Fluent Interface 模式

# 实现功能

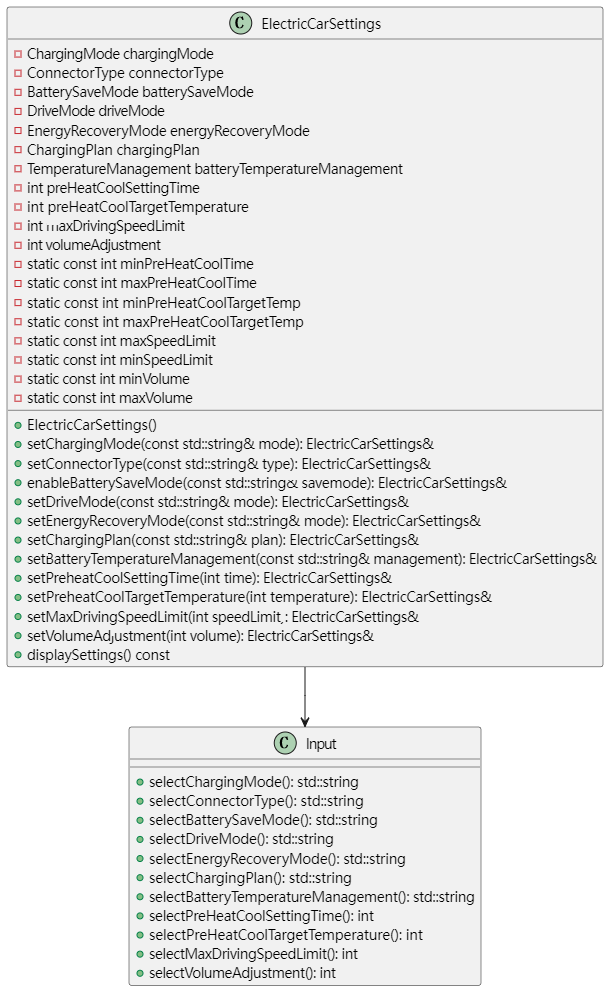
管理电动汽车的各种设置。

包括私有属性来存储不同的设置，比如充电模式、连接器类型、电池保存模式、驾驶模式、能量回收模式、充电计划、电池温度管理，以及预热/预冷设置、目标温度、行驶速度限制和音量调节等数字参数。

提供了设置各种汽车设置的方法，使用流畅接口。

提供了静态方法来从用户处输入不同的汽车设置。

# 类图



ElectricCarSettings 类：

包含了许多属性，例如充电模式、连接器类型、电池保存模式、驾驶模式等等。

这些属性通过对应的方法（setChargingMode、setConnectorType 等等）进行设置，并提供了一系列设置数字参数的方法。

包含了 displaySettings() 方法，用于展示设置的详细信息。

通过构造函数 ElectricCarSettings() 创建对象。

Input 类：

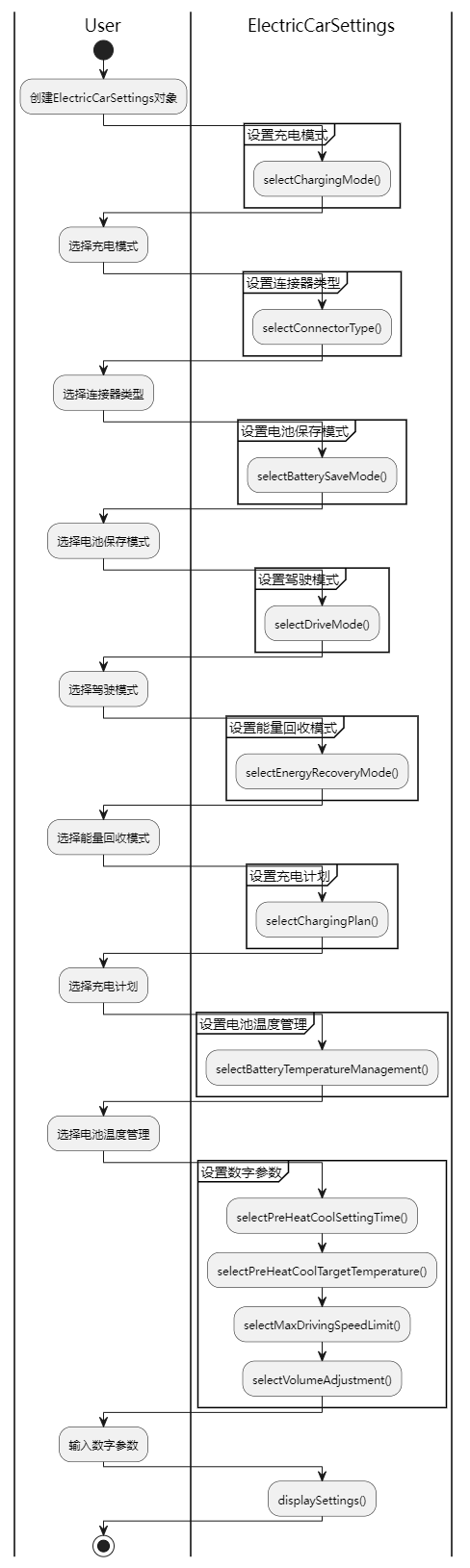
提供了一系列方法用于从用户输入中选择各种设置，例如选择充电模式、连接器类型、驾驶模式等等。

这些方法返回用户所选择的值。

关系：

ElectricCarSettings 类与 Input 类之间存在关联，Input为ElectricCarSettings的友元，表示 ElectricCarSettings 类可以使用 Input 类的方法来获取用户的输入，从而进行设置。

# 流程图



用户开始并创建 ElectricCarSettings 对象。

用户依次选择了充电模式、连接器类型、电池保存模式、驾驶模式、能量回收模式、充电计划和电池温度管理。

用户还输入了一系列数字参数，包括预热/预冷设定时间、预热/预冷目标温度、最大行驶速度限制和音量调节。

每个选择或输入操作都在相应的泳道中与 ElectricCarSettings 类进行交互。

最终，当用户完成所有配置选择后，调用 displaySettings() 方法，将所有配置的参数显示出来。

用户结束流程。

# 代价分析

内存和性能开销：

枚举类型和相关的 std::map 用于存储各种设置的映射关系。这可能会带来一些额外的内存开销，特别是在大量配置项的情况下。

对于每个设置，都有一个额外的 std::string 变量来存储用户选择的内容，并且存储在内存中。

对于数字参数（例如，预热/预冷设定时间、目标温度、行驶速度限制、音量调节），需要额外的整数变量来存储这些值。

复杂性和维护成本：

使用多个 std::map 来维护不同设置的映射，代码变得更加复杂。添加或删除设置可能需要修改多个位置的代码，增加了维护成本。

大量重复的代码，比如 Input 类中的各个选择函数都有类似的结构，可能导致代码重复和维护困难。

错误处理和鲁棒性：

代码中对用户输入的错误处理相对简单，通常只会提示用户输入有效值。更复杂或不合理的输入可能会导致程序崩溃或无法正确工作。

输入和设置之间的映射是硬编码的，如果用户输入的文本与预期的不匹配，程序可能无法正确识别设置，或者会输出错误信息。

可扩展性和灵活性：

如果需要添加更多的设置选项，可能需要修改多个地方的代码。这样的实现方式可能不够灵活，并且不易扩展。

性能方面的考虑：

在使用大量设置时，频繁的搜索 std::map 来转换用户输入可能会引起一些性能损耗，尤其在输入量很大的情况下。