wesleyw

Create a working model of a small bank

75	**Base Grade w/o Group**
84	**Final Grade with Group**

DIcE Rubric				**Deduction**	**Reason**
Documentation (20%)	***Design Plan (5%)***	Easy to understand, Logical sequence, Clearly defined sections. Provides general overview, etc.	5%		
	Testing Plan (3%)	Provides detailed steps to repeat test. Appropriate coverage of code.	3%		
	Project Writeup (3%)	Does the writeup document challenges and surprises and lessons learned encountered during the project?	3%		
	Grammar/ Spelling (3%)	Was the documentation free of grammatical and spelling errors and formatting inconsistencies?	3%		
	Code Formatting (6%)	Does the formatting of the code adhere to the common style as run through pep8?	6%	3	3 PEP08 errors/warnings Doc string for menu.puy module says is constains class - and functions - when it only contains a class - everyhting else is class methods account.py module docstring and Account docstring seem to be battling over who describes the class - the module should siomply state what is in the module - let the class describe itself
		Are variables/functions named appropriately and enable code readability?			
		Are comments placed appropriately, adhere to the style guide, and enable code readability?			
		Are doc strings utilized and adhere to PEP 257 guidelines?			
		Are classes/modules/files named appropriately and enable code readability?			
		Was borrowed code cited appropriately as per the style guide and the instructor?			
Implementation (30%)	***Version Control (5%)***	Is a branch created to address each requirement or feature? Is the history free of generated/artifact files? Are commit messages informative? Is the main branch free of direct work?	5%		
	Architecture (20%)	Are classes and inheritance used effectively for the problem at hand?	8%	3	your create_main_menu, creates a menu but then creates menu_options - that should be instance data of the Menu object - encapsulate what the main menu has - inside of the class - not inside other functions Your bank_of_ners is nothing but a ton of functions calls - which are better fof in a class called Bank for instance - taht become behavior o fthe bank object
		Was the code designed and constructed in a modular fashion?	4%	2	Your use_teller function of bank of nerds has way too much responsibility and as such has 200+ lines in it. Each elif in the while true - should have call a method/function repsonsible for that behavior - none of the behavior can be reused as parts because it is all in one place
		Were generally sound decisions made with regard to architecture?	4%	4	self.options is a list of strings no need to convert a string to a string on line 93 of menu.py one part of menu system provides numbered list of options, while another prompts for 'type:number:amt' to select the account - numbering and selecting from a number woudl be more consistent user interface The way the code is written - I can douible my money due to your rounding. Instead of deposit a penny and getting a penny - if I deposit a half a penny twice i end up with 2 cents What is the "return 1" of deposit on line 45 of account - and what is the variable "rc" that is used in conjuctuion with it the two put together make the deposit functionality nearly impossible to follow this returning of 1 or -1 is all over the place - with no indication of what it means or is used for
		Is the code DRY?	4%	3	"Savings", "Checking" etc repeated in bank_ofnerds.py lines 39 and 40 both convert idx to a string - convert once use twice lines 37-41 of retirement and 37-41 of savings are identical - put in a method do it in one place and call it in both
	Testing (5%)	Were comprehensive and robust test cases constructed to include but not limited to the test cases provided in this document? Are all tests repeatable?	5%		
	Parsing (5%)	Does the program pass python3 compileall . with no warnings?	5%		
		Does invalid input cause the program to crash?	5%		

Execution (35%)	*Safety (10%)*	Does invalid input cause the program to act inappropriately?	5%	5	when asked for the following 'type:number:amt' entering "a:b:c" terminates the program rather than prompting for correct info - or bringing it back a menu item Ctrl-D should not terminate program - handle it and recover from it in some manner if possible Names with nothing but spaces - and names ith 2k+ characters are notvalid names age of 0 is permitted and should not be Your usage statement gives the --secret away - and therefore is not really a secret
	Requirements (15%)	Were all requirements met?	8%		
		Were all inputs parsed correctly and yield the correct output?	7%	1	Savings:1:1.23456789 should not allow fractional cents as deposit or withdrawal
	Performance (5%)	Scales appropriately with input and execute in a timely manner?	5%		
		Documentation	***20***	***17***	
		Implementation	***30***	***18***	
		Execution	***35***	***29***	
		Total Points Available	***85***	***64***	
			Total Deductions	***21***	

	Area	*Feature*				
Suggested Features (15%)	***Documentation***	Write a man(1) page for your program.	2%	2		
	Documentation	Provide a UML Diagram for the classes in the project	3%	2		balance in subclasses is inherited without being changed and as such should not appear in sublclasses as addiontal variables
	Implementation	Use TDD to write as many tests as possible that can be run automatically. Put tests in a separate subdirectory of the project named ***test.***	4%	2		No refactoring done in test_checking - ~ "checking = Checking(1000) should be refactored into setUp test-withdraw has 3 different sets of tests it should have been broken down into
	Execution	Provide a Money Market Fund account. A MMF must not allow more than two transactions per month. To simulate this in our program, do not allow more than two MMF transactions on a single MMF account during a running of the program.	3%	3		
	Execution	Our customer is eager to retire to Costa Rica, Nice, France, or someplace warm. Allow the customer to withdraw or deposit in alternate currency.	4%	0		Not implemented
	Execution	Be a kind bank and provide overdraft protection for appropriate accounts to your customers. Of course, this protection is provided along with a small overdraft fee of $35 for each overdrawn transaction.	4%	2		Transaction failed: account overdrafted, States it failed, but it actually debited account and charged 35 fee
	Execution	Support the --persist command line flag. When the program is run with this flag, it should persist all data for the bank when the program exits. When the program is re-run with the --persist flag, it should load the data if it exists.	4%	0		
					11	Points out of max 15 available for Features
			Group Points	2	9	

	Requirement	*Area*
Functional Requirements	The project must run on the class Virtual Machine.	Execution
	The program must have accounts and customers. Customers are the account holders.	Execution
	Accounts must provide functionality for withdrawing and depositing.	Execution
	A customer can have multiple accounts such as Savings, Checking, a retirement 401(k), and Money Market Fund (MMF). At a minimum, the program must implement Savings, Checking, and 401(k).	Execution
	The program must provide a menu to display customers or create a new customer.	Execution
	When displaying a particular customer, the program must list all accounts for that customer as well as provide a means to create a new account and deposit into or withdraw from an existing account.	Execution
	There is only one basic type of customer; however, there is no limit to the number and types of accounts a customer can have.	Execution
	A customer can withdraw or deposit any number of times and at any age when working with a Savings or Checking account.	Execution
	A customer cannot withdraw from a 401(k) until they have reached the age of 67.	Execution

	When the program is run, there must be at least two customers that are automatically created.One customer must have at least a Savings account, the other customer must have at least a Savings, Checking, and 401(k) account. A command line flag of "--secret" with a value of "backdoor" should be accepted that then prints out any and all information about the two required customers including any accountinformation needed to access their accounts.	Execution
	Account balances must be managed correctly by following real-world rules depending on the type of account.	Execution

	Requirement	***Area***
Requirements	Design plan, test procedure, and writeup documents must be submitted with the project.	Documentation
	Test Cases used must be submitted with the project.	Implementation
	All source code and documentation must be submitted to the class version control system by 1159EDT on the due date specified.	Implementation
	At least withdraw and deposit must be implemented.	Implementation

	Constraint	***Area***
Constraints	Make use of appropriate variable names.	Documentation
	All documentation must be in PDF format.	Documentation
	PEP-8 code style is required.	Documentation
	Docstrings must be used appropriately.	Documentation
	The project should be stored in your assigned VCS account, under the project name ***bank_of_nerds*** .	Implementation
	No third-party files/libraries may be used unless signed off by the Program Managers or Instructors.	Implementation
	Each logical portion or feature must be built in its own branch.	Implementation
	Merge (do not fast-forward) all commits to branch main and tag releases appropriately.	Implementation
	The default branch to clone should be main.	Implementation
	Code must be DRY when possible.	Implementation
	The project must be written in Python 3.	Implementation
	Program must be invoked as ***./bank_of_nerds.py***	Execution
	Program must not crash or get stuck in an infinite loop.	Execution