**STUDENT - PRODUCT ASSESSMENT TASK**

|  |  |  |  |
| --- | --- | --- | --- |
| Task Number | 3 of 3 . | Task Name | Project |
| National unit/s code | ICTPRG430 | National unit/s title | Apply introductory object-orientation language skills |
| National qualification code | ICT50220 | National qualification title | Diploma of Information Technology |
| RMIT Program code | C5402 | RMIT Course code | COSC7391C |

Section A **- Assessment Information**

|  |  |
| --- | --- |
| **Duration and/or due date:** | Week 15. The exact date and time will be made available to students through CANVAS. |
| **Task Instructions** | |
| This information should be copied into the student version of the assessment task.  **Type of Product (tick which applies)**  Project  Report  Portfolio  Case study  **Summary and Purpose of Assessment**  This assessment is designed to test the students’ ability in understanding fundamental principles in Programming such as:   * Coding syntax and standards * Sequence * Selection and repetition * Functions (Modularity) * Classes and objects * List/ array (data type structure) of objects * Object Aggregation * Polymorphism * Debugging and resolving syntax and logic errors * Internal documentation * File I/O   **Assessment Instructions**  **What**  **This practical assessment consists of 5 Tasks:**   * **Task 1 (1a – 1h):** Create a file called **CAccount.py** and write code to implement the **CAccount** class as shown in the UML class diagram for checking accounts. * **Task 2 (2a – 2h):** Create a file called **SAccount.py** and write code to implement the **SAccount** class as shown in the UML class diagram for saving accounts. * **Task 3 (3a – 3f):** Create a file called **Customer.py** and write code to implement the **Customer** class as shown in the UML class diagram for banking customers. * **Task 4 (4a – 4i):** Create a file called **Bank.py** and write code to implement the **Bank** class as shown in the UML class diagram.   Please Note:   * The code written for Tasks 1, 2, 3 and 4 must follow the processes and techniques related to object-oriented programming, including the concepts and language.   Tasks 1, 2, 3 and 4 must be completed using a valid IDE application.   * **Task 5:** Organise a meeting to demonstrate your work to the client (Teacher). This meeting will take place face to face. You will need to register a time on Canvas to see your client (Teacher).   Task 5 will be a Demonstration of work to the client (Teacher). This meeting will serve as an opportunity to assess your completed application using the checklist provided in Section B.  If your application meets the user requirements, the client (Teacher) will sign off your work on Canvas. If further modifications are required, additional meeting(s) will be required until the product meets the user requirements.  Details are provided below and in Section B: Marking Guide.   * Please refer to **AT3\_Appendix 1** - UML\_Class\_Diagram\_PRODUCT\_ASSESSMENT.xlsx for the UML class diagram You must follow when writing the code for Tasks 1, 2, 3 and 4. * Please refer to **AT3\_Appendix 2** - User\_Requirements\_PRODUCT\_ASSESSMENT.docx for the user requirements You must meet when writing the code for Tasks 1, 2, 3 and 4. * Please refer to **AT3\_Appendix 3** – CAccounts.txt, AT3\_Appendix 4 – SAccounts.txt and AT3\_Appendix 5 – Customers.txt for the data You must use when writing the code for Task 4d.   **Where**  This assessment can be completed at home and in class. Location, due date and time have been set up for the assessment on CANVAS. **Note to assessor: you should have been notified through the announcements at least one week prior to the commencement.**  **How**  Students will be assessed against the criteria listed in the Marking Guide in Section B of this task. To achieve a Satisfactory result, they will need to address all criteria satisfactorily.  **Overview:**  You are a software developer working for a Bank called Maxordia Financial. Your daily tasks include the following:   * Planning, designing and implementing software applications. * Testing and evaluating new software applications. * Maintaining and upgrading existing applications. * Collaborating with other software developers and UX designers. * Writing and implementing code using various programming languages.   It is projected that there will be a large increase for investments in Australia in the next 4 to 6 years due to the ongoing positive economic growth. To meet this demand Maxordia Financial has allocated a budget for developing a new software application for processing banking transactions for checking and saving accounts. As the Software Developer, you will be required to develop the new software application that performs accurate banking transactions for employees and customers.  Below is the UML class diagram showing the structure for developing the new software application:    See AT3\_**Appendix 1 - UML\_Class\_Diagram\_PRODUCT\_ASSESSMENT.xlsx** for an enlarged size of the UML class diagram above.  The user requirements of the software application will be provided to you by the client (Teacher).  See AT3\_**Appendix 2 - User\_Requirements\_PRODUCT\_ASSESSMENT.docx** for the user requirements of the software application.  **Task 1:**  Create a file called **CAccount.py** and write code to implement the **CAccount** class as shown in the UML class diagram for checking accounts.  The **CAccount** class must contain the following:   1. 1 class variable assigned the value of **1246** 2. 4 primitive instance variables – as named per the UML diagram. 3. Constructor containing two options for creating an object of the **CAccount** class:   **1st option** is creating the object with user specified values for all 4 instance variables.  **2nd option** is creating the object with user specified values for only 3 instance variables and assigning default value of **50.00** for 4th instance variable.   1. Getter functions for accessing the values of the 4 specified instance variables. 2. Function using polymorphism for performing deposit transactions on the object of the **CAccount** class according to the user requirements of the client (Teacher). 3. Arithmetic operator for calculating the deposit transactions for checking accounts. 4. Function for displaying the object of the **CAccount** class in a String format according to the user requirements of the client (Teacher). 5. Coding conventions of the programming language such as appropriate variable names and indentation that specify at least 3 aspects, according to organisational requirements.   **Task 2:**  Create a file called **SAccount.py** and write code to implement the **SAccount** class as shown in the UML class diagram for saving accounts.  The **SAccount** class must contain the following:   1. 1 class variable assigned the value of **1246** 2. 4 primitive instance variables – as named per the UML diagram 3. Constructor containing two options for creating an object of the **SAccount** class:   **1st option** is creating the object with user specified values for all 4 instance variables  **2nd option** is creating the object with user specified values for only 3 instance variables and assigning default value of **500.00** for 4th instance variable   1. Getter functions for accessing the values of the 4 specified instance variables 2. Function using polymorphism for performing deposit transactions on the object of the **SAccount** class according to the user requirements of the client (Teacher) 3. Arithmetic operator for calculating the deposit transactions for saving accounts 4. Function for displaying the object of the **SAccount** class in a String format according to the user requirements of the client (Teacher) 5. Coding conventions of the programming language such as appropriate variable names and indentation that specify at least 3 aspects, according to organisational requirements   **Task 3:**  Create a file called **Customer.py** and write code to implement the **Customer** class as shown in the UML class diagram for banking customers.  The **Customer** class must contain the following:   1. 5 primitive instance variables – as named per the UML diagram 2. Constructor for creating an object of the **Customer** class with user specified values for all 5 instance variables 3. 5th instance variable using aggregation to store an object of the **CAccount** or **SAccount** class 4. Getter functions for accessing the values of the 5 specified instance variables 5. Function for displaying the object of the **Customer** class in a String format according to the user requirements of the client (Teacher) 6. Coding conventions of the programming language such as appropriate variable names and indentation that specify at least 3 aspects, according to organisational requirements   **Task 4:**  Create a file called **Bank.py** and write code to implement the **Bank** class as shown in the UML class diagram.  All code must be written only inside a single function called main()  The **Bank** class must contain the following:   1. 1st list/ array (data type structure) to store objects of the **CAccount** and **SAccount** classes   2nd list/ array (data type structure) to store objects of the **Customer** class   1. instancevariable for assigning each object of the **CAccount** / **SAccount** class to the object of the **Customer** class   The main()function for running the entire code in a sequence.  The main()function must be placed inside the **Bank** class.   1. **Reading** the data from **CAccounts.txt** file to create objects of the **CAccount** class inside the 1st list/ array (data type structure).   Reading the data from **SAccounts.txt** file to create objects of the **SAccount** class inside the 1st list/array (data type structure).  Reading the data from **Customers.txt** file to create objects of the **Customer** class inside the 2nd list/ array (data type structure).  Each object of the **CAccount** / **SAccount** class contained inside the 1st list (data type structure) must be assigned to each object created of the **Customer** class inside the 2nd list/ array (data type structure) according to the user requirements of the client (Teacher).   1. Using getter functions to display details of a checking account (**CAccount** object) for a customer before performing the deposit transaction according to the user requirements of the client (Teacher).   Using getter functions to display details of a saving account (**SAccount** object) for a customer before performing the deposit transaction according to the user requirements of the client (Teacher).   1. Performing an invalid deposit transaction on the object of the **CAccount** class contained inside the 1st list/ array (data type structure).   Performing an invalid deposit transaction on the object of the **SAccount** class contained inside the 1st list/ array (data type structure).   1. Performing a valid deposit transaction on the object of the **CAccount** class contained inside the 1st list/ array (data type structure).   Performing a valid deposit transaction on the object of the **SAccount** class contained inside the 1st list/ array (data type structure).   1. Inserting details of a checking account (**CAccount** object) into **BankingReceipt.txt** file after performing the deposit transaction according to the user requirements of the client (Teacher).   Inserting details of a saving account (**SAccount** object) into **BankingReceipt.txt** file after performing the deposit transaction according to the user requirements of the client (Teacher).   1. Coding conventions of the programming language such as appropriate variable names and indentation that specify at least 3 aspects, according to organisational requirements.   **Task 5:**  Organise a meeting to demonstrate your work to the client (Teacher). This meeting will take place face to face. You will need to register a time on Canvas to see your client (Teacher).  This meeting will serve as an opportunity to assess your completed application using the checklist provided in Section B.  During this meeting you may be asked several questions to demonstrate your understanding of the code.  If your application meets the user requirements, the client (Teacher) will sign off your work on Canvas. If further modifications are required, additional meeting(s) will be required until the product meets the user requirements.  **Additional Instructions:**   * The use of ChatGPT or any other AI tool is **STRICTLY PROHIBITED** for this Assessment. * Attempt ALL the questions/tasks in this Assignment. * Performance requirement:   + **Satisfactory (S) performance** - met the minimum requirement of all the questions /tasks listed for the Assignment Task.   + **Not Yet Satisfactory (NYS) performance** - did not meet the minimum requirement of all the questions /tasks listed for the Assignment Task. * Students need to achieve satisfactory (S) results in all three (3) assessments to be deemed Competent (CA) | |
| **Conditions for assessment** | |
| * You must complete the task within the maximum allowed duration as directed by the assessor. * This is an individual assessment task. You will be assessed individually against all assessment criteria. * Use of any AI assisted tool is strictly prohibited. * All work in this assessment must be in your own words. * Use of code that is not your own is strictly prohibited. * Students must attend a meeting with their assessor during a specified time. If a student does not attend this meeting, they will not be graded and may forfeit their submission attempt. * All work must be submitted prior to the due date, not the date of the meeting. If a student submits their work on the day of the meeting without an approved extension, this will count as their resubmission attempt. * This assessment will be undertaken both in class and at home. Class-time will be provided to complete this assessment. * You can make arrangements with the assessor at least one week prior to the assessment due date if you feel you require special allowance or allowable adjustment to this task. * Students found in breach of assessment conditions can be charged with academic misconduct, have their results cancelled, be excluded from the program and receive other penalties. Penalties can also apply if a student’s test material is copied by others. * Plagiarism is the presentation of the work, idea or creation of another person as though it is one’s own. It is a form of cheating and is a very serious academic offence that may lead to expulsion from the University. Plagiarised material can be drawn from, and presented in, written, graphic and visual form, including electronic data, and oral presentations. Plagiarism occurs when the origin of the material used is not appropriately cited. * RMIT special consideration is to enable students to maintain your academic progress despite adverse circumstances. The process for special consideration can be found at <http://www1.rmit.edu.au/students/specialconsideration> * Students with a disability or long-term medical or mental health condition can apply for adjustments to their study and assessment conditions (Reasonable Adjustments and Equitable Assessment Arrangements) by registering with the Equitable Learning Services (ELS) at https://www.rmit.edu.au/students/support-and-facilities/student-support/equitable-learning-services.If student has already registered with ELS and his/her study plan is approved, student should inform the teacher if this assessment task is not adjusted in line with approved study plan. * Please ensure your full and correct name is written on the student version of this assessment task (do not use nicknames or abbreviations). * You will be assessed as satisfactory or not yet satisfactory. * You can appeal the assessment decision according to the RMIT Assessment Appeal Processes. * You will have the opportunity to resubmit any tools that are deemed unsatisfactory (one resubmission allowed per unit, so that means students have two opportunities to submit). | |
| **Useful Links** | |
| Please use links below to access important information on assessments:    Assessment submission and appeals - [Assessments](https://www.rmit.edu.au/students/my-course/assessment-results/assessment)  Extensions and special considerations -  [Extensions and special consideration](https://www.rmit.edu.au/students/my-course/assessment-results/special-consideration-extensions/equitable-assessment-arrangements)  Equitable Assessment Arrangement (EAA) - [Equitable Assessment Arrangements](https://www.rmit.edu.au/students/support-services/equitable-learning)  Information on recording and retaining assessments and dealing with assessment appeals can be found in the [RMIT Assessment Processes document](https://policies.rmit.edu.au/download.php?associated=1&id=348&version=2) | |
| **Instructions on Submitting your Assessment** | |
| You must submit a zip file containing the following:   * Python files (.py) for tasks 1,2, 3 and 4:   + **CAccount.py**   + **SAccount.py**   + **Customer.py**   + **Bank.py**   + Use comments (**#**) at the top of the code inside each file that contains:     - Student Number     - Student Full Name * The naming convention of the zip file is: **<Student Number>\_<Student Full Name>\_Python\_Project.zip** | |
| **Equipment/resources students must supply:** | **Equipment/resources to be provided by RMIT or the workplace:** |
| * External hard drive to store electronic files * Pens * Notebook * Laptop (optional) | * Onsite computers with internet connectivity and Visual Studio application * Operating system with Python programming language * Canvas access * Appendix 1 – UML\_Class\_Diagram\_PRODUCT\_ASSESSMENT.xlsx - To be used for Tasks 1, 2, 3 and 4 * Appendix 2 – User\_Requirements\_PRODUCT\_ASSESSMENT.docx - To be used for Tasks 1, 2, 3 and 4 * Appendix 3 – CAccounts.txt - To be used for Task 4d * Appendix 4 – SAccounts.txt - To be used for Task 4d * Appendix 5 – Customers.txt - To be used for Task 4d |

Section B **– Marking Guide**

Below is a checklist for how this assessment task will be judged as satisfactory or not satisfactory.

| **Key Criteria that must be demonstrated** | | | | |
| --- | --- | --- | --- | --- |
| *Students need to follow the Assessment Instructions for Tasks 1, 2, 3, 4 and 5 and submit all the required answers. Students must complete* ***ALL*** *the listed criteria satisfactorily to successfully complete this assessment.* | | | | |
| **Criteria for Assessment**  *It is recommended that for product assessments you divide your criteria into key sections and under each section clearly describe the criteria students must demonstrate* | | **Satisfactory** | | **Comment** |
| **Y** | **N** |
| *1* | **Task** **1**  Student developed CAccount class that adhered to the following:  **Task 1a**  class variable assigned the value of 1246  **Task 1b**  4 primitive instance variables – as named per the UML diagram. |  |  |  |
| *2* | **Task 1c** Student developed CAccount class that adhered to the following:  Constructor containing two options for creating an object of the CAccount class:   * 1st option is creating the object with user specified values for all 4 instance variables. * 2nd option is creating the object with user specified values for only 3 instance variables and assigning a default value of 50.00 for 4th instance variable. |  |  |  |
| *3* | **Task** **1d**  Student developed CAccount class that adhered to the following:  Getter functions for accessing the values of the 4 specified instance variables. |  |  |  |
| *4* | **Task** **1e**  Student developed CAccount class that adhered to the following:  Function using polymorphism for performing deposit transactions on the object of the CAccount class according to the user requirements of the client (Teacher)  **Task** **1f**  Arithmetic operator for calculating the deposit transactions for checking accounts. |  |  |  |
| *5* | **Task** **1g**  Student developed CAccount class that adhered to the following:  Function for displaying the object of the CAccount class in a String format according to the user requirements of the client (Teacher). |  |  |  |
| *6* | **Task** **1h**  Student developed CAccount class that adhered to the following:   * Coding conventions of the programming language (at least three (3) aspects). * CAccount class must be developed according to organisational requirements. |  |  |  |
| *7* | **Task** **2**  Student developed SAccount class that adhered to the following:  **Task 2a**  1 class variable assigned the value of 1246.  **Task 2b**  4 primitive instance variables – as named per the UML diagram. |  |  |  |
| *8* | **Task** **2c**  Student developed SAccount class that adhered to the following:  Constructor containing two options for creating an object of the SAccount class:  **1st option** is creating the object with user specified values for all 4 instance variables.  **2nd option** is creating the object with user specified values for only 3 instance variables and assigning a default value of 500.00 for 4th instance variable. |  |  |  |
| *9* | **Task** **2d**  Student developed SAccount class that adhered to the following:  Getter functions for accessing the values of the 4 specified instance variables. |  |  |  |
| *10* | **Task** **2e**  Student developed SAccount class that adhered to the following:  Function using polymorphism for performing deposit transactions on the object of the SAccount class according to the user requirements of the client (Teacher) |  |  |  |
| *11* | **Task** **2f**  Arithmetic operator for calculating the deposit transactions for saving accounts. |  |  |  |
| *12* | **Task** **2g**  Student developed SAccount class that adhered to the following:  Function for displaying the object of the SAccount class in a String format according to the user requirements of the client (Teacher). |  |  |  |
| *13* | **Task** **2h**  Student developed SAccount class that adhered to the following:   * Coding conventions of the programming language (at least three (3) aspects). * SAccount class must be developed according to organisational requirements. |  |  |  |
| *14* | **Task 3**  Student developed Customer class that adhered to the following:  **Task 3a**  5 primitive instance variables – as named per the UML diagram |  |  |  |
| *15* | **Task 3b**  Student developed Customer class that adhered to the following:  Constructor for creating an object of the Customer class with user specified values for all 5 instance variables**.** |  |  |  |
| *16* | **Task 3c**  Student developed Customer class that adhered to the following:  5th instance variable using aggregation to store an object of the CAccount or SAccount class |  |  |  |
| *17* | **Task 3d**  Student developed Customer class that adhered to the following:  Getter functions for accessing the values of the 5 specified instance variables. |  |  |  |
| *18* | **Task 3e**  Student developed Customer class that adhered to the following:  Function for displaying the object of the Customer class in a String format according to the user requirements of the client (Teacher). |  |  |  |
| *19* | **Task 3f**  Student developed Customer class that adhered to the following:   * Coding conventions of the programming language (at least three (3) aspects) * Customer class must be developed according to organisational requirements. |  |  |  |
| *20* | **Task 4**  Student developed Bank class that adhered to the following:  **All code is written only inside a single function called main()**  **Task 4a**   * 1st list/array (primitive data type structure) to store objects of the CAccount and SAccount classes. * 2nd list/array (primitive data type structure) to store objects of the Customer class. |  |  |  |
| *21* | **Task 4b**  Student developed Bank class that adhered to the following:  **All code is written only inside a single function called main()**  variable for assigning each object of the CAccount / SAccount class to the object of the Customer class. |  |  |  |
| *22* | **Task 4c**  Student developed Bank class that adhered to the following:   * All code is written only inside a single function called main() * The main() function for running the entire code in a sequence. * The main() function must be placed inside the Bank class. |  |  |  |
| *23* | **Task 4d**  Student developed Bank class that adhered to the following:   * + All code is written only inside a single function called main()   + Reading the data from **CAccounts.txt** file to create objects of the **CAccount** class inside the 1st list/ array (data type structure).   + Reading the data from **SAccounts.txt** file to create objects of the **SAccount** class inside the 1st list/ array (data type structure).   + Reading the data from **Customers.txt** file to create objects of the **Customer** class inside the 2nd list/ array (data type structure).   + Each object of the **CAccount** / **SAccount** class contained inside the 1st list/ array (data type structure) must be assigned to each object created of the **Customer** class inside the 2nd list/ array (data type structure) according to the user requirements of the client (Teacher). |  |  |  |
| *24* | **Task 4e**  Student developed Bank class that adhered to the following:   * All code is written only inside a single function called main() * Using getter functions to display details of a checking account (**CAccount** object) for a customer before performing the deposittransaction according to the user requirements of the client (Teacher). * Using getter functions to display details of a saving account (**SAccount** object) for a customer before performing the deposit transaction according to the user requirements of the client (Teacher). |  |  |  |
| *25* | **Task 4f**  Student developed Bank class that adhered to the following:   * All code is written only inside a single function called main()   + Performing an invalid deposit transaction on the object of the **CAccount** class contained inside the 1st list/array (data type structure).   + Performing an invalid deposit transaction on the object of the **SAccount** class contained inside the 1st list/ array (data type structure). |  |  |  |
| *26* | **Task 4g**  Student developed Bank class that adhered to the following:   * All code is written only inside a single function called main()   + Performing a valid deposit transaction on the object of the **CAccount** class contained inside the 1st list / array (data type structure).   + Performing a valid deposit transaction on the object of the **SAccount** class contained inside the 1st list/ array (data type structure). |  |  |  |
| *27* | **Task 4h**  Student developed Bank class that adhered to the following:   * All code is written only inside a single function called main() * Inserting details of a checking account (**CAccount** object) into **BankingReceipt.txt** file after performing the deposit transaction according to the user requirements of the client (Teacher).   Inserting details of a saving account (**SAccount** object) into **BankingReceipt.txt** file after performing the deposit transaction according to the user requirements of the client (Teacher). |  |  |  |
| *28* | **Task 4i**  Student developed Bank class that adhered to the following:   * All code is written only inside a single function called main() * Coding conventions of the programming language (at least three (3) aspects).      * Bank class must be developed according to organisational requirements. * Correct syntax for the code as specified in the technical documentation of the programming language. |  |  |  |
| *29* | **Task 5**  Student presented the completed application to the client (Teacher) to be signed off on the assessment |  |  |  |

Section C **– Feedback to Student**

|  |  |  |  |
| --- | --- | --- | --- |
| **Has the student successfully completed the task?** | | Yes No | |
|  | |  |  |
| **Feedback to student:** | | | |
|  | | | |
| **Assessor Name** | **Date** | | |