C++ fundamentals with use cases from finance

Tutorial 3: Counterparty Credit Risk management, Backtesting set up

Ivan Zhdankin

Business Context: Counterparty Credit Risk management, Backtesting set up

- Counterparty Credit Risk - risk of the counterparty default and consequent loss in our portfolio
- For this tutorial we assume that we have several loans in our portfolio with different counterparties
- Loan is represented by:
  - Notional
  - APR (annual percentage rate)
  - Rank - probability of counterparty default ($\in [0, 1]$)
- In case of the counterparty default there is loss in our portfolio - exposure at default (EAD):

$$EAD = Notional$$

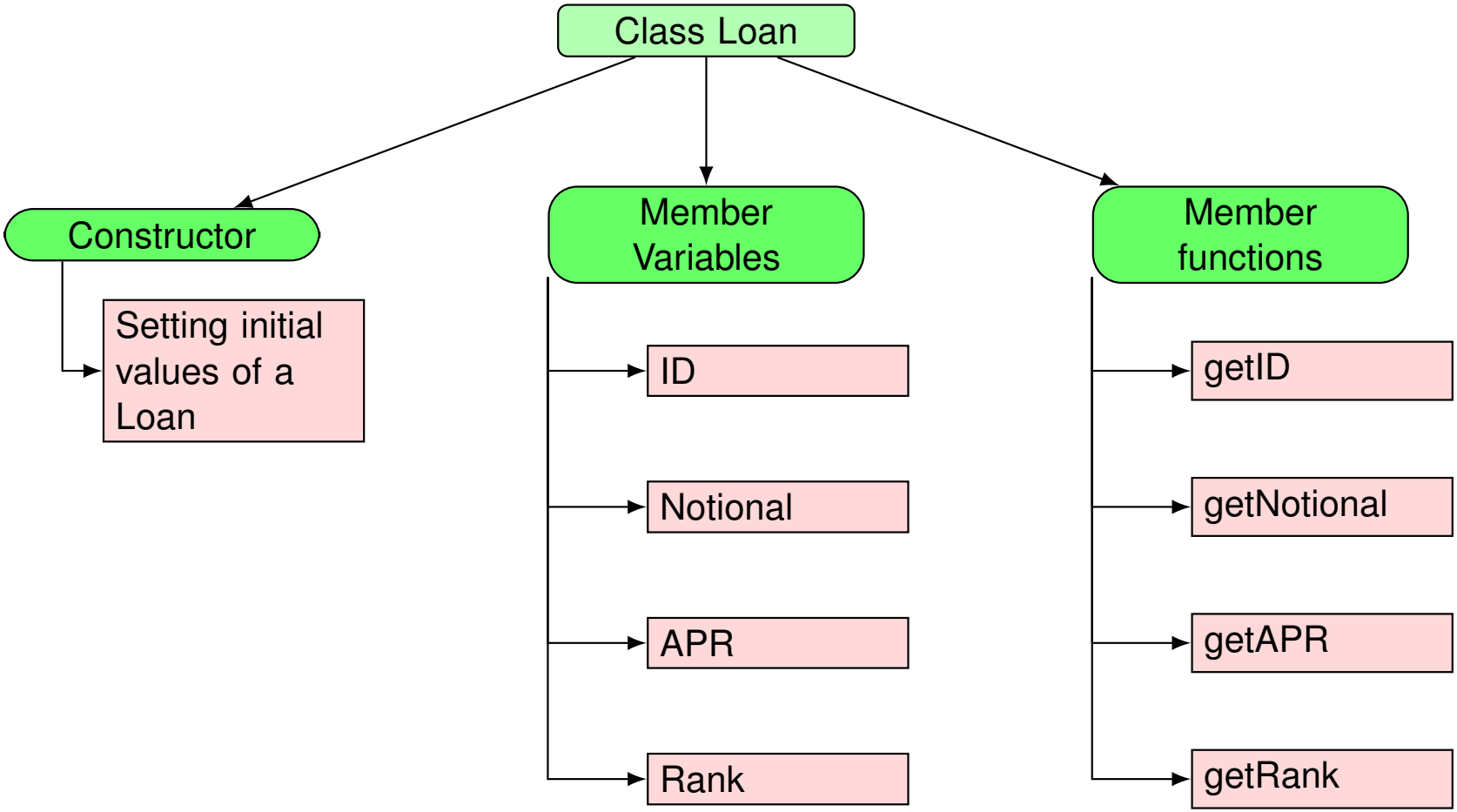
- Thus the Expected Loss is equal to:

$$EL = Notional * Rank$$

Which is loss weighted by the probability of loss

Part 1/4

- Define a Class Loan
- The class should contain private member variables: ID, Notional, APR (annual percentage rate), rank (probability of the counterpart default)
- The class should contain public member functions, use keyword *const* where possible:
  - GetNotional, GetAPR, GetRank, GetID - take no input parameters and return the corresponding values of the variables
- Define operator overloading as a free function to compare two objects of a class Loan (one loan is smaller than another if it is riskier than another, that is: $Notional_1 * rank_1 > Notional_2 * rank_2$)



Part 2/4

- Define operator overloading as a free function to merge two objects of a class Loan into one Loan object
- The merged load should have Notional equal to the sum of the individual Notionals:

$$Notional = Notional_1 + Notional_2$$

- The merged loan should have rank and APR which are defined as weighted average of the individual ranks and APRs:

$$rank = \frac{Notional_1 * rank_1 + Notional_2 * rank_2}{Notional_1 + Notional_2}$$

$$APR = \frac{Notional_1 * APR_1 + Notional_2 * APR_2}{Notional_1 + Notional_2}$$

Part 3/4

- Modify the class Loan to be template class that can have different types of the ID (int, double, string)
- Create different instances of the class to validate your code
- Create template free function which takes the 4 variables and return the weighted average of them (this function would help to calculate weighted Rank and APR)
- In main function create instance of a class and then define a reference and a pointer to it
- Print out on the screen the ID of the object using the reference and using the pointer
- Using the keyword ***this*** define the member function of the class to return the combined ID: the function should take only one object of a class and return combined IDs
- Let us define a ***friend*** function that is able to change the ID of any Loan

Part 4/4

- Define a class Backtesting
- The class should have private variable name
- The class should have public member functions:
  - Constructor - takes name of the backtest as input parameter, assign it to the member variable name and print out on the screen ”test has started”
  - Destructor - print out on the screen ”test has finished”
  - GetName - takes no parameters and return the name of a backtest
- In main function instantiate an object of a class in a separate scope defined by the curve brackets
- Instantiate an object of the class and allocate memory on the heap using operator ***new***
- Using operator ***delete*** remove the created object from the heap