# Larger Task – Bank Loan

Think about a bank loan service. You can apply for a car loan, house loan and consumer loan.

After a loan is created, you can register the loan through loan proxy.

As a customer, you will also be registered through customer proxy.

You will also create a user interface to handle bank loan, which supports Scanner.

Task 1: Create several support classes and enums.

Create enum LoanType, which includes HouseLoan, ConsumerLoan, and CarLoan.

Create enum InsuranceType, which includes FullKasko, DeltKasko, ToppKasko.

Create CustomerInfo Class, feel free to include information. Make sure it contains at least String ssn.

Create LoanException Class that extends Exception,make a few constructors based on super().

Task 1 code is already done in the provided code basis.

Task 2: Create Abstract Loan class, which implements java.io.Serializable interface.

The Loan class has several attributes: double loanAmount, int numberofYears, LoanType loanType, double annualInterestRate, java.util.Date startDate, etc. One important attribute is CustomerInfo loanTaker. Make constructor and Getters/Setters.

Make a regular method called calculateLoanPayment().

Follow below formula and output necessary information.

double monthlyInterestRate=annualInterestRate/1200;  
double monthlyPayment = loanAmount \* monthlyInterestRate / (1 -  
 (Math.*pow*(1 / (1 + monthlyInterestRate), numberOfYears \* 12)));  
double totalPayment = monthlyPayment \* numberOfYears \* 12;

Make two abstract methods.

One is called getAnnualInterestRate(). It will be overridden by child classes since different loan classes will have different interest rate.

Another one is called printLoanInformation(). It will also be overridden by child classes since different loan classes will contain different loan information.

Task 3: Polymorphism. Create CarLoan, HouseLoan, and ConsumerLoan which extends Loan Class.

CarLoan will have three extra attributes: String carModel, String miles, InsuranceType typeOfInsurance.

Make constructor based on super() and corresponding Getters/Setters.

Implement getAnnualInterestRate, which sets annualInterestRate for car loan. Implement @Override printLoanInformation. This method will call calculateLoanPayment () defined in abstract Loan plus extra loan information related to CarLoan such as carModel, miles, typeOfInsurance, customer information, etc.

ConsumerLoan does not have extra attributes, or you can feel free to add any you think is relevant to consumer loan.

Make constructor based on super() and corresponding Getters/Setters.

Implement getAnnualInterestRate, which sets annualInterestRate for consumer loan. Implement @Override printLoanInformation, which call the regular method defined in abstract Loan class calculateLoanpayment(), and customer information.

HouseLoan is a bit different.

It has one extra attribute called coLoaner, since a house loan is usually shared with coLoaner. We will also include one attribute called String houseAddress.

Create Getters/setters.

When we create Constructors, think about Constructor overloading. Why is it? Because a houseLoan might only have one Loaner, or have both mainLoader and coLoaner. So we need to implement two constructors.

Implement getAnnualInterestRate, which sets annualInterestRate for consumer loan. Implement @Override printLoanInformation, which call the regular method defined in abstract Loan class calculateLoanpayment(). You need to check if the houseLoan has only one Loaner or two Loaners, and print out corresponding customer information.

Task 4: Create CustomerProxy

Each loan is related to customers. So we need to create a CustomerProxy. The CustomerProxy will help you register customers, fetch customers based on ssn, and validate ssn.

First, we create Interface called ICustomerProxy.

CustomerInfo getCustomer(String ssn);  
boolean isValidSsn(String personalIdentifyNumber) throws Exception;  
void registerCustomer(CustomerInfo customerInfo) throws Exception;

Then, create CustomerProxy class which implements ICustomerProxy.

We need to create a HashMap to contain customer information. Each customer has an unique ssn, so ssn is naturally the key in key/value pair. CustomerInfo will be the value in key/value pair.

registerCustomer is to put a new key/value pair into HashMap.

getCustomer is to fetch key/value pair based on key.

isValidSsn is a bit tricky, feel free to implement it, for example, check if it is 11 digits long…

Note that when we register customer, if the ssn is not valid, we throw an exception. When we get customer, it can return null.

Task 5: Create LoanProxy

We need to create a LoanProxy since we need to register Loan for customers. The LoanProxy will help you registerLoan, fetchLoan, etc.

First, we create Interface called ILoanProxy.

Loan getLoan(Integer loanId);  
void registerLoan(Loan loan) throws Exception;  
void printAllLoans() throws Exception;  
void printLoanbySsn(String ssn) throws Exception;

The, create LoanProxy class which implemented ILoanProxy interface.

We need to create a HashMap to contain loan information. Can we use ssn as key? No. because HashMap must have unique key, while each customer can have multiple loans. So, what shall we choose? We can create a loanId each time when we create a loan, make loanId as static, so that it is shared by instances and incremented by one each time when we create a loan instance.

So the key/value pair of loanHashMap will be Integer loanId, Loan loan.

The same as CustomerProxy:

registerLoan is to put a new key/value pair into HashMap.

getLoan is to fetch key/value pair based on key.

We can simply print some messages for printAllLoans and printLoanBySsn.

Task 6: We will make a LoanFactory to create loan

In the Loanfactory class, we will create a function called CreateLoan(). It will create different types of loan based on the specified LoanType. Hint: think about switch statement.

The main method of LoanFactory class will be a sequence of operations which are:

Create customers by using new CustomerInfo()

Register customers by using CustomerProxy.

Create Loans by using CreateLoan ().

Register loans by using LoanProxy.

Try to use upcasting when we create different loans since we will put them all into HashMap while using LoanProxy. If we need to access Child class attributes,use downcasting.

Note that CarLoan and ConsumerLoan only allows one loan taker. So we must throw exception if we figure out there are more than one loan taker when creating a carLoan or consumerLoan.

Task 7: We are almost there! However…

Think about we created a customer while forgot to register it to CustomerProxy. We sort of lose synchronization between CustomerProxy and LoanProxy, right?

When we create a loan, during the constructor of Loan class, we can check if the customer is already registered in CustomerProxy. If not, try to register the new customer.

Here we meet two challenges:

First, We need to pass CustomerProxy as an argument to Loan class so that we can use CustomerProxy methods. Think about Aggregation we have learned.

public CarLoan(int numberOfYears, double loanAmount, LoanType loanType, CustomerInfo customer,  
 CustomerProxy customerProxy)

Secondly, can we implement registering new customer in abstract Loan class?

The answer is that we shall implement registering new customer in Child Loan classes. Because for HouseLoan, you might need to register two customers, if the loan applies to both mainLoanTaker and coLoanTaker.

Task 8: Extra task to print out all loans or loans based on ssn

This is a tough task. Remember that we have defined printAllLoans() and printLoanBySsn(String ssn) in the LoanProxy class. How shall we implement them?

For printAllLoans() recall how we iterate HashMap keysets?

printLoanBySsn(String ssn) is a bit tricky. You need to check if loan customer ssn equals to the input argument ssn. You can try to use Lambda function. For example

hashmap,entrySet().stream().map().filter().collect(Collectors.toList()).

For example

Map<Integer, Loan> newLoanHashMap = loanHashMap.entrySet()  
 .stream()  
 .filter(entry-> entry.getValue() instanceof HouseLoan)  
 .map(loan -> (HouseLoan)loan)  
 .filter(loan-> loan.customer.getSsn().equals(ssn) || loan.getCoLoanTaker().getSsn().equals(ssn))  
 .collect(Collectors.toMap(loan -> loan.getLoanId(), loan -> loan));

However, we met a problem for HouseLoan. We want to print out ssn also for coLoaners!

Can’t we just use Loan.getCoLoanTaker().getSsn()? No! Since loanHashMap is <Integer, Loan>, and remember upcasting? Loan doesn’t have access to coLoanTaker since it belongs to HouseLoan..

Think about it, try to use instanceof and try downcasting!

Task 9: Make a user interface so that you can register customers, register loans, search for loans based on ssn, or print out all loans. Use Scanner.

*\* The new user interface shall be able to handle:  
\* 1. create a new customer  
\* 2. register a customer after created  
\* 3. create a new loan  
\* 4. you will be able to choose the type: CarLoan, ConsumerLoan, HouseLoan  
\* 5. register the loan after created  
\* 6. print all loans  
\* 7. print loan based on SSN*

Good luck!