```
CollectionAgency.java
*****************************************************************************************
********************
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class CollectionAgency {
    /**
     * This method should take the file path as argument
     * and it should parse the data stored in the file and
     * it should validate the policy Id by invoking the validate(String policyId)
method,
     * if valid, construct a Payment object for each record in the file,
     * and then calculate the payment amount by invoking the calculatePaymentAmount
method of Payment class.
     * After calculating the payment amount,
     * each Payment should be added to the list and this method should return the
list of Payment.
     * @param filePath Path include the name where the file is located
     * @return List of Payment after reading data from the file
     * @see Payment
     */
    public List<Payment> generatePaymentAmount(String filePath) {
        List<Payment> paymentList = new ArrayList<>();

        try {
            // Creating scanner object for reading data from the file
            Scanner scanner = new Scanner(new BufferedReader(new
FileReader(filePath)));

            while (scanner.hasNext()) {
                String[] values = scanner.nextLine().split(",");
                String policyId = values[0];
                double monthlyPremium = Double.parseDouble(values[1]);
                int noOfMonth = Integer.parseInt(values[2]);
                try {
                    // Validating policyId
                    if (validate(policyId)) {
                        Payment payment = new Payment();
                        payment.setPolicyId(policyId);
                        payment.setMonthlyPremium(monthlyPremium);
                        payment.setNoOfMonths(noOfMonth);
                        payment.calculatePaymentAmount();

                        // Adding new Payment to the paymentList
                        paymentList.add(payment);
                    }
                } catch (InvalidPolicyIdException e) {
                    // Printing error message if the policy id is invalid
                    System.out.println(e.getMessage());
                }
            }

            scanner.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

        return paymentList;
    }


    /**
     * This method should validate the policyId,
     * if valid return true else this method should throw an user-defined exception
     * and adding it to the list.
     * The policyId should be in the following format:
     * 1.The policyId should contain exactly 10 characters
     * 2.The fifth character must be an alphabet “H” in upper-case only.
     * If the policyId is valid then parse the data and calculate the payment amount
     * else throw a user defined Exception “InvalidPolicyIdException” with a message
"Invalid Policy Id".
     * @param policyId Policy Id of a customer
     * @return true if the policyId qualify the specification given
     * @throws InvalidPolicyIdException when policyId does not match the
specification
     */
    public boolean validate(String policyId) throws InvalidPolicyIdException {
        Pattern pattern = Pattern.compile("^\\w{4}H\\w{5}$");
        Matcher matcher = pattern.matcher(policyId);

        if (matcher.matches()) {
            return true;
        } else {
            throw new InvalidPolicyIdException("Invalid Policy Id");
        }
    }


    /**
     * This method should update the balance_premium by reducing the existing value
with the calculated payment amount in the Policy_Detailstable.
     * Assume that the balance_premium will be greater than or equal to calculated
payment amount.
     * @param paymentList List of Payment
     * @see Payment
     */
    public void updatePolicyDetails(List<Payment> paymentList) {
        Connection connection = new DBHandler().establishConnection();

        for (Payment payment : paymentList) {
            try {
                // Getting current balance premium
                PreparedStatement preparedStatement1 =
connection.prepareStatement("select balance_premium from Policy_Details where
policy_id = ?;");
                preparedStatement1.setString(1, payment.getPolicyId());
                ResultSet resultSet = preparedStatement1.executeQuery();

                resultSet.next();
                double currentBalance = resultSet.getDouble(1);
                double updatedBalance = currentBalance - payment.getPaymentAmount();

                // Updating the balance premium with the new value
                PreparedStatement preparedStatement2 =
connection.prepareStatement("update Policy_Details set balance_premium = ? where
policy_id = ?;");
                preparedStatement2.setDouble(1, updatedBalance);
                preparedStatement2.setString(2, payment.getPolicyId());

                preparedStatement2.executeUpdate();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
DBHandler.java
******************************************************************************

import java.io.FileInputStream;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;

public class DBHandler {
    /**
     * This method should connect to the database by reading the database details
from the db.properties file and it should return the connection object
     * @return Connection to the MySQL database or null when there is some problem
connecting to the database
     * @see Connection
     */
    public Connection establishConnection() {
        Properties properties = new Properties();

        try {
            // Creating input stream from db.properties file
            FileInputStream fileInputStream = new FileInputStream("db.properties");
            properties.load(fileInputStream);

            // Getting value of the properties file
            String driver = properties.getProperty("db.classname");
            String url = properties.getProperty("db.url");
            String username = properties.getProperty("db.username");
            String password = properties.getProperty("db.password");

            // Making sure drive jar is available
            Class.forName(driver);

            // Returning a new database connection
            return DriverManager.getConnection(
                    url,
                    username,
                    password
            );
        } catch (IOException | ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }

        return null;
```

{1730519521478786166}
```
    }
}
```

InvalidPolicyIdException.java

****
*

```
public class InvalidPolicyIdException extends Exception {
    /**
     * Custom exception for invalid policy id
     * @param message Message passed to be thrown when the invalid policy id is
detected
     */
    public InvalidPolicyIdException(String message) {
        super(message);
    }
}
```

Main.java

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;

public class Main {
    private static void printDatabase() {
        Connection connection = new DBHandler().establishConnection();

        try {
            ResultSet resultSet = connection.createStatement().executeQuery("select
* from Policy_Details;");

            while (resultSet.next()) {
                String policyId = resultSet.getString(1);
                double totalCoverage = resultSet.getDouble(2);
                double balancePremium = resultSet.getDouble(3);
                int premiumDurationYears = resultSet.getInt(4);
```

{1730519521478786166}

```
                System.out.println(String.format("%-20s%-20s%-20s%-20s",
"policy_id", "total_coverage", "balance_premium", "premium_duration_year_int"));
                System.out.println(String.format("%-20s%-20.2f%-20.2f%-20d",
policyId, totalCoverage, balancePremium, premiumDurationYears));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        CollectionAgency collectionAgency = new CollectionAgency();

        System.out.println("Payments Retrieved from the text file...");
        List<Payment> paymentList =
collectionAgency.generatePaymentAmount("PolicyPaymentDetails.txt");
        paymentList.forEach(System.out::println);

        System.out.println("Database before updating...");
        printDatabase();

        System.out.println("Database after updating...");
        collectionAgency.updatePolicyDetails(paymentList);
        printDatabase();
    }
}
```

Payment.java

```
public class Payment {
    private String policyId;
    private double monthlyPremium;
    private int noOfMonths;
    private double paymentAmount;

    public String getPolicyId() {
        return policyId;
    }

    public void setPolicyId(String policyId) {
        this.policyId = policyId;
    }
    public double getMonthlyPremium() {
        return monthlyPremium;
    }

    public void setMonthlyPremium(double monthlyPremium) {
        this.monthlyPremium = monthlyPremium;
    }

    public int getNoOfMonths() {
        return noOfMonths;
    }

    public void setNoOfMonths(int noOfMonths) {
        this.noOfMonths = noOfMonths;
    }

    public double getPaymentAmount() {
        return paymentAmount;
    }

    public void setPaymentAmount(double paymentAmount) {
        this.paymentAmount = paymentAmount;
    }

    /**
     * This method should calculate and set the payment amount based on the monthly
Premium and
     * no of Months for each payment.
     *
     * No Of Months     Penalty Percentage on the paymentAmount
     * 1                        0% (No penalty)
     * >1 and <=5               3%
     * >5 and <=12              5%
     * >12                      7%
     *
     * For example: If a payment has a monthly premium of Rs. 5000 and the number of
months as 4, then the payment amount will be (5000*4) which is 20000.00. Since the
number of months is 4, the penalty percentage will be 3%.
     * Therefore, the penalty will be (20000.0*(3/100)) which is Rs. 600.00.
Therefore, the payment amount for this payment will be((5000*4)-600.0) which is
Rs.19400.00.
     * After calculating the payment amount for each payment, store the payment
object into a list.
     */

    public void calculatePaymentAmount() {
        paymentAmount = monthlyPremium * (double) noOfMonths;
        double percentage = 0.0;
        if (noOfMonths > 1 && noOfMonths <= 5) {
            percentage = 3;
        } else if (noOfMonths > 5 && noOfMonths <= 12) {
            percentage = 5;
        } else if (noOfMonths > 12) {
            percentage = 7;
        }

        double penalty = paymentAmount * percentage / 100.0;
        paymentAmount -= penalty;
    }

    @Override
    public String toString() {
        return "Payment{" +
                "policyId='" + policyId + '\'' +
                ", monthlyPremium=" + monthlyPremium +
                ", noOfMonths=" + noOfMonths +
                ", paymentAmount=" + paymentAmount +
                '}';
    }
}
```

----
db properties

```
db.classname=com.mysql.jdbc.Driver
db.url=jdbc:mysql://localhost:3306/testbase
db.username=ritam
db.password=password
```

----

script file

```
drop database if exists Insurance;

create database Insurance;

use Insurance;
```

```
create table Policy_Details
(
    policy_id              varchar(25) primary key,
    total_coverage         double(10, 2),
    balance_premium        double(10, 2),
    premium_duration_years int
);

insert into Policy_Details
values ('2005H37012', 100000, 100000, 15);
insert into Policy_Details
values ('2006H37013', 100000, 85000, 20);
insert into Policy_Details
values ('2007H37014', 150000, 150000, 25);
insert into Policy_Details
values ('2008H37015', 250000, 150000, 10);
insert into Policy_Details
values ('2009H37016', 800000, 75000, 30);

select *
from Policy_Details;

truncate Policy_Details;

COMMIT;
```