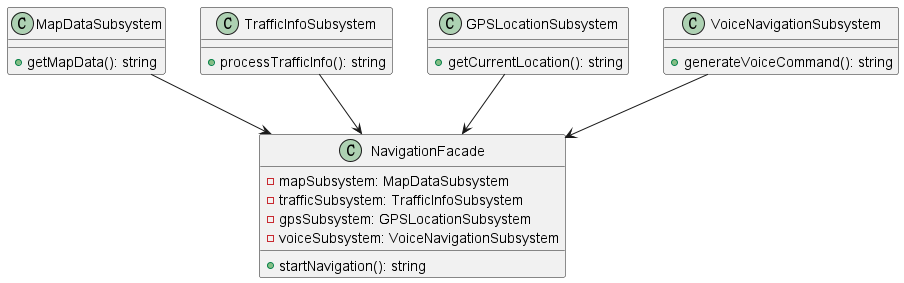
3.1 外观模式

3.1.1 实现描述

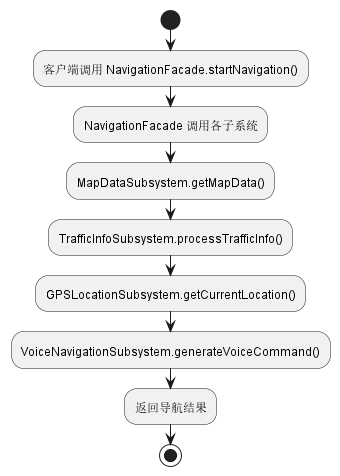
外观（Facade）模式是“迪米特法则”的典型应用，通过为多个复杂的子系统提供一个一致的接口，使这些子系统更加容易被访问。外部应用程序不用关心内部子系统的具体的细节，这样会大大降低应用程序的复杂度，提高了程序的可维护性。

在本系统中，考虑一个新能源汽车的导航系统，包括地图数据获取、实时交通信息处理、GPS定位和语音导航子系统。导航系统可能需要与地图数据提供商的 API 进行交互以获取实时地图数据；处理实时交通信息可能涉及到复杂的算法和数据分析；处理 GPS 定位信息可能包括硬件和软件层面的复杂性；导航系统可能涉及语音合成和导航指令生成。外观类NavigationFacade调用各个子系统的功能，封装子系统的复杂性，提供简单的接口供客户端使用，通过该接口即可启动所有导航相关功能。

3.1.2 类图以及流程图



1. NavigationFacade 是外观类，隐藏了导航系统内部子系统的复杂性，包含了对各个子系统的调用。
2. MapDataSubsystem、TrafficInfoSubsystem、GPSLocationSubsystem、VoiceNavigationSubsystem 是导航系统的子系统，负责处理地图数据获取、实时交通信息处理、GPS 定位、语音导航等功能。
3. 外观类 NavigationFacade 包含了对所有子系统的引用，并提供了 startNavigation 方法，该方法调用了各个子系统的方法来完成导航操作。



3.1.3 代价分析

1. 不符合开闭原则，修改起来特别复杂；考虑需要频繁变更子系统接口时，会导致大量修改。
2. 存在一定性能开销，因为抽象出了一个额外的间接层NavigationFacade
3. 灵活性降低：外观模式可能会使得系统变得不够灵活。如果客户端需要直接访问子系统的某些特定功能，外观模式就不太适用。
4. 新增子系统困难：当需要添加新的子系统或修改现有子系统时，可能需要修改外观类，这可能引入一些风险，尤其是在系统已经投入使用的情况下。

3.1.4 其他内容（如果有的话）

在实际应用中，需要注意外观模式的合理使用。过度使用外观模式可能导致系统设计不当，因此需要根据具体情况仔细评估