**1. Car.py 文件**

这个文件定义了一个 Car 类。它的主要作用是表示一个汽车对象，每辆车都有一个注册号（carReg）和一个车主（carOwner）。代码中使用了 Python 的封装机制，即使用双下划线（\_\_）来使属性变为私有属性，只能通过 getter 和 setter 方法访问或修改。代码中的一些知识点包括：

* **封装**：通过属性的私有化来保护数据的完整性。
* **属性装饰器**：@property 用来创建 getter 和 setter 方法，这样可以更优雅地访问或修改属性。
* **特殊方法**：\_\_str\_\_ 用来定义对象的字符串表示，\_\_eq\_\_ 用来定义两个对象的比较行为。

class Car:

    def \_\_init\_\_(self,creg):

        self.\_\_carReg = creg

        self.\_\_carOwner = "None"

    @property

    def CarReg(self):

        return self.\_\_carReg

    @CarReg.setter

    def CarReg(self, value):

        self.\_\_carReg = value

    @property

    def CarOwner(self):

        return self.\_\_carOwner

    @CarOwner.setter

    def CarOwner(self, value):

        self.\_\_carOwner = value

    def \_\_str\_\_(self):

        return self.\_\_carReg + " " + self.CarOwner

    def \_\_eq\_\_(self, other):

        return self.CarReg == other.CarReg

**2. CarController.py 文件**

这个文件定义了 CarController 类，负责管理汽车和车主的对象。主要功能包括：

* **创建新汽车和新车主**：通过 newCar 和 newPerson 方法，分别创建 Car 和 Person 对象，并存储在相应的列表中。
* **查找汽车和车主**：findCar 和 findPerson 用于在列表中查找指定的汽车或车主。
* **更改汽车的车主**：changeOwner 方法通过检查当前车主的状态，来为汽车指定一个新的车主。

这是一个典型的控制器类，它管理和协调不同对象之间的交互，体现了**控制器模式**（Controller pattern）。

**3. carGUI.py 文件**

这个文件实现了基于 tkinter 的图形用户界面 (GUI)，提供了一些功能如更改车主和查询车主。代码使用了 tkinter 的一些基础组件，如窗口、小部件（Listbox、Button）以及事件处理功能。

相关知识点包括：

* **图形界面编程**：使用 tkinter 来创建窗口和处理用户交互。
* **事件驱动编程**：通过按钮和输入框与用户交互，并根据用户操作调用控制器类的方法。

**4. Person.py 文件**

这个文件定义了一个 Person 类，表示车主。每个车主都有一个唯一的 ID（personID）和名字（personName），并且可以拥有多辆车。它还定义了车主的增删汽车的方法。

相关知识点：

* **对象关系**：Person 和 Car 类之间存在多对多的关系，一个人可以拥有多辆车。
* **列表操作**：使用列表来存储车主拥有的汽车，体现了对多个对象的管理。

**总结**