SOLID LAB

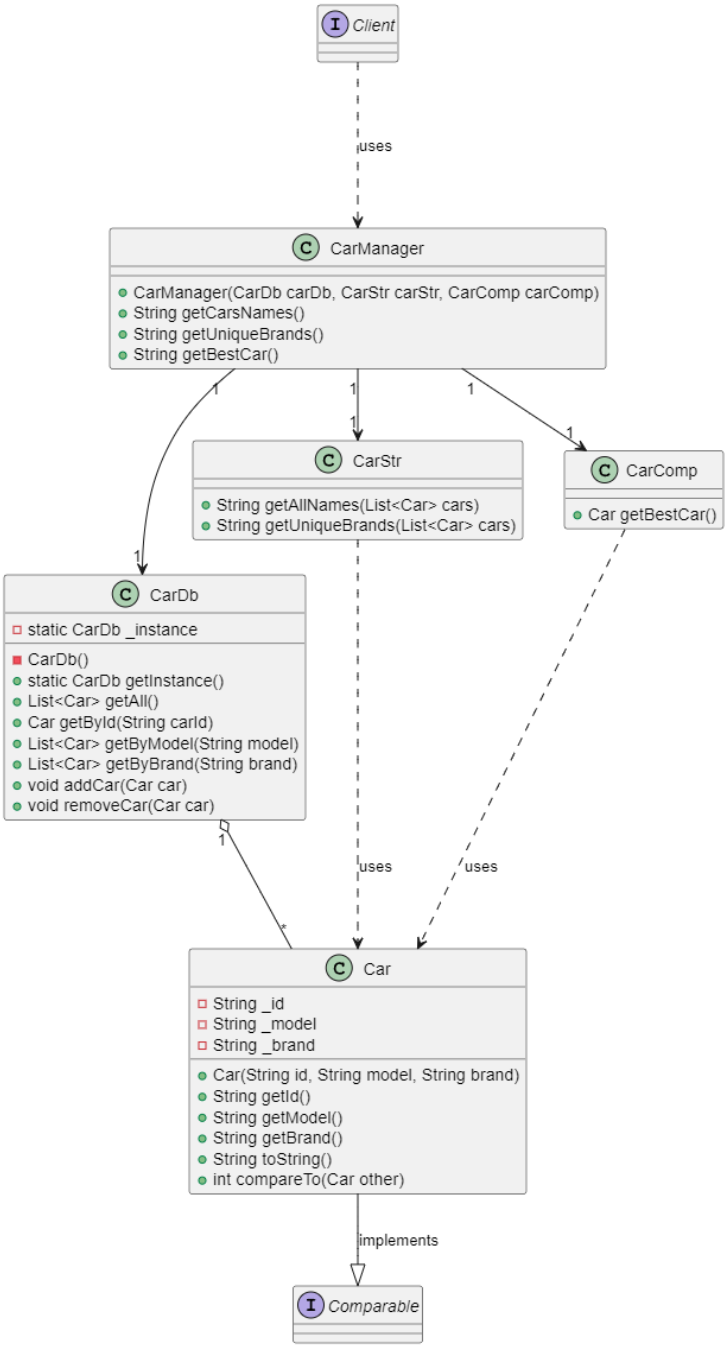
Github repo : https://github.com/idris-saddi/SOLID-Lab

Work done by :

- Idris SADDI
- Aziz Masmoudi

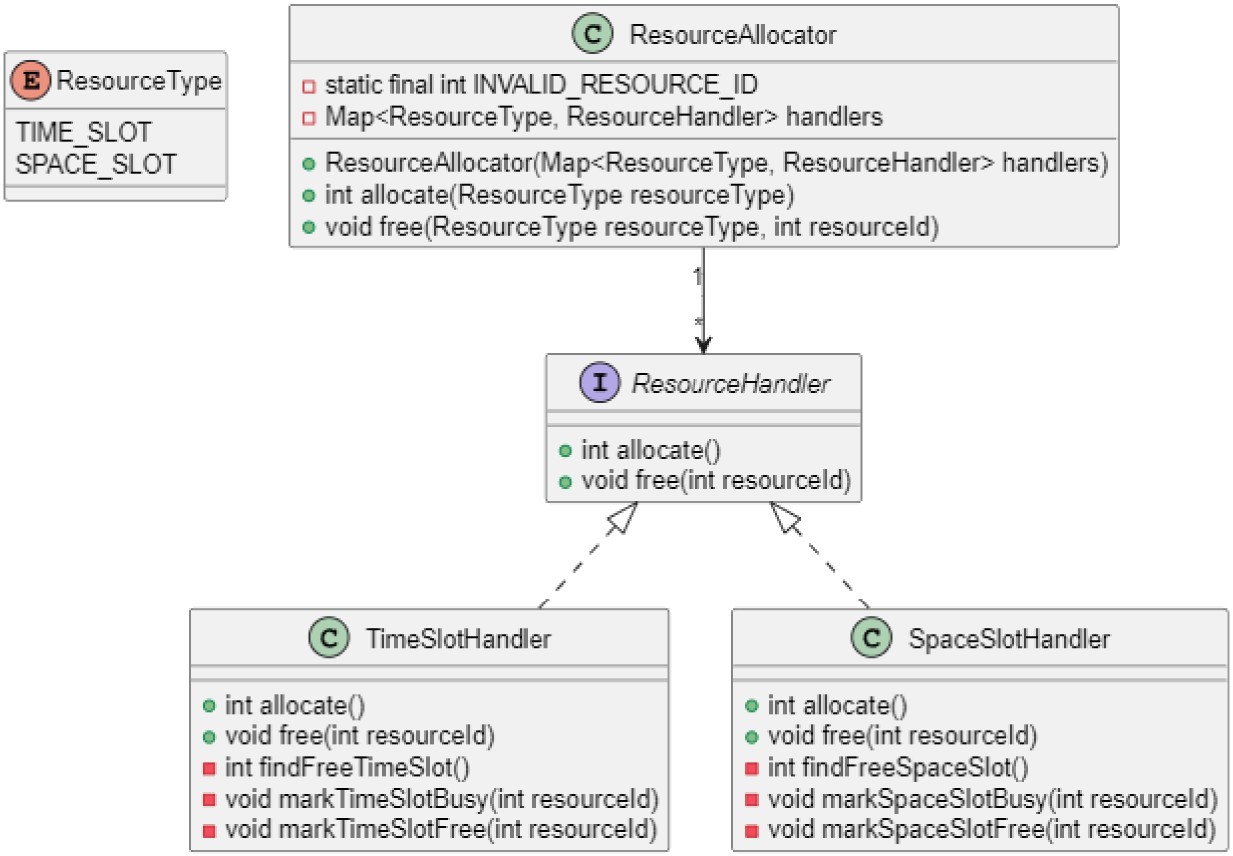
Explanation of SOLID Principles in Each UML Diagram

1. **SRP (Single Responsibility Principle) - refactoredSRP(S)**
   - The classes in this diagram follow SRP by separating concerns:
     - Car handles only car properties and comparison.
     - CarDb is responsible for database interactions.
     - CarStr deals with string representations of cars.
     - CarComp is used for car-related computations.
     - CarManager acts as a coordinator without violating SRP.
   - This ensures that each class has a single, well-defined responsibility.



2. **OCP (Open/Closed Principle) - ResourceAllocator(O)**

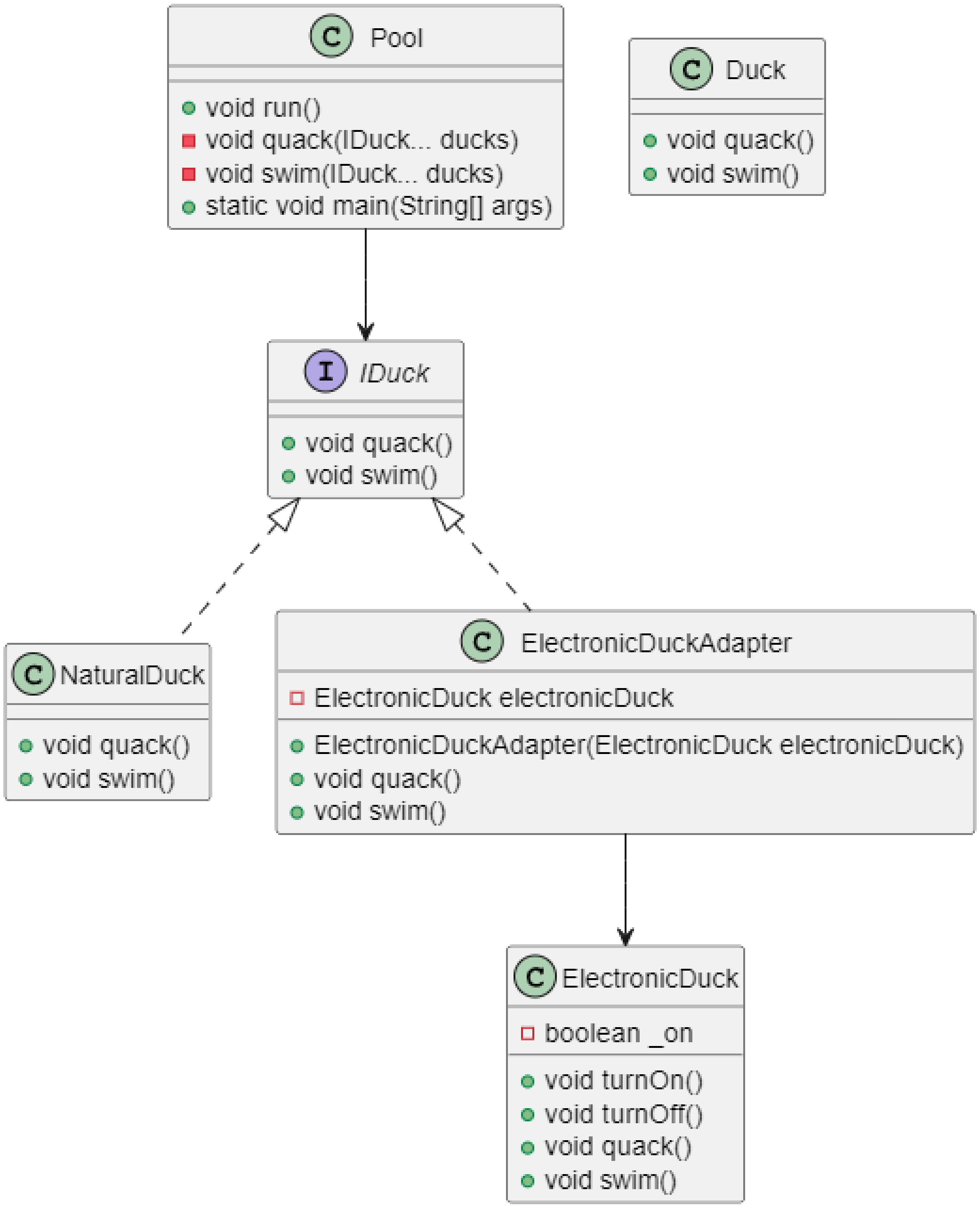
   - The system is open for extension but closed for modification.
   - ResourceAllocator depends on an interface (ResourceHandler), allowing new resource types (e.g., TimeSlotHandler, SpaceSlotHandler) to be added without modifying the allocator itself.
   - This enables easy extensibility while keeping the core logic unchanged.



E ResourceType
TIME_SLOT
SPACE_SLOT
C ResourceAllocator
static final int INVALID_RESOURCE_ID
Map<ResourceType, ResourceHandler> handlers
ResourceAllocator(Map<ResourceType, ResourceHandler> handlers)
int allocate(ResourceType resourceType)
void free(ResourceType resourceType, int resourceId)
1
*
I ResourceHandler
int allocate()
void free(int resourceId)
C TimeSlotHandler
int allocate()
void free(int resourceId)
int findFreeTimeSlot()
void markTimeSlotBusy(int resourceId)
void markTimeSlotFree(int resourceId)
C SpaceSlotHandler
int allocate()
void free(int resourceId)
int findFreeSpaceSlot()
void markSpaceSlotBusy(int resourceId)
void markSpaceSlotFree(int resourceId)

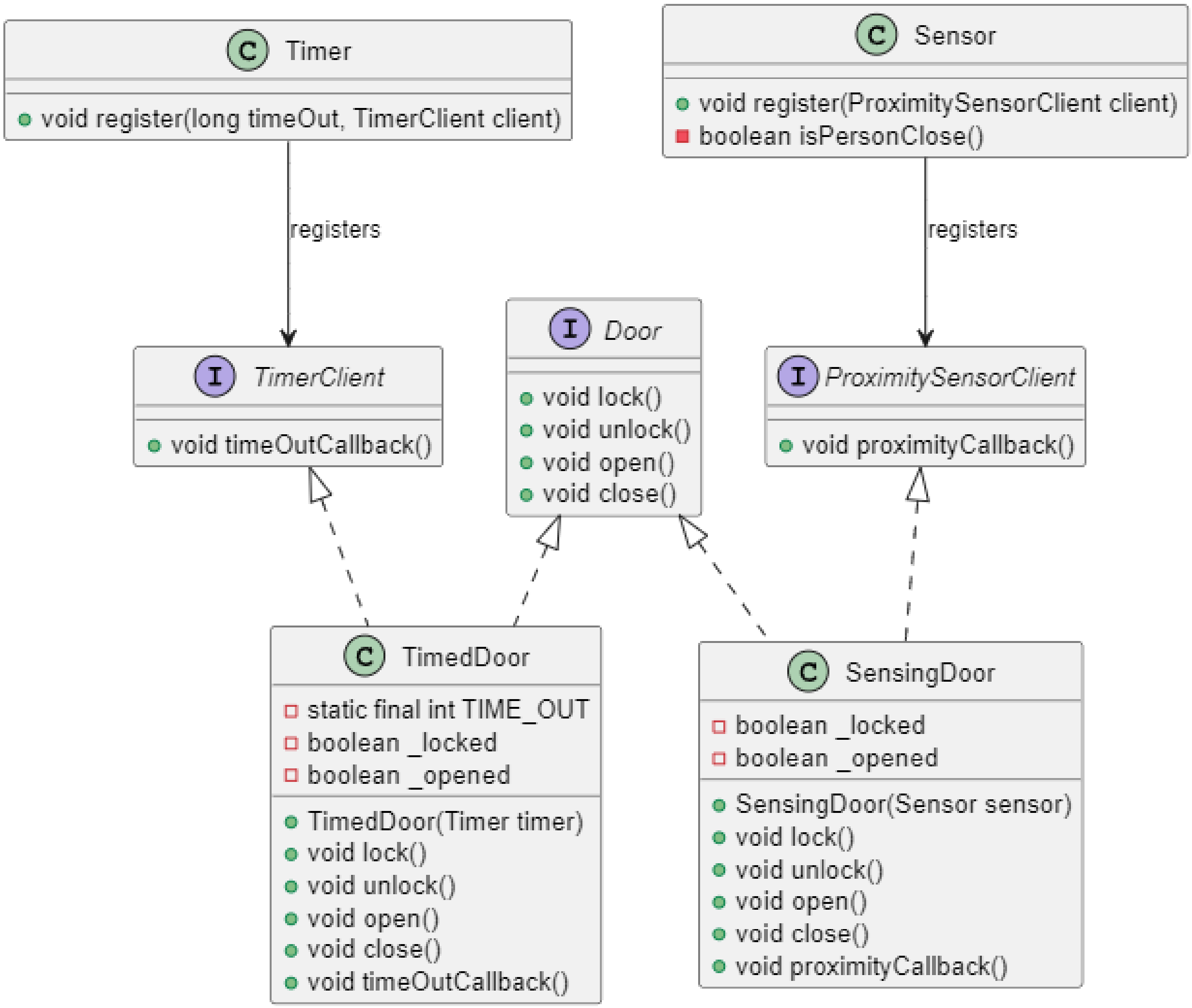
3. **LSP (Liskov Substitution Principle) - DuckSystem(L)**

   - IDuck defines a common interface for all duck types.
   - NaturalDuck and ElectronicDuckAdapter both implement IDuck, meaning they can be used interchangeably in Pool without breaking functionality.
   - ElectronicDuckAdapter adapts ElectronicDuck, ensuring it fits the expected behavior.



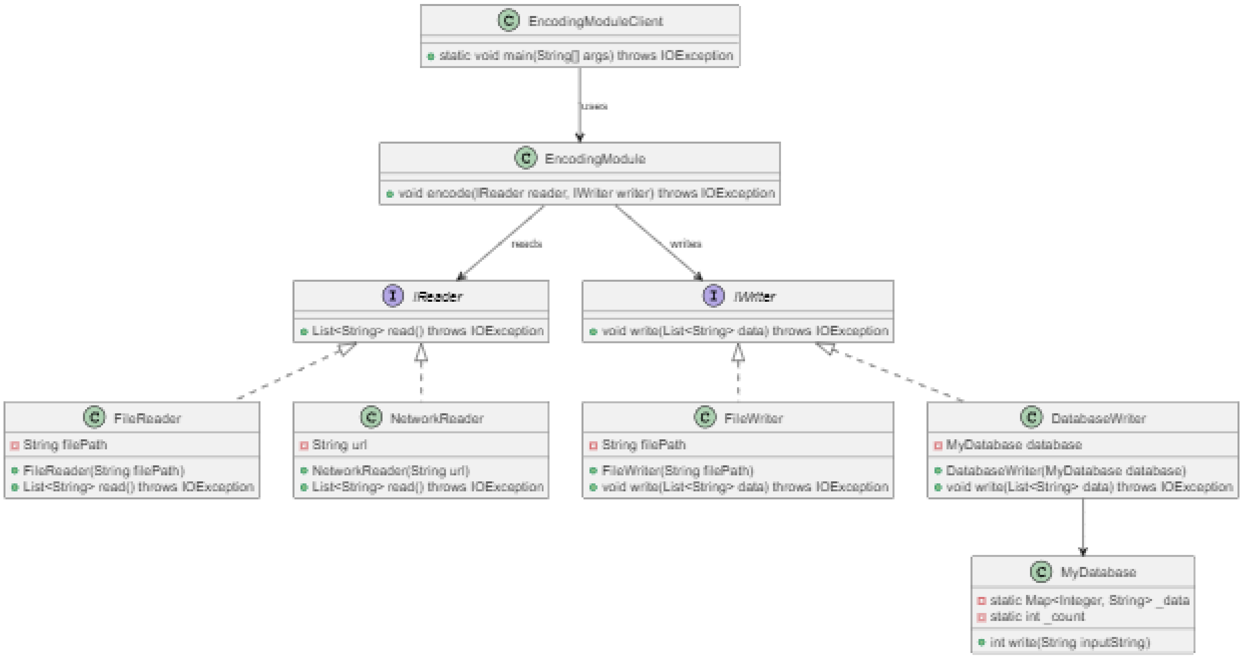
4. **ISP (Interface Segregation Principle) - DoorSystem(I)**

   - Instead of forcing all doors to implement unnecessary methods, separate interfaces (Door, TimerClient, and ProximitySensorClient) are created.
   - TimedDoor only implements TimerClient, and SensingDoor only implements ProximitySensorClient, preventing unnecessary dependencies.
   - This ensures that classes only depend on the methods they actually use.



5. **DIP (Dependency Inversion Principle) - EncodingSystem(D)**

   - EncodingModule depends on abstractions (IReader and IWriter) instead of concrete implementations (FileReader, NetworkReader, FileWriter, DatabaseWriter).
   - This allows for flexibility and easy replacement of data sources without modifying EncodingModule.
   - The high-level module (EncodingModule) is independent of low-level details, following DIP.


C EncodingModuleClient
static void main(String[] args) throws IOException
uses
C EncodingModule
void encode(IReader reader, IWriter writer) throws IOException
reads
writes
I IReader
List<String> read() throws IOException
I IWriter
void write(List<String> data) throws IOException
C FileReader
String filePath
FileReader(String filePath)
List<String> read() throws IOException
C NetworkReader
String url
NetworkReader(String url)
List<String> read() throws IOException
C FileWriter
String filePath
FileWriter(String filePath)
void write(List<String> data) throws IOException
C DatabaseWriter
MyDatabase database
DatabaseWriter(MyDatabase database)
void write(List<String> data) throws IOException
C MyDatabase
static Map<Integer, String> _data
static int _count
int write(String inputString)