

**BSc in Computing & Multimedia**

**OOP - BSC20921 - Semester 1**

**October 2021**

**Continuous Assessment number 1 (Individual )**

**Module Title: Computer Programming**

**Module Code:** ESILV - OOP

**Assessment Type**: Practical Assessment

**Weighting: 40%**

**Maximal Possible Mark:** 100 marks

**Submission Date:** See Moodle for submission date

**Expected Programming Time: 2-3 hours**

# Banking Application

You are required to create a console-based banking application in an OOP language of your choice (e.g. C# or Java or Python) that meets the following requirements outlined below. Where you find ambiguity in the requirements, you are expected to use your initiative and make a judgement call to complete the assignment.

# Logging In

* An opening menu should ask if you are a Bank Employee or a Customer.
* To successfully login as a bank employee you must type in the pin 'A1234'
* To successfully login as a customer, you must enter your first name, last name, account number and pin.

# Bank Employee View

1. As a bank employee you can create and delete customers.
2. Each new customer gets a savings account and a current account.
3. You can only delete customers who have zero balances.
4. You can create transactions (lodge, deposit) for each customer. You should be able to add and withdraw for a specified account.
5. To create a customer account you need first name, last name and email.
6. You should be able to show a complete list of customers including their balances in savings and current account.
7. There should be a menu item allowing you to list customers, their account numbers.

# Customer View

1. To login, a customer must enter their name, account code AND a pin number for their account
2. A customer can retrieve the transaction history for their specified account.
3. They can add and subtract money to either their savings account or current account.
4. They cannot have negative balances

# Creating Accounts

1. Your application should create a file called customers.txt which stores a list of customer accounts. Each account should be stored on a single line; it is up to you to decide the format. This file should be created when the program runs for the first time.
2. The system should create accounts based on the following rules:

The filename will be called **xx-nn-yy-zz** where **xx** is the initials of the customer, **nn** is the length of the total name (first name and last name, **yy** is the alphabetical position of the first initial and **zz** is the alphabetical position of the second initial (see table below) - together they make up the pin number

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |

Example Joe Smith = js-8-10-19: account number is js-8-10-19, pin code is 1019

You will create a savings file is called js-8-10-19-savings.txt, and a current account is called js-8-10-19.current.txt.

Each file stores the transactions for the account.  You are required to have a transaction file for the savings account AND a transactions file for the current account

Each transaction record should store the date, the action (withdraw, lodgement), the money transferred / withdrawn and the final balance. Use tabs ('\t') to delineate the fields

        e.g. 23-01-2017     Withdrawal    25.67    100.17

A bank employee can add and withdraw money on behalf of a specified customer

# Coding Requirements

1. Each code file should contain your name and student number. You will be negatively marked (-5 marks) if you fail to identify your code files.
2. A readme.txt file should be describe what you have done and the challenges you encountered
3. Your code should demonstrate the appropriate use of the following:
   1. Commenting
   2. Interfaces
   3. Inheritance
   4. Functions / reuse
   5. Arrays / Collections
   6. Code should be appropriately separated into files
   7. A simple test module for confirming functions and objects behave appropriately – this can be a ‘driver’ file.
4. Ensure your code is separated into separate files to allow for reusability
5. Ensure your code has a 'testing' module to confirm your functions are working correctly
6. Create a text file (readme.txt) explaining what you have done, what you have completed and what you could not do
7. **Code must be submitted via private repository on github and shared out to** [**outsidethebox@eircom.net**](mailto:outsidethebox@eircom.net)
8. **Working code will be demonstrated in class on November 16th**
9. **Github must show individual contributions**

# Marking Scheme

|  |  |
| --- | --- |
| **Appropriate demonstration of:** | Max Marks |
| Interfaces | 5 |
| Inheritance | 5 |
| Classes | 5 |
| Functions | 5 |
| Arrays | 5 |
| Variables | 5 |
| Code formatting, layout, appropriate file separation | 5 |
|  |  |
| **Meeting coding specifications:** |  |
| Creating files, opening files, reading files | 15 |
| Showing on screen menus | 5 |
| Login specifications (customer / employee) | 10 |
| File naming requirements | 5 |
| Reading and writing transactions appropriately | 10 |
| **Program Construction** |  |
| Appropriate program flow, code organisation | 20 |
| **Total:** | **100 marks** |
|  |  |
| **Negative Marking** |  |
| Failure to identify project, code files with student name and number | -5 per file to max of -15 marks |