Executive Summary

Holiday Travel Vehicles (HTV) is a company specialising in the trading of new and used vehicles and trailers. The company is in the process of upgrading its sales system to improve its efficiency and thus better reflect its business goals. Issues with the previous system stem from the fact that the records for the client's detail and statement are both electronic and paper-based which can lead to several issues including bottlenecking the sales process and data duplication in the record management.

This report contains a deep analysis of the background problem which was used to develop a system design that describes what the new system will contain. The proposed solution that this report entails will feature a completely electronic system with an electronic database for the storage of information. This report also contains candidate class specification, a discussion into design quality and patterns, an overview of the bootstrap process as well as a verification for the design.

Table of Contents

1 Introduction

This Object Design Document describes the problem analysis conducted which identified classes, interfaces, and guidelines required to form the system. This document serves as a reference for developers, managers, and stakeholders in the system to exchange information about the system design and future testing.

1.2 Outlook of solution

The proposed solution assumes that it will be built on top of a database. The database knows nothing of the business design, but allows callers to execute arbitrary SQL code.

When the program is initialised, a database connection is started. When each object that requires a connection to the database is instantiated, the database connection is passed to that object's constructor.

1.3 Trade-offs and object design

1.3.1 Naming Convention

This section is described in subsection 1.4.

1.3.2 Boundary Cases

Inside the User Interface (UI) for any text entry field there should be a fixed maximum length which indicates a boundary.

1.3.3 Invalid Password or Username Exception

When prompted the user enters the employee ID and password into the text field and selects 'enter'. If any of the details are incorrect the system will throw an Invalid User Name or Password Exception.

1.3.4 Invalid Data Exception

This is thrown whenever there data type entered does not match with what is expected inside the field.

1.4 Documentation, and guidelines for interface

Identifier	Rules	Examples

Classes	Names for classes are to be descriptive and simple as possible. Class names are to be capitalised.	class Person
		class PaymentMethod

Interfaces	Like class names, interface identifiers are to be capitalised.	interface Payment
Variables	Variable names are to be short and meaningful and start with an underscore character.	string _fileLocation

1.5 Definitions, acronyms, and abbreviation

Holiday Travel Vehicles	HTV
Sales Information System	SIS
Database	DB
Workflow	WF
Data Model	DM
User Interface	UI

2 Problem Analysis

The Software Requirements Specification included a requirements analysis which revealed a range of functional and quality requirements to build an effective solution going forward. The requirements analysis revealed a list of key functionality that the program needs to perform to meet the specifications of the various use cases.
The key functionality of the application includes:

- Record details for
  - Sales Person
  - Vehicle
  - Vehicle Options
  - Customer
- List customers with outstanding debts
- Amend details for
  - Sales Person
  - Vehicle
  - Vehicle Options
  - Customer
- Cancel a purchase
- Record payment status
- Create Sales Report
- Create an Invoice

2.1 Assumptions

A1	Only cars and trailers will be sold.
A2	A vehicle in this context describes either a car or trailer.
A3	All vehicles have a unique serial number
A4	All vehicles have a name, model, year, manufacturer and a standard price.
A5	A vehicle may be sold to a customer, brought back later as a trade in, and then resold.
A6	Each vehicle sale is for exactly one vehicle, one invoice, one customer and one salesperson.
A7	For a customer to make a purchase, a customer must surrender their name, address, and bank account details to be recorded by HTV.
A8	If a customer has not made a purchase before, they will be assigned a unique customer ID upon their first purchase.
A9	All salespersons, customers, vehicle, sales, and options will be registered in the system.
A10	All customers have a unique id, name, address and phone number.
A11	A trade-in can only occur when a customer purchases a new vehicle.
A12	A customer may trade in no more than one vehicle on a single purchase of a new vehicle
A13	Once the details of a customer, salesperson and vehicle are registered, they can never be removed.
A14	The company sells both new and used vehicles.

A15 The company sells approximately 50 cars each day.
A16 A car is new if the previous owner is the manufacturer.
A17 An instalment shall mean an amount paid towards the total cost of a purchase.
A18 A payment shall mean either an instalment or a purchase that meets the balance.
A19 A customer is permitted to make multiple payment instalments provided the balance is not being outstanding exceeding the 30th day after purchase.
A20 After each payment/instalment a receipt is delivered to the customer.
A21 A salesperson is not needed to make a purchase of a vehicle.
A22 The system resides in a secure environment so that payment information does not have to be encrypted

2.2 Simplifications.

Based on the assumptions made to requirements there were some simplifications made to the project that allowed us to structure the application more effectively:

- Information for cars and trailers will be stored in the same way in the database and be treated as the same in the application as they are traded in the same way.
- Used vehicles are treated the same as new vehicles with the exception of having a "Used" option added to it.

2.3 Design Justification

It is assumed that a database is backing the system. The classes declared here become the interface to the database. These classes could then be used by any other application without knowledge of the database.

The design is such that no redundant information is stored in any of the designated classes, thus, the design achieves 'third order normalisation'.

Each class has been delegated to know some amount of information (which will ultimately be stored in the database); other classes are then delegated the responsibility perform queries on that data to produce information relevant to that class.

2.4 Discarded Class List

- Inventory
  - This class was disregarded from our design because its initial purpose was to be a list of what vehicle stock the company currently has. This responsibility falls with the database as there will be a vehicle object in the program and the database will store a list of these vehicles.
- NewVehicle/TradeIn
  - These classes were to be subclasses of Vehicle. It was noted that NewVehicle had no additional responsibilities, and TradeIn only knew how to find it's previous owner by inspecting the PurchaseRecord history. Therefore, the classes have need to be merged into Vehicle. Asking for the previous owner of a "new vehicle" will correctly return null.

3 Candidate Classes

3.1 Candidate class list

- PurchaseRecord
- Vehicle
- SalesInvoice
- Option
- Menu
- Person
    - SalesPerson
    - Customer
- Payment
- PaymentMethod
    - Cheque
    - Card
    - Account
- Database

3.2 UML Diagram

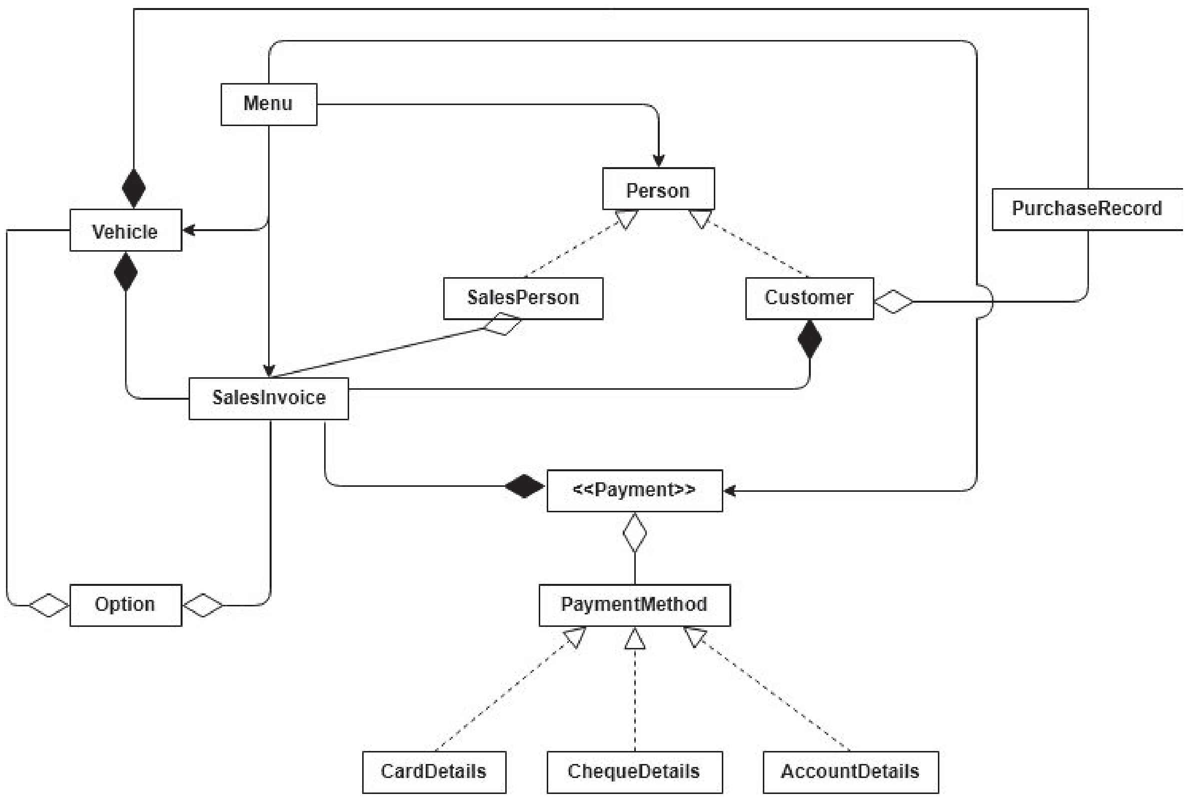


Figure 1: UML Diagram

After conducting an in-depth problem analysis a series of classes were identified through applying separation of concerns in order to develop SIS. This included a menu class that is fundamental to the bootstrapping process. Once instantiated upon runtime, dependant on the task to be achieved (as per Section 4.1 from 'Assignment 1 SRS'), the user selects an option which results in the instantiation of one or more objects.

This has been designed under the assumption that a Sales Invoice generated in the transaction cannot exist without a customer, vehicle, or payment method. These are the minimum requirements necessary for a transaction to go ahead. The payment structure will be designed in accordance with the factory design model.

3.3 CRC Cards

3.3.1 Vehicle

Class Name: Vehicle **Super Class:** -	
A vehicle represents a single vehicle. A vehicle is identified by a unique serial number and there will never be two instances of a vehicle with the same serial number.	
Responsibilities	**Collaborators**
● Knows manufacturer, make, year and serial number	NA
● Knows vehicle type (car/trailer)	NA
● Knows which Options have been installed	Option
● Can determine if currently sold	SalesInvoice, PurchaseRecord
● Can determine who the current owner is (or if it is in inventory)	SalesInvoice, PurchaseRecord

3.3.2 Option

Class Name: Option **Super Class:** -	
Options describe a pre-made selection of items that may be installed into a vehicle (e.g. air-conditioning)	
Responsibilities	**Collaborators**
● Knows id	NA
● Knows name and description	NA
● Knows expected price	NA

3.3.3 SalesInvoice

Class Name: SaleInvoice **Super Class:** -	
SalesInvoice records the history of a single sale and the state of payment on those sales.	
Responsibilities	**Collaborators**
● Knows Sale ID	Payment
● Knows customer, SalesPerson and Vehicle	Customer, SalesPerson, Vehicle
● Knows sale price	NA
● Knows the date of purchase	DateTime
● Knows which Options have been agreed on	Option
● Holds a signature	Image (built-in type)
● Can determine the remaining balance	Payment
● Can print/email an invoice	DocumentPrinter (built-in class)

3.3.4 PurchaseRecord

Class Name: PurchaseRecord **Super Class:** -	
The PurchaseRecord class is designed to keep a record of cars bought by HTV and where that vehicle came from. There is one PurchaseRecord for every time a Vehicle is bought. This information cannot be part of Vehicle as a Vehicle can be sold multiple times	
Responsibilities	**Collaborators**
● Knows vehicle purchased	Vehicle
● Knows the date of purchase	
● Knows a base cost of the vehicle	
● Knows the customer or manufacturer purchased from	Customer

3.3.5 Person

Class Name: Person **Super Class:** -	
A person knows the commonalities between any real human. It is not guaranteed that an instance of person is a unique real person.	
Responsibilities	**Collaborators**
● Knows name, address and phone number	NA

3.3.6 Customer

Class Name: Customer **Super Class:** Person	
A customer is a unique reference to a Customer and any other details that HTV wants to know about that customer (e.g. prefered payment method). The additional information has not been specified so is not listed, but it is noted that the design allows for an arbitrary amount of information to be stored. The Customer class is responsible for determining information related to themselves	
Responsibilities	**Collaborators**
● Knows ID	NA
● Can find a list of traded in vehicles by this customer	PurchaseRecord
● Can find a list of vehicles purchased by this customer	SalesInvoice
● Can determine the balance of this customer	SalesInvoice, Payment

3.3.7 SalesPerson

Class Name: SalesPerson **Super Class:** Person	
Sales Person is responsible for knowing information about a single real SalesPerson.	
Responsibilities	**Collaborators**
● Knows ID	NA
● Can find a list of vehicles sold by this SalesPerson	SalesInvoice

3.3.8 Payment

Class Name: Payment **Super Class:** -	
The payment class records a single instalment towards a car	
Responsibilities	**Collaborators**
● Knows amount paid	SalesInvoice
● Knows payment method	PaymentMethod
● Can print/email receipt for Customer	DocumentPrinter (built-in class)

3.3.9 PaymentMethod

Class Name: PaymentMethod **Super Class:** -	
A Payment Method is an abstraction of some method of payment. It does nothing in itself except provides a common interface for subclasses to implement	
Responsibilities	**Collaborators**
● Can withdraw a given amount	Abstract method for subclass’

3.3.10 CardDetails

Class Name: CardDetails **Super Class:** PaymentMethod	
Card details records how to pay by credit card	
Responsibilities	**Collaborators**
● Knows details of a card	
● Can deduct from a credit card account	

3.3.11 ChequeDetails

Class Name: ChequeDetails **Super Class:** PaymentMethod	
ChequeDetails records details of how to pay by a check	
Responsibilities	**Collaborators**
● Knows cheque details	
● Can deduct from the cheque account	

3.3.12 FinanceAccount

Class Name: FinanceAccount **Super Class:** PaymentMethod	
FinanceAccount records details of how to pay by a finance account.	
Responsibilities	**Collaborators**
● Knows details of a finance account	
● Can deduct from a finance account	

3.3.13 Database

Class Name: Database **Super Class:**	
The database class allows execution of arbitrary SQL. It knows nothing of the business logic. This call will either be provided by a third party library or be a thin wrapper around the external library.	
Responsibilities	**Collaborators**
● Knows connection details	
● Can execute SQL	

3.3.14 Menu

Class Name: Menu **Super Class:**	
The menu class is the overarching control class	
Responsibilities	**Collaborators**
● Interacts with the user	GUI framework (not included in design)
● Controls what other classes do	
● Starts the initialisation process	All other classes

4 Design Quality

4.1 Design Heuristics

H1	A class should capture one and only one key abstraction
H2	All data unique to a class should be kept hidden inside of that class.
H3	All data inside a base class should be declared private and not accessed by any other class.
H4	Employ consistency in language, diagrams, and actions.
H5	Generated errors will be plain in language and proactively suggest a solution.
H6	A derived class should not be capable of overriding a base class method that returns a null operation.
H7	Do not create any ‘super-class’ or a god class.
H8	Base classes should be declared abstract.
H9	Abstract classes should only be base classes.

5 Design Patterns

5.1 Creational Patterns

5.1.1 Factory Method Design Pattern

This pattern is particularly useful for creating a range of different persons who have certain commonalities by different roles and responsibilities. Currently, two subclasses have been identified as required for the current problem definition going forward, however, should additional personnel objects be required (managers, contractors, labourers) this can be of use to Holiday Travel Vehicles in the future thus increasing the lifetime of their product.

The implementation here is the creation of a special factory method instead of a direct object construction method.

5.1.2 Singleton Design Method

This design pattern ensures that only one existence of a class exists in a system. This is applicable when envisioning how the program is initialised and how functionality is controlled and implemented. This can be implemented via calling the default constructor inside this class private to ensure that there is only a single instance of the software.

In addition, this is particularly applicable to the initialisation of the Database which forms the backbone of the information storage of the system.

5.2 Behavioural Patterns

5.2.1 Strategy Design Pattern

This design pattern is specifically aimed at payment implementation. The software reaches the payment interface from which the Payment details are encapsulated in a base method class, and the implementation is contained inside the derived payment method classes; CardDetails and ChequeDetails. There will be no impact to the software when the implementation of one of the derived classes is altered.

5.3 Structural Patterns

5.3.1 Model - View - Controller Pattern

Each object in the software that is carrying data is a model; each vehicle carries unique information to that vehicle in question. The models represented in the program display the appropriate information to the user* and allows them to perform actions based of the view. This is inherent to achieving a workflow through each of the software's functions.

6 Bootstrap Process

This bootstrapping process reflects a salesperson selling a car to a new customer.

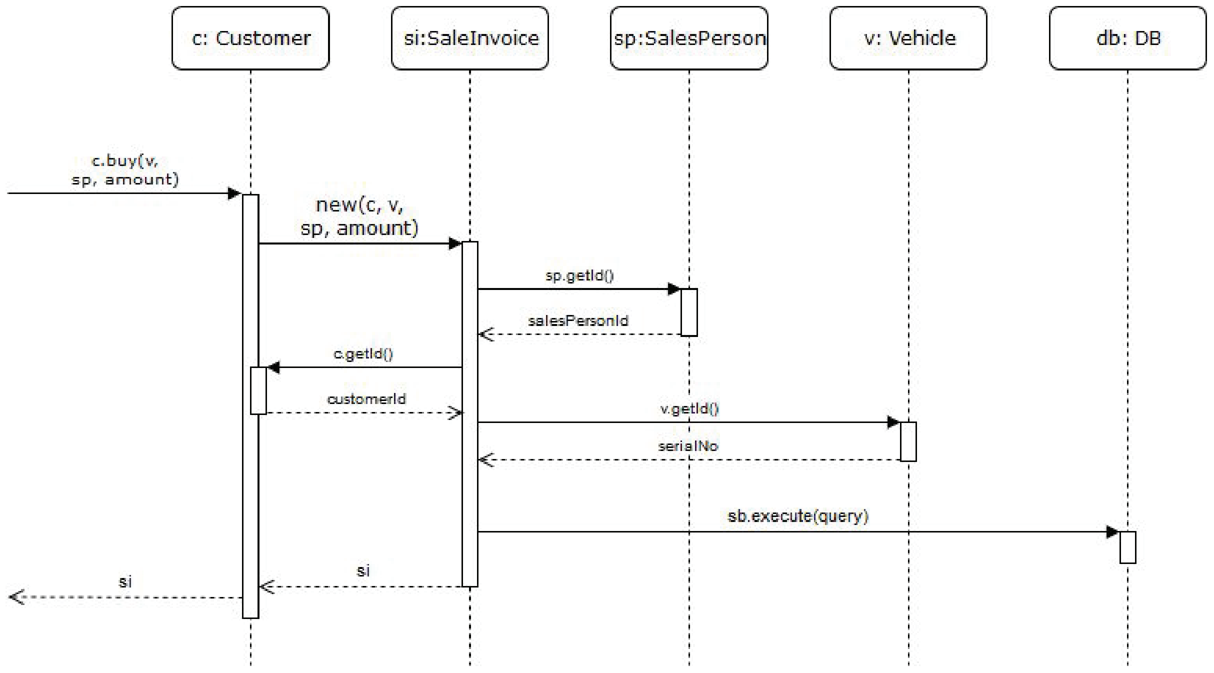
- Main creates an instance of the menu class
- Main creates a SalesInvoice class
- SalesInvoice creates a customer class
- Customer class takes inputted information and pushes it to the database
- SalesInvoice creates Payment class
- Payment class takes inputted information and pushes it to the database
- SalesInvoice will create a vehicle class
- Vehicle will get information from the database
- Vehicle will create a list of Options based on input from the database
- SalesInvoice will create a PurchaseRecord and store this in the database

7 Verification

The proposed design has been verified by simulating several use cases shown below

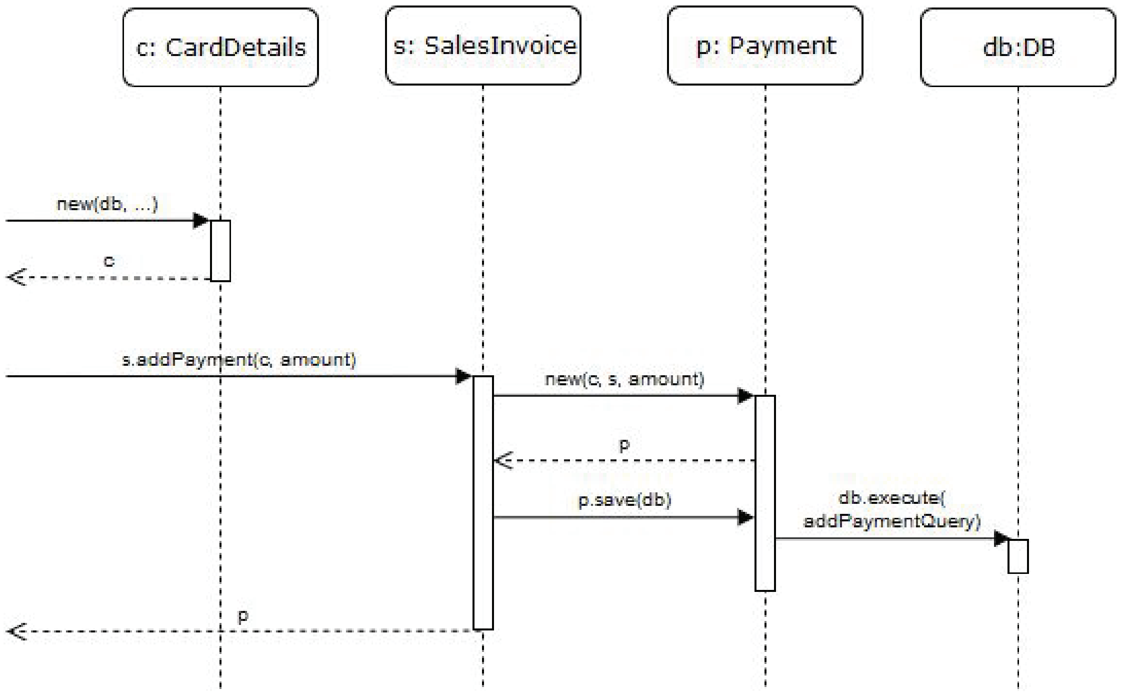
7.1 Customer purchases a car

This situation shows that the interactions needed to purchase a car.



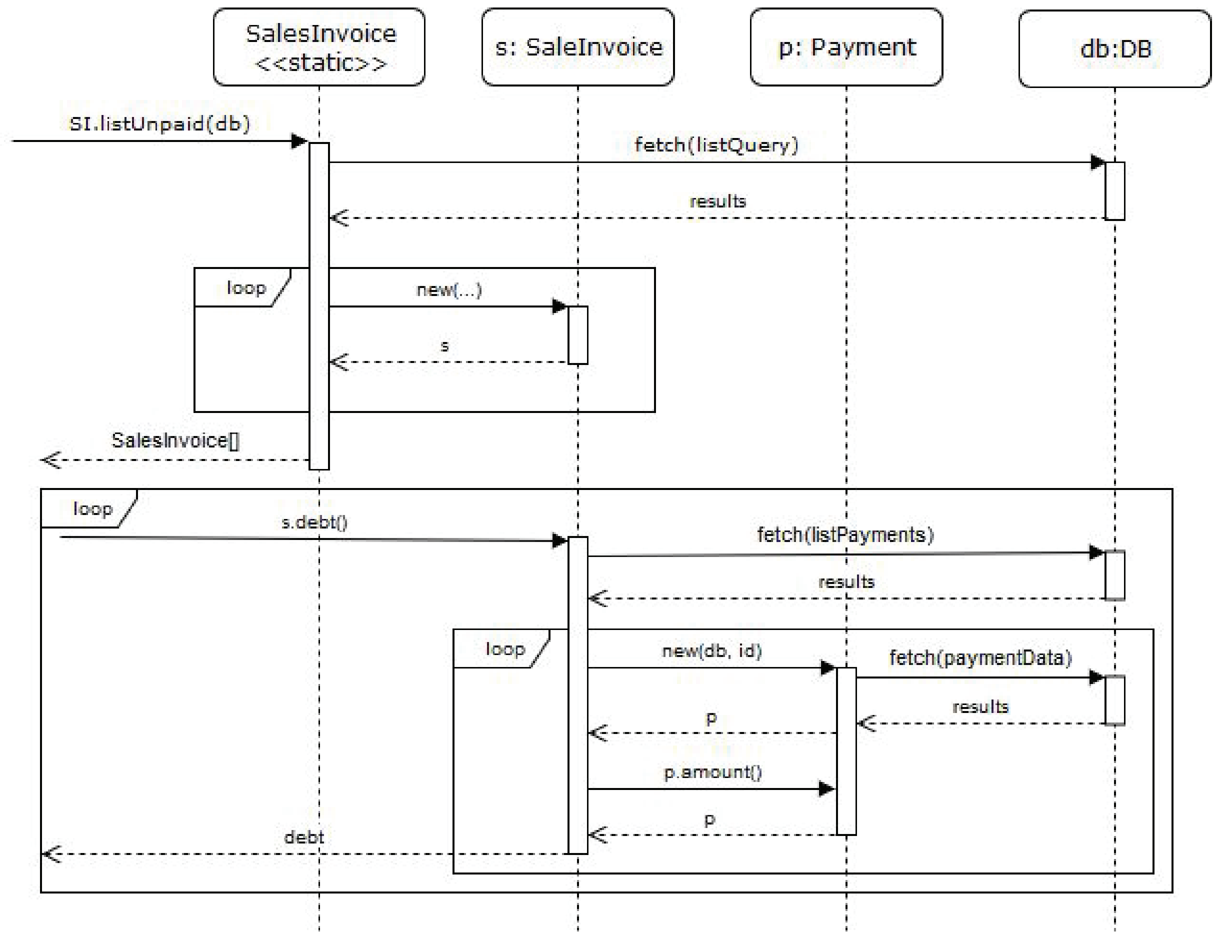
The caller tells the customer to buy the given car, from the given salesperson, for the given amount. The customer creates a new SalesInvoice. The SalesInvoice informs the database of its creation, and uses the various ids of the passed object to build the query string. The SalesInvoice returns itself which can be used for further processing (e.g., adding payments to).

7.2 Add payment to a car



Payments can be added to a sales invoice by creating a card details (entered by the user), then telling the sales invoice to take an amount from that card. The payment object is created as a byproduct, which is returned and can then be used to print/email a receipt.

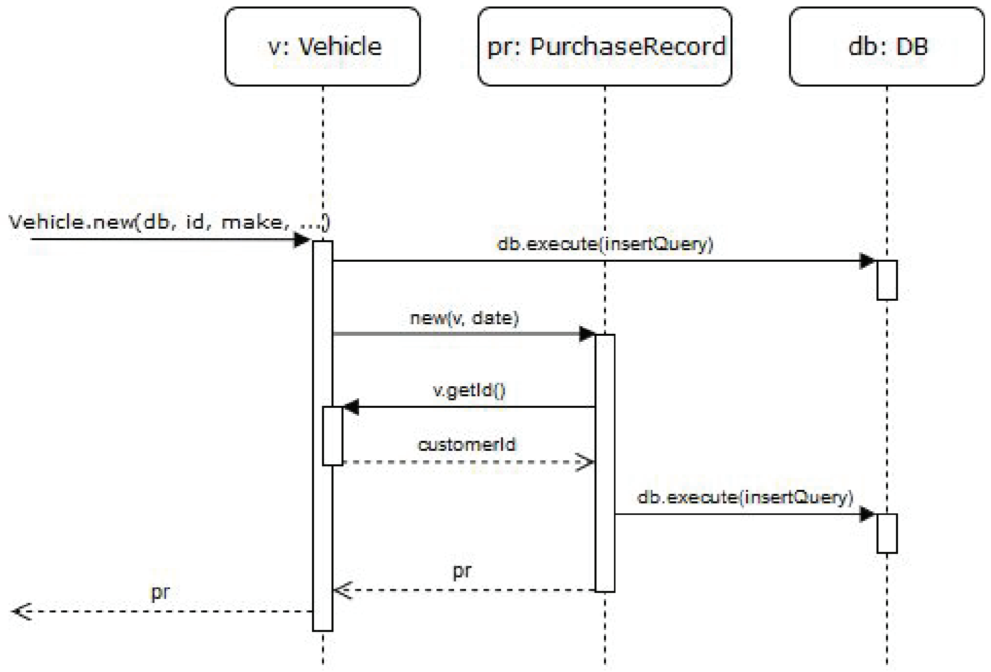
7.3 Calculate outstanding debt



The above diagram shows how to determine the debt of each invoice. The SalesInvoice class is responsible for polling the database to request the indebted SalesInvoices. Each SalesInvoice can then be queried for how much debt it owes individually and the amount summed.

Note that to determine the debt for a single SalesInvoice requires summing each payment. In the above diagram, this operation is done twice (once to calculate which SalesInvoices are in debt, and again to determine the actual debt). The diagram shows one possible implementation that can be used when the debt has not yet been calculated,, but, in practice, the SalesInvoice should cache the results of s.debt() and return it directly.

7.4 Register new vehicle in system



To register a new vehicle, the vehicle can be created by requesting a new vehicle with the given information (entered by the user). The class then registers the vehicles details into the database. In addition, a purchase record is created for the car which is also registered into the database. The created purchase record is returned. If the created vehicle is needed instead of the purchase record, then the vehicle can be requested from the purchase record.

8 References

[1] V. Huston, "Object-Oriented Design Heuristics", Vincehuston.org, 2019. [Online]. Available: http://www.vincehuston.org/ood/oo_design_heuristics.html. [Accessed: 03- May- 2019].

[2] "Design Patterns", Refactoring.guru, 2019. [Online]. Available: https://refactoring.guru/design-patterns. [Accessed: 05- May- 2019].

[3] "Design Patterns MVC Pattern", www.tutorialspoint.com, 2019. [Online]. Available: https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm. [Accessed: 05- May- 2019].