## 状态模式：

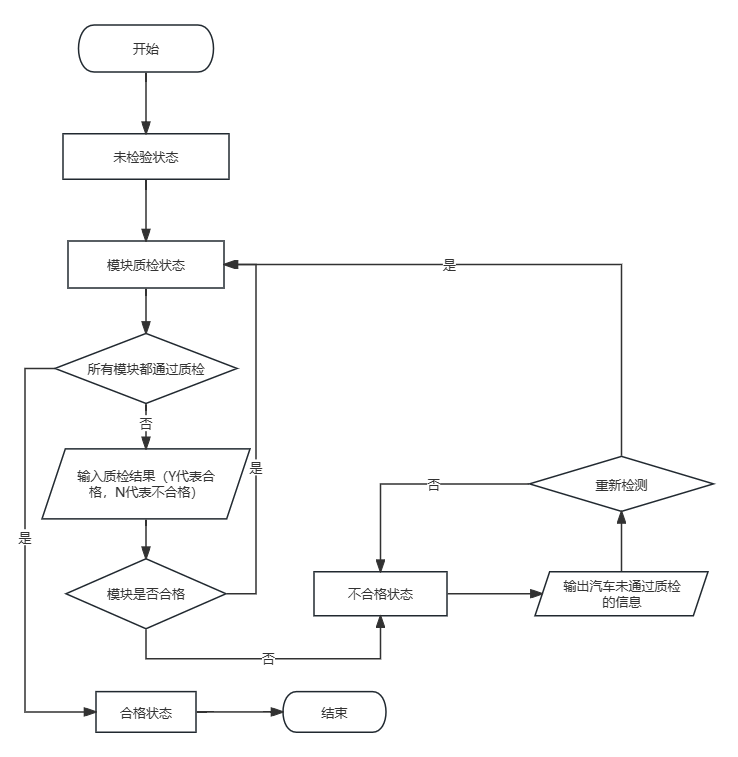
### 实现描述：

1. InspectionState 接口： 定义了状态的接口，包含了 inspect 和 getStateName 两个纯虚函数，负责执行具体状态的行为和获取状态名称。
2. 具体状态类： NotInspectedState、ModuleInspectionState、PassInspectionState、FailInspectionState 分别表示未检验状态、模块质检状态、合格状态和不合格状态。它们实现了 InspectionState 接口中定义的方法。
3. Car 类： 包含了状态切换和状态委托的逻辑。Car 类有一个 currentState 指针，指向当前的质检状态。在每个状态中，质检结果的输入会触发状态的切换或重新检测。

我们在主函数中模拟了汽车组装和质检的整个流程，通过循环调用 Car 类的 inspect 方法，直到汽车达到合格状态。

### 流程图及UML图：

**流程图：**



**类图：**

图示

描述已自动生成

**状态图：**

图示, 工程绘图

描述已自动生成

### 代价分析：

1. 可维护性： 状态模式提高了代码的可维护性，因为每个状态都被封装在独立的类中，易于理解和修改。如果需要添加新的状态，只需创建新的状态类并实现接口即可。
2. 可扩展性： 新的状态可以轻松地添加到系统中，而无需修改现有状态类。这使得系统更具弹性，可以适应未来的变化。
3. 复杂性： 虽然状态模式提高了可维护性和可扩展性，但也引入了更多的类和对象，可能增加了系统的复杂性。在简单情况下，可能会显得过于繁琐。
4. 耦合性： 状态模式降低了模块之间的耦合性，每个状态类都相对独立。这有助于实现模块化设计，减少了不同状态之间的直接依赖。