Continuous Practical Assignment II

Programming Methodology (CSF101)

Bachelors of Engineering in Electrical Engineering (EE), Mechanical Engineering (ME), Instrumentation and Control Engineering (ICE) & Electronics and Communication Engineering (ECE)

Objective:

1. **Application of Python Programming:** To demonstrate proficiency and understanding in Python programming, including knowledge of data types, control structures, functions, libraries/modules to create an OOP based application using python.
2. **Self-Learning:** To demonstrate self-learning capabilities after understanding the fundamentals of Python programming.
3. **Implement OOP principles:** Create an OOP based application using OOP principles including classes, objects, inheritance, encapsulation, polymorphism, and abstraction.

Instructions:

Create a terminal based functioning banking application where:

- You can open a Bank Account and the application will give you account number and a default password. Save the information like account number, password, bank type and then balance amount in a txt file called "accounts.txt".
- Implement login functionality for created accounts by reading the information from the file and checking balance.
- Make the Account Class have properties like accountNumber, balance, accountType, savings, current, etc.
- Implement functionality for depositing, withdrawing, and deleting personal accounts after logging in.
- Implement sending money to other accounts and implement error handling like insufficient funds or if the receiving account exists or not.
- Implement functionality for Types of Bank Account like BusinessAccounts & PersonalAccounts.
- NOTE: Use OOP principles like Classes, Objects, Inheritance, Abstraction, etc.

Note:

- ***PLAGIARISM WILL NOT BE TOLERATED*** and will result in 0 mark - especially copy pasting from someone else's repository from GitHub. [WAL Section D4: Academic Dishonesty]
- Any internet articles you refer while solving the Assignment must be put at the top of the code in comments.
- Follow the submission guidelines. Failure to follow the instructions will result in deduction of marks or *0 if none of the instructions are followed.*

Submission:

1. Create a GitHub repository *for the OOP Banking Application*.
2. Submit the GitHub repository link of OOP Banking Application in VLE.
3. Submit by 24th May, 2024 both in VLE and turnitin. *The submission will close on midnight 25th May, 2024. Failure to submit the assignment in either portal will result in it being considered void.*
4. *Your solution program/code file should be named as: CAP2_ID.PY*
   Example: "CAP2_02233456.py"

NOTE: The project should be well-organized and easy to read. Students should use clear and concise language, and they should avoid using jargon. Students should also include comments in their code to explain their code and make it easier for others to understand.

Plagiarism will not be tolerated and will result in a failing grade (0 Marks) for the assignment.

Example Code for Comments and Documentation:
https://github.com/TandinPeljor/Pygame/blob/main/CAP2_Pygame/Hangman.py

Marking Rubrics

Rubric for Practical Assignment III

Criteria	**Description**	**Weightage**	**Accomplished (3)**	**Good (2)**	**Satisfactory (1)**	**Needs Improvement (0)**
Code Quality	Structured, documented, formatted	20%	Modular components, descriptive names, commented logic, properly formatted	Mostly meaningful naming and comments, proper formatting	Limited structure, minimal comments, minor formatting issues	No modularization, cryptic names, no comments, formatting issues
Functionality	Completeness of required gameplay features	30%	All features flawlessly implemented, no bugs	Complete with minor gaps, few bugs	Major features work, some gaps/bugs	Major gaps, bugs prevent functionality
OOP Principles	Encapsulation, inheritance, abstraction	25%	Excellent class abstraction and encapsulation, advanced patterns	Appropriate encapsulation and inheritance	Weak abstraction and OO design	No custom classes or OO principles

Error Handling	Anticipating and handling errors	25%	All edge cases handled elegantly with custom messaging	Reasonable validation and failure handling, few unhandled errors	Basic handling of some errors	Little to no error handling