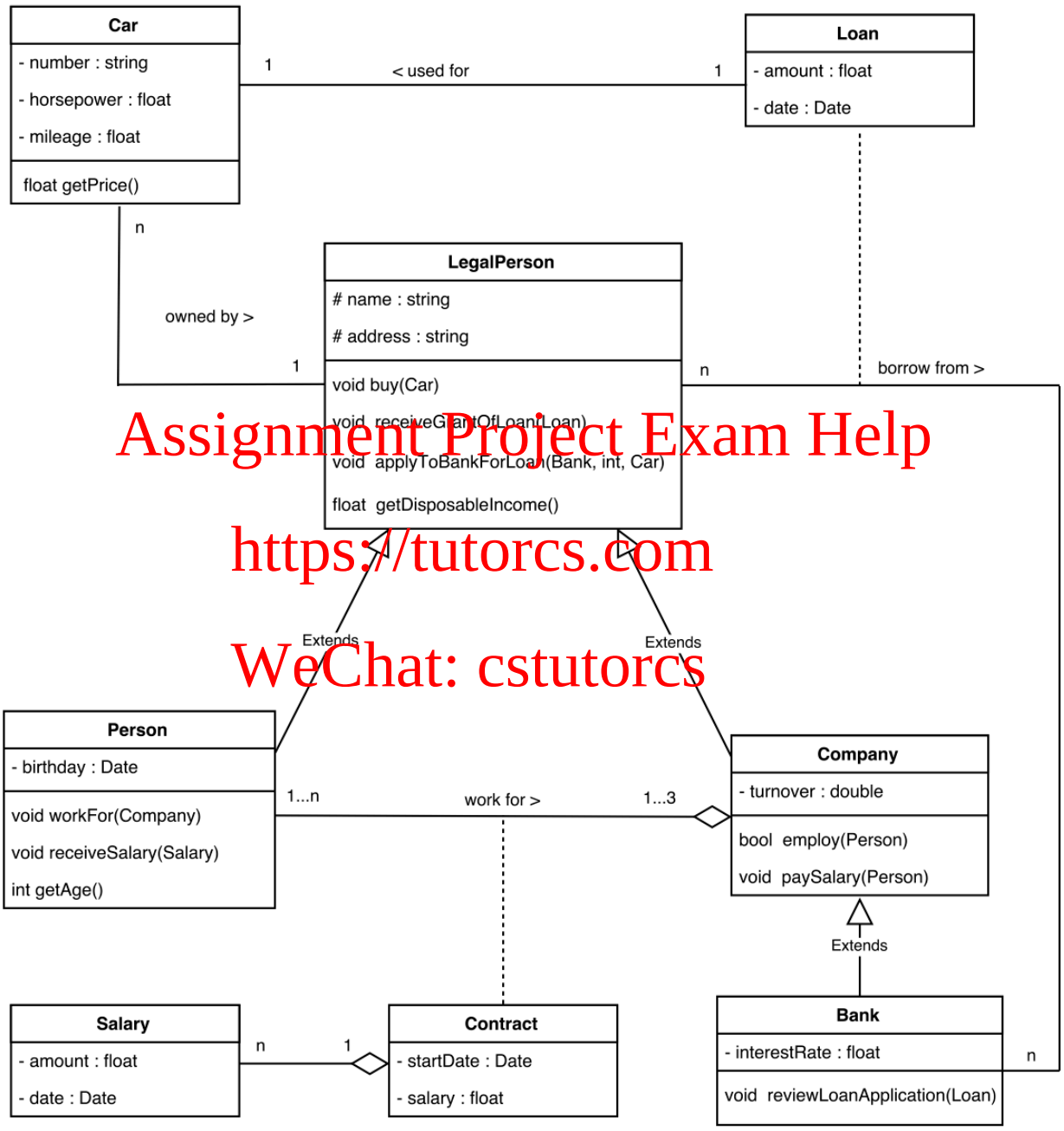
OODP – Car Loans

1 UML Class Diagram



2 Description

The inheritance hierarchy and association classes are specifally designed for this project. Therefore, this section will focused on the details and effects of them.

2.1 Inheritance Hierarchy

A. LegalPerson

The class called *LegalPerson* is designed as the superclass for the subclasses *Person* and *Company*. These two subclasses have several same attributes and behaviors:

- *Company* or *Person* may apply to a bank for a loan to buy a car,
- *Bank* grants the loan to a person and a company based on certain criteria,
- *Person* and *Company* are responsible to send a letter of acceptance when he/she/it receives the grant of a loan.

In this project, the only attribute needed in the *Person* class is birthday. For company, it only needs to contain its own unique attributes, such as, turnover. These subclasses can inherit or overwrite the same operations from the superclass.

B. Company

The inheritance relationship also exists between the classes *Bank* and *Company*. More specifically, a bank is a kind of company in the real world. *Bank* has the attributes turnover as *Company* does. *Bank* can also apply for a loan from other banks. Though the information is not specifically indicated on the problem descriptions. Summarily, *Bank* can inherit a few same attributes, such as, name, address, and possibly the above-mentioned behavior.

2.2 Association Classes

A. Contract

Person works for up to three companies. Therefore, association relationship is appropriate between these two classes and an association class called *Contract* is used in this project. Each contract contains the information related the relationship between person and company, such as start date of contract, salary. The use of class *Contract* can extend the functionality of this contract relationship, to include more detailed information for further considerations.

B. Salary

A *Salary* class is needed to document the information related to each salary payed. The use of *Salary* class can contain more detailed information related to salary, such as salary pay day, and amount. The relationship between *Salary* and *Contract* is aggregation.

C. Loan

A *Loan* class is created to denote the association relationship between *LegalPerson* (superclass of *Company* and *Person*) and *Bank*. *LegalPerson* applies to a bank for a loan to buy a car. The bank grants the loan to *LegalPerson* based on certain criteria. In this case, borrow date, amount, are needed in the class *Loan*. However, more attributes or operations can be added to this Loan class to describe more detailed information about this “borrow” behavior for further considerations.

To conclude, *LegalPerson* is the super class of *Company* and *Person*. *Company*, as the subclass of *LegalPerson*, is also the super class of *Bank*. An association class *Contract* is used to document the relationship between *Person* and *Company*. A *Salary* class is contained in

Contract to document the salary information about each contract. An association class *Loan* is created to denote the relationship between *LegalPerson* and *Bank*. Detailed design of the association relationships and operations of each class is elaborated in implementation section.

3 Implementation

3.1 A person may be working for one or more, but for no more than three companies.

bool Company::employ(Person),

where '*Person*' is the employee to be employed by the company, and the function returns true if the company employs the person successfully, otherwise returns false.The function generates a contract between the company and the person, which records start date, salary, status (whether the contract is expired or not) and etc. and will be stored as an attribute by both the person using function *workFor(Company)* and the company. True is returned in this case. The contract will not be generated if the person has already worked for three companies, in which case false is returned.

3.2 A person receives a salary from each company he/she works for.

void Company::payoff(Person),

where '*Person*' stands for the person to receive salary, and there is no return. According to the contract between the company and the person, the function generates an instance of *Salary* class which contains attributes, for example, amount, date, from (whom), to (whom) and etc. The instance is then stored in contract object, which can be accessed by the company and the person via contract.

3.3 The disposable income of person.

float Person::getDisposableIncome(void),

By querying all contracts the person has which records salaries, he/she is able to sum up all salaries received. To acquire the person's age, function *Person::getAge()* could be invoked. The sum of prices of all cars owned by the person is calculated by cars stored as an attribute inside *Person* object and *Car::getPrice()*, whereas remaining amount on existing loans by loans stored as an attribute inside the object.

3.4 The disposable income of company.

float Company::getDisposableIncome(void),

which is an implementation of function declaration in its superclass. All employees of a company can be acquired by the list of '*Contract*' stored in '*Company*', whereas remaining amount on existing loans by loans stored inside the object.

3.5 A car may be owned by a person, a company or a bank.

virtual void LegalPerson::buy(Car),

where '*Car*' is a car to be bought by someone, and there is no return. The function records the car as an attribute in a list in the object *LegalPerson*. It can be accessed by *LegalPerson* via cars collection data member.

3.6 Person or company applies to a bank for a loan

void LegalPerson::applyToBankForLoan(Bank, int, Car),

create a loan object to record the information of application and invoke the function of bank *bank.reviewLoanApplication(loan)* to review the application.

3.7 Bank reviews the application from company or person and makes a decision

void Bank::reviewLoanApplication(Loan),

- get the objects related with grants, borrower(*LegalPerson*) and purpose(*Car*), from passed loan object
- get the disposable income of borrower and price of car by calling corresponding operations, *getDisposableIncome()* and *getPrice()* respectively
- compare borrower's disposable income and car's price multiplied by interest rate of current bank
- if the disposable income is greater, grant the loan, add this loan object into bank's loans collection, and inform the result to borrower by function returning true (synchronous), which is not used in this case or invoking borrower's corresponding operation(asynchronous) accepted by this design.
- if not, refuse the loan and destruct the loan object

3.8 Person or company confirms the result of application from bank and if receiving grant, sends a confirmation letter back to bank

void LegalPerson::receiveGrantOfLoan(loan),

confirm the loan by adding this object to *LegalPerson*'s loans collection or database and inform the bank of receiving the result of grant by calling its corresponding function.