1.

重点是空调将来可能会有新模式，要做到开闭原则，当增加新的模式时，不需要使用修改原有的空调类。所以将空调与模式分离，让空调使用抽象出来的模式。于是就可以使用**桥接模式**使得空调类的功能层次结构与模式实现层次结构分离。抽象空调类作为父类拥有模式属性，具体空调可以在抽象空调的基础上增加一些新的功能。而增加新的模式时，只需要实现抽象模式类即可。同时多个空调可能会使用同一个模式，因此可以使用**享元模式**来减少内存开销。

2.

可以使用**适配器模式**，在本例中Target是Linux使得应用程序，Adaptee是原有的Windows程序，Adapter使用Linux的fork方法来创建一个CreateProcess方法，来使得程序适配。

3.

为界面定制一些特效，如带滚动条、 能够显示艺术字体的透明窗体等，也就是需要的是能够为一个已有的界面动态地增加一些效果。**装饰器模式**保证了父类与子类的一致性，可以动态的增加功能，只需要一些定义装饰物品就可以添加许多功能，而且还是在不改变被装饰物的前提下增加功能，被装饰物向外提供的接口仍然保持不变。所以在此处使用**装饰器模式**最为合适。

4.

对于每一个活动节点有多种处理方式，而且还要求方便扩展，可以使用**访问者模式。**Element就是节点，visitor就是要对节点但进行操作的类，当要扩展新的处理方式时，只需要实现新的visitor即可，做到了对扩展开发，对修改关闭。

5.

（1）需要更换促销方式而少修改原有代码是核心要求。可以使用**策略模式**，将各种促销方式抽象出来作为一个抽象促销类（Strategy），增加新的促销只需要实现抽象促销类。令订单模块作为Context使用Strategy 角色来完成促销。

（2）可以使用**适配器模式**，来调用已有的税率计算器，公司只需要通过Adapter调用这些不同的接口来向target也就是己方公司系统提供统一接口即可。

6.

这些业务的访问过程一致，于是可以使用**模板模式**，在抽象类中完成模板方法，并且声明在模板方法中所使用的抽象方法。对于具体的子类再去实现这些抽象方法即可。

7.

要求是屏蔽芯片之间的数据交互，但是又不能阻断芯片的联系，于是使用**中介者（仲裁者）模式**，引入一个中介者，芯片之间的交互通过中介者，而不是相互交互。

8.

因为考虑到敏感词可能会增加，也可能减少，在不同场景下敏感词还可能不同，因此使用**责任链模式**最为合适。要增加屏蔽模式只需要实现Handler抽象类，在不同场景下，可以组织不同的责任链来满足要求。

9.

（1）**命令模式**，因为需求是记录动作，容易混淆的是备忘录模式，虽然备忘录模式也可以撤销重做，但是此处不适用，因为备忘录模式记录的是状态。在本例将每个动作抽象为一个命令类更为合适，用一个栈记录这些动作即可实现撤销，重做。

（2）**状态模式**，一个状态就是一个类，在本例中为了避免复杂的逻辑判断，可以利用状态模式将图像操作与对应状态绑定起来。再根据图像特征更换状态，这样才更为清晰。

（3）**策略模式**，将各种处理算法抽象为一个Strategy抽象类，使用算法是只需要选择对应的具体Strategy，要想增加新算法，也只需要实现新的Strategy类即可。

10．

问题一：

创建型模式包括构造器模式（Builder）、原型模式（Prototype）；

结构型模式包括适配器模式（Adapter）、外观模式（Facade）、代理模式（Proxy）；

行为型模式包括命令模式（Command）、中介模式（Mediator）、状态模式（State）、策略模式（Strategy）。

问题二：

（1）需要更换促销方式而少修改原有代码是核心要求。可以使用**策略模式**，将各种促销方式抽象出来作为一个抽象促销类（Strategy），增加新的促销只需要实现抽象促销类。令订单模块作为Context使用Strategy 角色来完成促销。

（2）可以使用**适配器模式**，来调用已有的税率计算器，公司只需要通过Adapter调用这些不同的接口来向target也就是己方公司系统提供统一接口即可。