School of Computing and Information Systems

PROGRAMME: BSC COMPUTER SYSTEMS ENGINEERING

CSE202- OBJECT ORIENTED ANALYSIS & DESIGN WITH JAVA Year 2 Semester 1

ASSIGNMENT

Hand Out Date: 12 Sep 2025 **Hand In Date: 17 Nov 2025**

Total Marks: 100

Instructions to candidates

1. Candidates must attempt **ALL** questions.
2. You are to make your submission on turn-it-in. You may consult with your tutor/lecturer on how this will be done.
3. Ensure that you have an account on turn-it-in by going to **www.turnitin.com**. Use the credentials provided for accessing this system. If you do not have them, get hold of the tutor/lecturer as soon as possible.
4. Any work with plagiarism level above **30 % will not be marked.** It your responsibility to make sure that your plagiarism level is within this level. Monitor it on regular bases. If your share your solution with others, chances of the plagiarism rising above this level are high.
5. Its your responsibility to ensure that you have **YOUR** module in turn-it-in before submission date and you do not drop the module. Consult with your tutor/lecturer if this is not the case.

Assignment Mark:= (Part A) * **0.4** + (Part B) * **0.6**

This assignment guides you through the entire software development process for a Banking System, with a focus on **Object-Oriented Analysis and Design (OOAD)**. You will move from initial requirements to a fully integrated application using modern tools like **GitHub Codespaces**.

The assignment is partly based on the tutorials that will be done during the semester during the tutorial sessions. It will be assessed in two (2) parts as follows:

Assignment Component	**Date Week**
Part A	
❑ Requirements Elicitation	Sep. 08 - 12
❑ Behavioural Modelling	Sep. 15 - 19
Part B	
❑ Implementation of Core Model	Oct. 06 - 10
❑ GUI Design & Implementation	Oct. 13 - 17
❑ Controller Implementation	Oct. 20 - 24
❑ Database Connectivity	Oct. 27 - 31
❑ Module Integration & Testing	Nov. 03 - 07
Final Presentation of Work	Nov. 24 – 28

SCENARIO: The BANKING System

Banking has over the years been a fundamental aspect of our daily lives. Bank systems have been changing with technology to foster great service delivery to clientele. Banks offer to **individuals** or **companies**/organisations, called **Customers**, services where they can securely put their money for future use, or where they can be paid their funds on called an Account. In a Bank, a customer can open or have one or multiple accounts. Accounts in a bank differ according to what and how the customer may prefer. These include;

1. A **Savings** account where a customer may deposit funds for future use. This account is capable of paying some monthly interest based on the amount, balance, in the account. This account does not allow any withdrawals from it.

2. The **Investment** account which is an account that pays more interest than the savings account. The customer is allowed to deposit funds into the account as well as be able to withdraw any funds they need. The account can only be opened with an initial deposit of BWP500.00.
3. A **Cheque** account that is normally used for crediting of salaries. Customers normally get their salary payment through this account. The account allows deposits and withdrawals of any amount of funds. This account can only be opened for any person who is working. The customer in this case must provide information on where they are working (i.e. the company, the company address,)

Identify Element/Entities

Account, Customer

Add Attributes and Characteristics

Account (accountNumber, balance, branch, etc.). REMEMBER: an account cannot exist without being held by a customer.

Customer (firstname, surname, address, etc.)

The bank solution MUST allow a customer to have multiple accounts (i.e. a customer can have a **savings** and an **investment**/interest bearing account or all the three types of accounts).

The design MUST depict majority, if not **ALL OOP** principles including but not limited to interfaces, polymorphism, abstraction, inheritance, overriding and overloading.

Developing the Banking System Program

The Banking System must;

1. Allow a customer to open an account with a bank
2. Allow customer to make deposits in any or all of the accounts they hold.
3. Ensure interest is paid to the appropriate accounts to customer. i.e. 5% monthly for a **InvestmentAccount** and 0.05% monthly for **SavingsAccount**.

Part A: System Documentation Instructions (40%)

This assignment is a continuous and weekly based. You are required to submit some work every week on the day and time as prescribed by the Module Leader.

WEEK Sep. 08 – 12

1. Requirements Elicitation [10 marks]

Objective: Elicit and specify the system's requirements to create a solid foundation for development.

1.1. **Functional Requirements** **[5 marks]** (what the system does): A clear and concise explanation of the Functional Requirements. You are expected to conduct a mock interview with a client (e.g., the instructor/tutor/lecturer) to identify the core functions and quality attributes of the banking system. Consider features like customer registration, user authentication, account management, and transaction history.

An appendicle of the **Interview Record**.

1.2. **Non-Functional Requirements**: **[5 marks].** A clear description of the non-functional features of the system (e.g., security, performance, usability).

2. Structural UML Modelling [20 marks]

Objective: Visually model the static structure of the system using UML diagrams.

2.1. System Use Case Diagram [10 marks]

Create a **Use Case Diagram** to represent the system's external behavior. Identify the key actors and use cases.

- **Correct identification of actors and use cases (5 marks)**
- **Clear and accurate relationships shown (e.g., extends, includes) (5 marks)**

2.2. Class diagram [10 marks]

- **2 marks for abstraction, 2 marks for inheritance, 2 marks using an interface, 1 mark for correct diagrams, 1 mark for encapsulation, 1 mark for correct class structures.**

WEEK Sep. 15 – 19

3. Behavioural UML Modelling) [10 marks]

Objective: Model the dynamic behavior of the system and how objects interact over time.

3.1. **Sequence Diagrams:** Create a Sequence Diagram for two of the most critical use cases (e.g., *Login* and *Dposit* Funds). The diagram should show the flow of messages between objects in a time-ordered sequence.

- **Accurate and logical flow of messages between objects (6 marks)**

3.2. **State Diagram:** Select a key class with a complex lifecycle, such as PayInterst, and create a State Diagram to show all possible states an object can be in and the events that cause transitions between them.

- **Correct states and transitions for the selected class (4 marks)**

Deliverable: Two Sequence Diagrams and one State Diagram.

Part B - System Implementation (60%)

WEEK Oct. 06 – 10

4. Implementation of Core Model [15 marks]

Objective: Begin implementing the core business logic of the system in a shared development environment.

For this part of the assignment, you are required to use you **GitHub codespaces** to create all classes. You may work on the local repository and use GIT to upload your code into the remote GitHub repository you have created. This allows your tutor/lecturer to validate, assess and run your code from **codespaces.**

You are to convert the domain model classes (*see Class Diagrams*), into some working code. Focus only on the attributes and basic business logic (e.g., *deposit*, *withdraw*, etc). Do not add GUI or database code yet.

- **Project is correctly set up in a public or private GitHub repository (5 marks)**
- **Correct implementation of core model classes with attributes and basic business logic (10 marks)**

WEEK Oct. 13 – 17

5. GUI Design & Implementation [10 marks]

Objective: Design and implement the user interface, separating it from the core logic.

This week period, you are expected to create GUI components or screens that will allow user to interact with the system. You shall be using JavaFX to achieve this. Create the Boundary Classes that represent the GUI (e.g., *LoginView*, *AccountView*). These classes should be responsible for displaying information to the user and collecting their input. They should not contain any business logic.

- **Logical and user-friendly GUI design (5 marks)**
- **Correct implementation of boundary classes, with no business logic (5 marks)**

WEEK Oct. 20 – 24

6. Controller Implementation [10 marks]

Objective: Implement the classes that manage the flow of control and application logic.

- Controller Classes: Create Controller Classes (e.g., *LoginController*, *AccountController*). The controllers act as intermediaries between the boundary classes and the core domain classes. They handle user requests, validate input, and orchestrate the necessary business logic.
  Correct implementation of controller classes to mediate between the GUI and the core model [6marks]
- Connecting Layers: Refactor your code to ensure the View classes only communicate with the Controller classes, and the Controller classes handle all interactions with the core domain model.
  Proper separation of concerns (no business logic in the GUI, no GUI logic in the core model) [4 marks]

WEEK Oct. 27 – 31

7. Database Connectivity [10 marks]

Objective: Add a persistence layer to your application using JDBC for database connectivity.

- **Database Setup:** Set up a simple local or in-memory database (e.g., H2, SQLite) within your Codespace. Create a database schema for your core classes (customers, accounts, transactions).
- **JDBC Classes:** Create the data access layer classes, often called Data Access Objects (DAOs) or repository classes (e.g., AccountDAO, CustomerDAO). These classes will contain all the JDBC code to connect to the database and perform CRUD (Create, Read, Update, Delete) operations. Deliverable: A Git commit with the database schema and JDBC implementation.

  Correctly defined database schema that aligns with the Class Diagram **(5 marks)**

  Correct implementation of a data access layer (DAO) for CRUD operations **(5 marks)**

WEEK Nov. 03 – 07

8. Module Integration & Testing [10 marks]

Objective: Integrate all the components of your application and perform a final round of testing.

- ❑ **Full Integration:** Connect all the modules you have built over the past weeks: the Boundary (GUI) classes, the Controller classes, the Core domain model classes, and the JDBC data access classes.

  Successful integration of all modules into a single, working application (5 marks)

- ❑ **Integration Testing:** Write a short integration test to verify that the entire system works as a whole. For example, test a full user flow from a user logging in through the GUI, to the controller handling the request, to the core model performing a transaction, and finally to the JDBC class updating the database.

  Clear and comprehensive report detailing the integration tests and their outcomes (5 marks)

WEEK Nov. 24 – 28 [TBC]

9. Final Presentation of Work [5 marks]

0 = Needs Working **2 = Satisfactory** **3 = Commendable** **5 = Excellent**

GUIDELINES

1. Requirements Engineering documentation should include UML diagrams and written descriptions of each class or module and to be submitted on TurnItIn.

2. UML diagrams should be done professionally with Visual Paradigm or Dia Portable

3. A MySQL or Oracle DBMS may be used to store application data

4. A sample records of 10 must be inserted into the database at the time of submission presentation.

The submissions on TurnItin (TII) must be on the set deadline and time. We practice zero tolerance for late submission

SUBMISSION OF ASSIGNMENTS USING TURNIT IN GUIDELINES

NOTES TO STUDENTS!

All assignments for year 1, 2 and 3 will be submitted through TurnItIn, a system that is meant to assist students to write and submit original work that minimizes plagiarism. All students have been given access to the TurnIt In system. Care should be taken to ensure that your paper is unbiased and accurately referenced using the Harvard referencing system. Quotations should be no more than two lines long and, taken all together, should represent less than 10% of the words written. The remaining 90% of this assignment must be entirely in your own words. The TurnItIn URL will be given to students to check for their registration

Assignment Submission and Marking

Your Part A of the assignment should be submitted as one document. This should start with a title page which includes the following information: - the module code, title of paper, your name and student registration number. The title page should then be followed by the body of the assignment which should be a correctly formatted document. The assignment will be submitted electronically via TurnItIn and there is no requirement for any other form of submission. Uploading your assignment constitutes the submission. Turnitin will be closed at 1159pm on the submission date and therefore there is no room for late submissions. Assignments submitted in Microsoft Word format will be accepted for marking. Files must be less than 20 MB. Page **10** of **10**

Essential Information

1. *This is an individual assignment, the work must be entirely your own. The safety of your assessments is your responsibility. You must not permit another student access to your work.*

2. *Your assignment will be submitted electronically via TurnIt In. You must therefore sort out any module registration log in problems as soon as possible for your module and upload your assignment by the date specified. If you cannot log in and upload an assignment by the due date you will fail this assignment. You are strongly recommended to upload a draft assignment at least 1 week before the deadline and to keep uploading revised versions. Technical problems on the deadline day will not be accepted as a valid excuse for non-submission.*

3. The assignment deadline will be midnight on the date specified by your tutor.

4. *Plagiarism, paraphrasing and downloading large amounts of information from external sources, will not be tolerated and will be dealt with severely.*

5. *All text taken from other sources must be in quotations and the sources cited. Quotations should be no more than two lines long and, taken all together, should represent less than 10% of the words written. The remaining 90% of this assignment must be entirely in your own words.*

6. *You should upload a draft assignment report much earlier and respond to the 'TurnItIn' report generated. This report will indicate any non-original words in your paper including a) correctly quoted text, b) the reference list and c) and any plagiarized text. Reports that come back rated at less than 20% non-original text are usually fine. Those that come back with a score of over 25% i.e. yellow, orange or red usually need fixing. Plagiarized text will not be tolerated. Please note that the generation of originality reports can take up to 24 hours though often this is available much sooner and in some cases within minutes.*

7. *You can repeatedly submit your assignment up until the deadline and last assignment uploaded at the deadline will be marked. You will be marked online and will be able to see your marks and feedback online at the same location).*

8. *If you cannot complete this assignment for reasons that are outside of your control, e.g. serious illness, you can apply to the mitigation panel asking for a deferral but you will need to provide appropriate evidence. Technical problems on the day of the assignment deadline, module registration issues and failure to back up your work will not be accepted as valid excuses therefore you should a) ensure you can log into the module b) upload a draft assignment early and keep uploading revisions as you make changes c) keep electronic copies of your work.*