COURSEWORK 1(25%)
Due date – 22nd February, 23:59

1 Submission files:

A zipped file contains all of your *.java* files. Be sure that your code is well documented. To get a higher mark, your code should be supported by comments.

2 Penalties:

- 5% penalty will be applied to any work submitted late (i. e. the maximal mark is 20%).
- No submissions made later than 24 hours will be accepted, unless you have applied for Mitigating Circumstances.

3 Scenario - Loans Management System:

You need to develop a code to register a credit record for a ***XYZBank*** customer. For each application, the following information should be entered:

- Record ID, in the format *'XXXXXX'*, where *X* stays for a digit 0,...,9;
- Customer ID, in the format *'AAAXXX'*, where *A* denotes a capital letter *A,..., Z*, and *X* stays for a digit *0,...,9*;
- 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.

4 Task Description:

1. Create a public class named **Record**:
   - Each instance of the class represents one loan record.
   - The class' private properties should correspond to the characteristics described in Section 3.
   - It should be possible to create instances with a default or a particular (custom) set of properties values.

- The class should provide methods for updating the information, i.e. writing/reading the values.

2. Create a main class, called ***XYZBank***. The main method

   - Should create an array of records to store the relevant information (maximum number of records should be defined by the user).
   - Each record should be implemented as an instance of **Record** class.
   - The information about each loan should be entered by the user (one customer can have several loans, each should correspond to a different record).
   - The code should provide an option to create and fill a new record up to the maximum number.
   - When all (necessary) records are registered, they should be printed in a "tabular" form (see below examples of the output).

5 Example of the printed results:

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

Version 1:

```
Maximum number of Records: 3
Registered records: 3

RecordID     CustomerID   LoanType    IntRate     AmountLeft   TimeLeft
000001       AAA001       Auto        13.50       26           10
000102       BBB002       Mortgage    6.95        157          18
000223       ABC005       Other       25.00       17           3
```

Version 2:

```
Maximum number of records: 7
Registered records: 3

RecordID  CustomerID  LoanType  IntRate  AmountLeft  TimeLeft
  000001      AAA001      Auto    13.50          26        10
  000102      BBB002  Mortgage     6.95         157        18
  000223      ABC005     Other    25.00          17         3
```

6 Marking Criteria

Criteria	0-39%	40-49%	50-69%	70-79%	80-100%
Object-Oriented Design (30)	Classes and methods not properly utilized or missing.	Basic class structure implemented but with significant design flaws.	Correct use of classes and methods but lacks advanced features	Effective use of OOP principles. Advanced class structures well-implemented.	Innovative design patterns, exceptional use of OOP concepts, and flawless integration of advanced features.
Implementation of Functional Requirements (30)	Incomplete implementation, major functionalities missing.	Essential functionalities present but with errors. Limited implementation of requirements.	Most functionalities implemented correctly. Some advanced features may be missing or partially implemented.	Full implementation of all required functionalities. Some advanced features successfully implemented.	Comprehensive and flawless implementation, exceeding specified requirements. Creative solutions and additional functionalities added.
Code Quality (15)	No or irrelevant comments, making the code difficult to understand.	Minimal comments present, offering little clarity or insight into the code's functionality.	Comments are used to explain the purpose of classes, methods, and significant blocks of code.	Detailed comments that enhance understanding, including explanations of complex logic and decisions.	Exceptional use of comments throughout the code, providing clear, concise, and valuable insights into code functionality and design choices.
User Input and Error Handling (15)	No input validation, program crashes with invalid input.	Basic input validation implemented but error handling is minimal.	Adequate input validation and error handling, covering common input errors.	Robust input validation and error handling, ensuring program stability under erroneous conditions.	Advanced input validation and error handling, including informative feedback to the user on errors.
Output Format and Accuracy (10)	Output is incorrect or not presented.	Output contains errors or formatting issues. Basic requirements met with inaccuracies.	Output is mostly accurate and meets specified format requirements. Minor errors may be present.	Accurate output that adheres to all specified formatting requirements. Clear and well-presented.	Exceptionally formatted output, accuracy in all scenarios tested. Goes above and beyond in clarity and presentation quality.