Coursework 2 (25%)

Submission Deadline: 25 March 2024, 23:59

1 Submission Details

- **Format:** Submit all your source code files (.java) in a single compressed (zipped) folder.
- **Documentation:** Ensure your code is well-documented with comments to support understanding and facilitate assessment.
- **Deadline:** Submissions are due by 25 March 2024, 23:59.
- **Penalties:** Late submissions will incur a 5% deduction per day, up to 24 hours after the deadline. Submissions made later than 24 hours post-deadline will not be accepted unless mitigating circumstances are provided.

2 Scenario - Loans Management System:

In this task you consider the similar scenario as in CW1. A `XYZBank` bank has credit records for all its customers. Each customer can have several loans if they are eligible.

Eligibility criterion: "Total amount (total amount left to pay) should not be greater than 4 times the customer's annual income".

You need to develop a code that allows to:

- Register a new customer(s), including their **Customer ID** (in the format `'AAAXXX'`, where `A` denotes a capital letter `A,...,Z`, and `X` stays for a digit `0,...,9`) and their **annual income** .
- Register a credit record for any existing customer (given their Customer ID), if they satisfy the Eligibility criteria (see above). The following information should be entered:
  - Record ID, in the format `'XXXXXX'`, where `X` stays for a digit `0,...,9;`
    Note: That Record ID should be unique across all records in the bank.
  - A type of the loan: "`Auto`", "`Builder`","`Mortgage`", "`Personal`", or "`Other`";
  - Interest rate - a number;

  - Amount left to pay, in thousands pounds, - a number, for example, 360 for £360,000;
  - The loan term left, in years - a number.

- Have an interface to print information about all customer's loans, if the ID is given, or about all loans of all customers.

3 Task Description:

Your code should include the following components:

1. An abstract class, named `Loan`, which has attributes:
   - `Record ID`,
   - `Loan type`,
   - `Interest rate`;
   - `Loan term left`.
2. Five sub-classes, corresponding to the types of loans (see the information above)
   Note:
   - All classes should allow to create instances with a default or a particular (custom) set of properties values.
   - The classes should also provide methods for updating the information, i.e. writing/reading/printing the values.
   - "Builder" and "Mortgage" loans should have an *overpayment* option - a number, representing a percentage between 0 and 2.
3. An interface, named `CheckerPrinter`, which provides methods for:
   - Checking a customer's eligibility;
   - Printing customer's details.
4. A class `Customer`, extending the interface (CheckerPrinter) with the attributes:
   - `Customer ID` (see the format above);
   - `Customer income`;
   - `Eligibility status`;
   - `Credit records` (an array of the instances of the concrete classes).

The class should provide the methods to fill in and read necessary information, and to print the customer details (see the example of output below).
The class should provide options to create an instance of the class either with default parameters or with some set of parameters, in both cases credit records can be empty.

5. Main class, called `XYZBank`, should allow:

   - To register a new customer;
   - Update information about existing customers: update income, status, add a new loan record, remove an old record.
   - Eligibility of a customer should be verified before adding a new loan or updating existing loan.
     A user should be notified if the operation is not valid.
   - A user should be able to print information about a particular user, given a Customer ID, as well as to print information about all customers.

4 Example of the printed results:

Use the appropriate methods to produce an output similar to one of these examples:

Maximum number of Records: 10
Registered records: 5
==========
CustomerID AAA001

Eligible to arrange new loans - NO

RecordID	LoanType	IntRate	AmountLeft	TimeLeft
000001	Auto	13.50	50	5
000102	Mortgage	6.95	250	18
000223	Other	25.00	23	3

==========
CustomerID ABC001

Eligible to arrange new loans - YES

RecordID	LoanType	IntRate	AmountLeft	TimeLeft
000005	Personal	9.50	20	3
000302	Mortgage	6.95	157	15

5 Marking Criteria

Criteria	0-39%	40-49%	50-69%	70-79%	80-100%
Object-Oriented Design (30)	Inadequate implementation of classes and inheritance, with key OOP principles ignored.	Basic implementation of the Loan and Customer classes but lacking effective inheritance and polymorphism.	Proper use of classes, inheritance, and some polymorphism, but may lack interface implementation.	Effective use of inheritance, polymorphism, and interfaces with minor improvements possible.	Excellent application of OOP principles, including well-designed abstract classes, subclasses, and interfaces.
Implementation of Functional Requirements (30)	Critical functionalities such as customer registration or loan management are missing or flawed.	Basic functionalities are present but contain significant errors; eligibility checks may be incorrect or missing.	Functional requirements are mostly met, including customer registration, loan management, and eligibility checks, with minor issues.	All specified functionalities including complex eligibility checks are correctly implemented with minor omissions.	All functionalities are flawlessly implemented, with advanced features enhancing the loans management system.
Code Quality (15)	Poorly documented code, making it difficult to understand the logic and structure.	Some documentation provided, but it does little to clarify complex sections of code.	Good documentation of methods and logic, aiding in understanding of the basic flow and functionalities.	Very well-documented code, with clear explanations of complex algorithms and system design.	Exceptionally clear and thorough documentation throughout, facilitating easy maintenance and understanding of advanced concepts.
User Input and Error Handling (15)	Neglects basic user input validation, leading to frequent crashes or errors.	Limited attempt at input validation and error handling; system is prone to errors with unusual inputs.	Good input validation and error handling for common scenarios, with occasional oversight in edge cases.	Strong validation and error handling, ensuring system resilience against incorrect inputs and providing clear error messages.	Comprehensive input validation and sophisticated error handling, including detailed user feedback for error correction.
Output Format and Accuracy (10)	Outputs are frequently incorrect or incoherent, with significant formatting issues.	Outputs generally follow the required format but with noticeable inaccuracies or inconsistencies.	Output is largely accurate and meets most formatting requirements, with minor mistakes.	Outputs are accurate, well-formatted, and effectively convey the required information.	Exceptionally well-formatted and accurate outputs, enhancing readability and user experience.