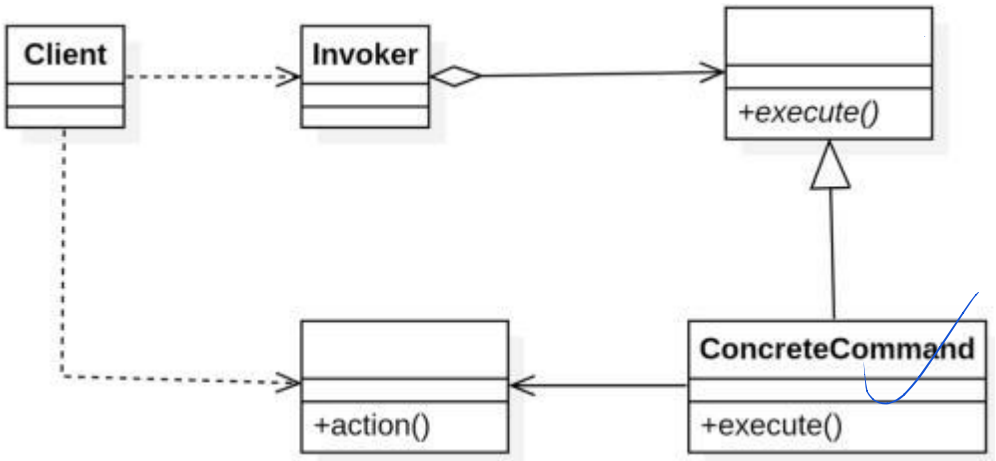
3._____模式能够在不改变各元素类的前提下定义作用于这些元素的新操作，体现了双分派的技术。

4.模版方法模式：定义一个操作中的算法的_____而将一些步骤延迟到_____中实现。

5.Java 类库中 IO 流的设计采用了_____设计模式。

6._____模式使多个对象都有机会处理请求，从而避免了请求的发送者和接收者之间的耦合关系。将这些对象连成了一条链并沿着这条链传递该请求， 直到有对象处理它为止。

7. 命令设计模式的结构中有 Client、Invoker、Command、ConcreteCommand、Receiver 类，请将类名补充到下面类图中未写类名的类中。



得　分	
评分人	

三、简答题（共 15 分,每小题 3 分）

1.简述单一职责原则

2.简述依赖倒置原则

3. 简述合成/聚合复用原则

4.简述模版方法模式的意图

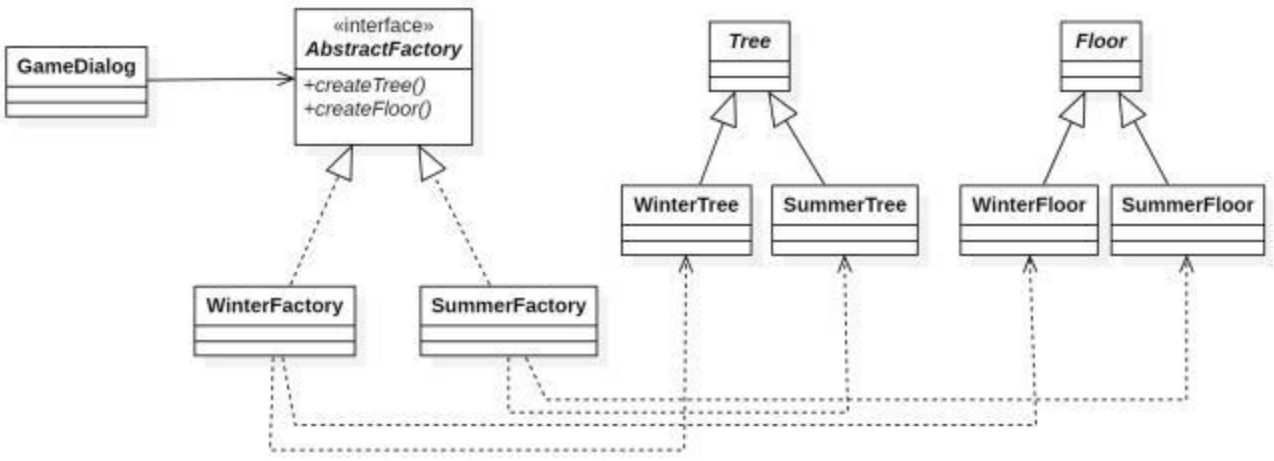
5.简述组合设计模式的意图

得　分	
评分人	

四、程序填空（共 25 分，第 1 小题 10 分，第 2 小题 15 分）

1.某游戏界面 GameDialog 类中使用不同系列的树 Tree 和土地 Floor 的对象构建游戏场景，例如冬天场景使用冬天系列的树 WinterTree 和土地 WinterFloor，夏天场景使用夏天系列的树 SummerTree 和土地 SummerFloor，为便于产品系列的相互替换， 使用了抽象工厂设计模式。

请在虚线框（共 3 处）中补充完整程序的实现。（10 分）



```
abstract class Tree{};
abstract class Floor{};
class WinterTree extends   Tree{
    public WinterTree(){
        System.out.println("createWinterTree");
    }
}
class SummerTree extends   Tree{
    public SummerTree(){
        System.out.println("createSummerTree");
    }
}
class WinterFloor extends   Floor{
    public WinterFloor(){
        System.out.println("createWinterFloor");
    }
}
class SummerFloor extends   Floor{
    public SummerFloor(){
        System.out.println("createSummerFloor");
    }
}
interface AbstractFactory  {
```

```
}
class WinterFactory implements AbstractFactory{
```

```
}
```

```
class SummerFactory implements AbstractFactory{   //省略   }
class GameDialog{
    AbstractFactory factory;
    Tree tree;
    Floor floor;
    public void setFactory(AbstractFactory factory) {
        this.factory = factory;
    }
    public void init(){
        tree=factory.createTree();
        floor=factory.createFloor();
    }
}
public class Main{
    public static void main(String[] args){
            //创建一个 GameDialog 类的对象,其使用冬天的场景
    }
}
```

2.某赛况发布系统中，GameScore 类中 playingMinute 数据域表示比赛已进行的分钟数，homeScore 数据域表示主队得分，visitorScore 数据域表示客队得分， public void setScore(String gameTime,inthomeScore,int visitorScore) 方法设置比赛得分的变化。采用观察者设计模式，GameScore 类实现 Subject 接口， 比赛得分变化后通知观察者 VoiceScore 类语言播报比分、DisplayScore 类大屏显示比分。此外 GameScore 类采用单例模式，只能有一个实例。

请在虚线框（共 2 处）中补充完整程序的实现。（15 分）

```
interface Subject  {
    void registerObserver(Observer o);
    void removeObserver(Observer o);
    void notifyObserver();
}
interface Observer  {
    void update(String gameTime,inthomeScore,int visitorScore);
}
class GameScore implements Subject{
    private String gameTime;//比赛进行时间
    private inthomeScore;//主队得分
    private int visitorScore;//客队得分
```

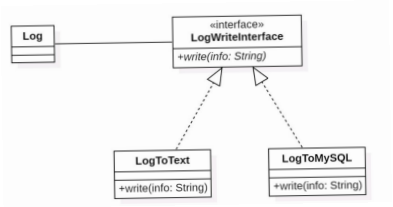
```
    public void setScore(String gameTime,inthomeScore,int visitorScore){
        this.gameTime=gameTime;
        this.homeScore=homeScore;
        this.visitorScore=visitorScore;
        notifyObserver();
    }
    //下框中补充完整单例模式，  public static GameScore getInstance()方法返回唯一的实例
```

```
    //下框中补充观察者模式，定义观察者列表并实现 Subject 接口中的方法
```

```
}
class VoiceScore implements   Observer{//省略}
class DisplayScore implements   Observer{//省略}
```

得　分	
评分人	

五、应用题（共 30 分， 每小题 15 分）



1.某日志 Log 类，类中 writeLog(String info)方法中可使用不同的日志记录策略，例如将 Info 写入数据库中或文本文件中， 日志记录策略需易于切换和扩展， 因此采用策略设计模式。日志记录接口 LogWriteInterface 中有 write(String info)方法， LogToText 类实现该接口将日志信息写入文本文件，LogToMySQL 类实现该接口将日志信息写入 MySQL 数据库。

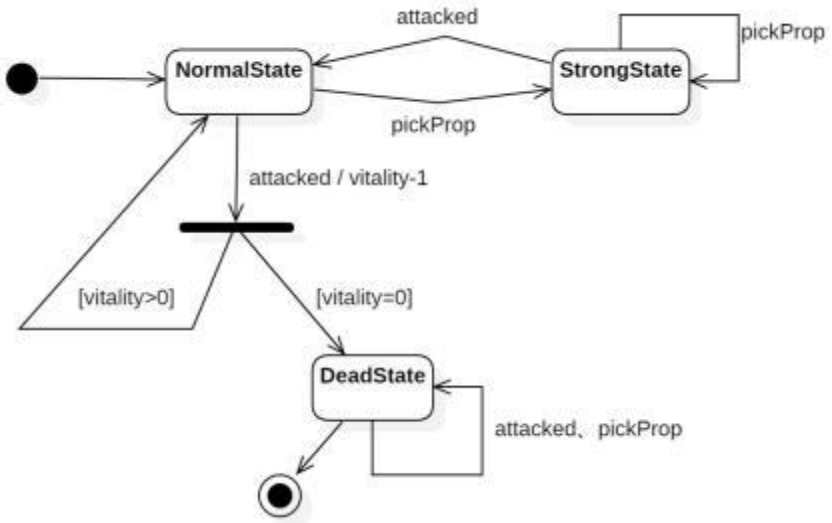
(1)画出 UML 类图。

(2)编写伪代码。

2. 某游戏中，游戏角色 Role 类中有 vitality 数据域（表示生命值，整型， 初始值为 5），attacked()方法表示该角色被攻击， pickProp()方法表示该角色捡到了加强道具，在角色处于不同的状态（常规状态 NormalState、强化状态 StrongState、死亡状态 DeadState）下这两种方法的行为不同，说明如下表：

状态	被攻击 attacked()	捡到加强道具 pickProp()
常规状态 NormalState	生命值 vitality-1，若生命值>0,状态不变，若生命值=0，跳转到死亡状态 DeadState	跳转到强化状态 StrongState
强化状态 StrongState	跳转到常规状态 NormalState	状态不变
死亡状态 DeadState	状态不变	状态不变

UML 状态图如下：



请用状态设计模式解决该问题。

(1) 请画 UML 类图。

（2）请编写伪代码。