**Instructions:**

1. Create new maven project with the following details:

group-id: com.java.training

artifact-id: coding-activity-02

package name: com.java.training.codingactivity02

1. Create each class based on the UML diagram

Diagram, schematic

Description automatically generated

1. Class Definitions
   1. Payment – a domain class that will hold the details of the loan payment. Create setters and getters for each field.
      1. amountPaid – payment amount
      2. modesOfPayment – the payment mode which can either be “Cash”, “Check”, “GCash” or any other specific payment method
      3. loanType – the type of loan whether Salary or Calamity loan
      4. Payment(double amountPaid, String modesOfPayment, int loanType) – constructor to set Payment details
   2. Customer – a domain class that will hold the customer details. Create setters and getters for each field.
      1. firstName – customer first name
      2. lastName – customer last name
      3. amountLoan – the amount loaned by the customer
      4. terms – payment term in months (*i.e. 12, 24, 36*)
      5. monthsOfContribution – number of months the customer has been contributing to CPF
      6. displayDetails() – method to display the customer details such as **Customer Name, Loan Amount, Loan Terms** and **Months of Contibution**.
   3. Loan – an abstract class that will contain the Payment transactions and the details of the customer who applied for loan. Create setters and getters for each field.
      1. customer – holds the customer details.
      2. payments – loan payment transactions made by the customer
      3. Loan(Customer customer) – constructor to set the Customer object.
      4. addPayment(Payment payment) – adds payment to payment list.
      5. getTotalPayment() – sums up all the payment transactions made

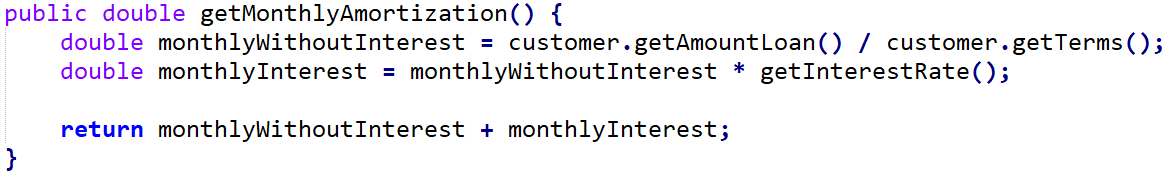
**Hint:** Iterate over your payment list and sum up all the amounts paid

* + 1. *getInterestRate()* – an abstract method that returns the specific interest rate depending on the type of loan.
    2. getTotalInterest() – returns the total interest for the specified loan amount.

**Hint:** Customer’s loaned amount multiplied by the interest rate returned by getInterestRate() method.

* + 1. getMonthlyAmortization() –returns the monthly amortization of the loan amount.

Following calculations may be utilized:

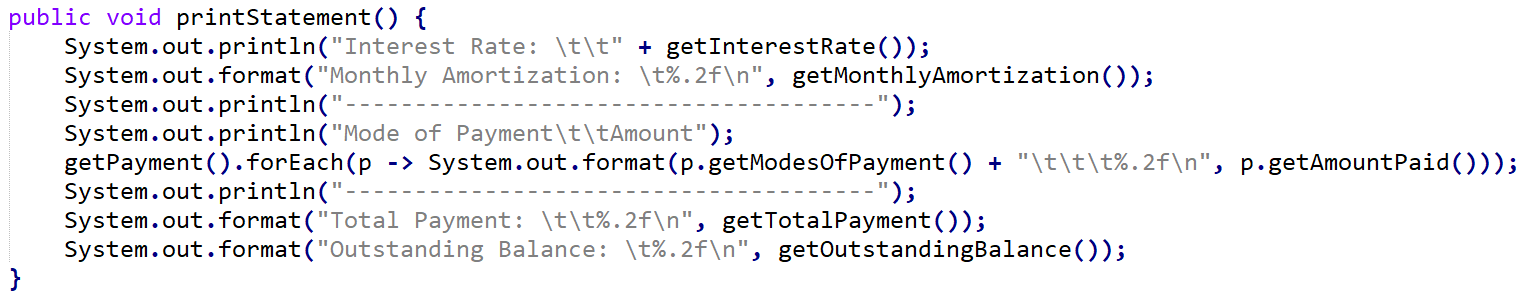


* + 1. getOutstandingBalance() – returns the remaining loan balance.

**Hint:** Deduct the total payments made (*getTotalPayment()* method) from the sum of the Customers loaned amount (*customer.getAmountLoan()*) and the total interest (*getTotalInterest()*)

* + 1. printStatement() – prints the loan details such as **Interest Rate, Monthly Amortization, payments made, Total Payment** and **Outstanding Balance**.

Following code may be utilized:



* 1. SalaryLoan – a concrete class for salary loan type
     1. SalaryLoan(Customer customer) – sets the customer. This will call the superclass’ constructor by passing the customer object.
     2. getInterestRate() – returns the decimal equivalent of interest rate for salary loan type.

Salary Loan Interest rate is 10%

* 1. CalamityLoan – a concrete class for calamity loan type
     1. CalamityLoan(Customer customer) – sets the customer. This will call the superclass’ constructor by passing the customer object.
     2. getInterestRate() – returns the decimal equivalent of interest rate for calamity loan type.

Calamity Loan Interest rate is 5%

* 1. InvalidLoanException – custom exception class to handle exceptions for invalid loan if contributed months is less than 36. Contains a constructor that receives one parameter **String message**.
  2. LoanUtility – utility class containing the validation of contributed months.
     1. validateLoan() – method to validate the customer’s number of contribution months. If customer contribution months is less than 36, method throws custom InvalidLoanException.

1. CodingActivity02 Class:
   1. Instantiate Customer object and set the following details.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **First Name** | **Last Name** | **Loan Amount** | **Terms** | **Contribution Months** |
| Jane | Doe | 30000.00 | 36 | 40 |

* 1. Display customer details.
  2. Validate Loan contribution by calling validateLoan() method.
  3. Instantiate SalaryLoan object.
  4. Create three Payments for Salary Loan

|  |  |  |
| --- | --- | --- |
| **Amount** | **Mode of Payment** | **Loan Type No** |
| 916.67 | Cash | 1 |
| 920.00 | Bank | 1 |
| 930.00 | Check | 1 |

* 1. Add payments to salary loan.
  2. Print the statement.
  3. Instantiate CalamityLoan object.
  4. Create three Payments for Calamity Loan

|  |  |  |
| --- | --- | --- |
| **Amount** | **Mode of Payment** | **Loan Type No** |
| 875.00 | Cash | 2 |
| 875.00 | Bank | 2 |
| 880.00 | Check | 2 |

* 1. Add payments to calamity loan.
  2. Print the statement.
  3. Instantiate another Customer object and set the following details.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **First Name** | **Last Name** | **Loan Amount** | **Terms** | **Contribution Months** |
| Stacey | Doe | 15000.00 | 24 | 12 |

* 1. Display customer details.
  2. Validate Loan contribution by calling validateLoan() method.

1. Sample Output:

