**Table of Content**

[**Introduction**](#_xn6dhze9e71) **2**

[**Project Goal and Requirements**](#_lyeypcrd2thr) **2**

[**Object-Oriented Principles in Practice**](#_hjyfl9ac2hr4) **2**

[Encapsulation](#_qtfhw7bgzxr9) 2

[Inheritance](#_exbh4y36p0zl) 2

[Composition](#_4xmmdd7oo9ip) 3

[**Design Decisions and UML Class Diagram**](#_cv88mr6g9oeb) **3**

[**Critical Analysis and Justification**](#_6zo80uvqtbkq) **3**

[**Conclusion**](#_w4i6diiwv86w) **4**

# **Introduction**

The project exemplifies outstanding mastery of Object-Oriented Programming (OOP) by designing and building a basic banking system framework. The software implementation emphasizes encapsulation as well as inherits and composes different components. The author demonstrates theoretical understanding together with practical knowledge making sure all elements follow rational justifications that maintain connections to final assignment objectives.

# **Project Goal and Requirements**

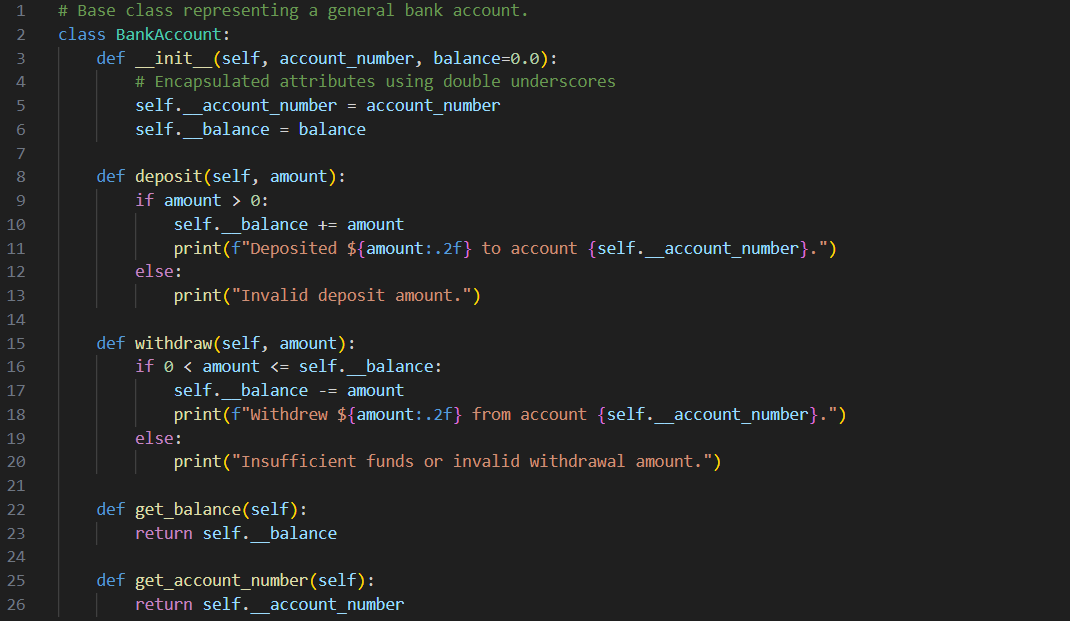
Developing a small piece of software stands as the primary task which shows basic usage of OOP principles. The assignment contains two main sections:

1. A UML class diagram showing the organizational structure of the application. A visual representation shows the program components alongside their defined properties and functions together with all relationship connections.
2. The application uses Python language for its development and requires three distinct class definitions. One class shows inheritance by hiding its internal data through encapsulation while methods allow exposure of functionalities to external access.

# **Object-Oriented Principles in Practice**

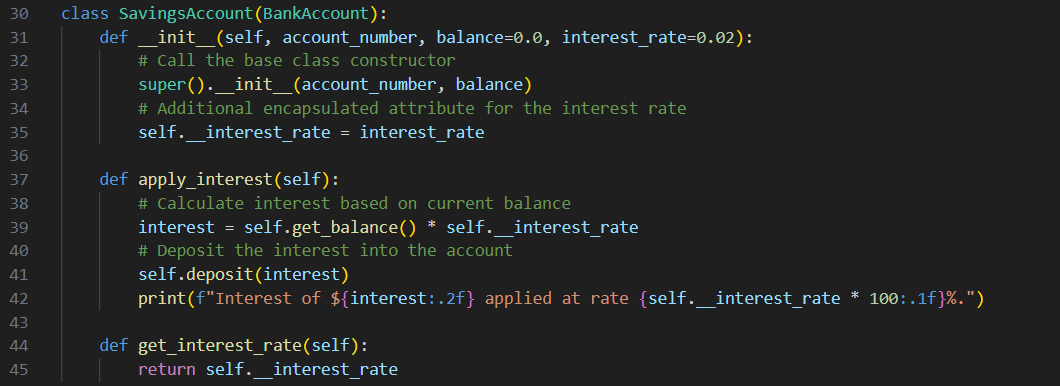
## **Encapsulation**

First-class concepts play a central role throughout banking system design processes. Inside the classes the sensitive attributes such as account balance and account number protect themselves from outside alterations because they maintain private accessibility. The application implementsDeposit(), Withdraw() and getBalance() methods for external use which enable controlled access to the private attributes. Designing the banking system this way implements data hiding principles which protects against unwanted object manipulation and ensures users have clear interfaces during object manipulation.



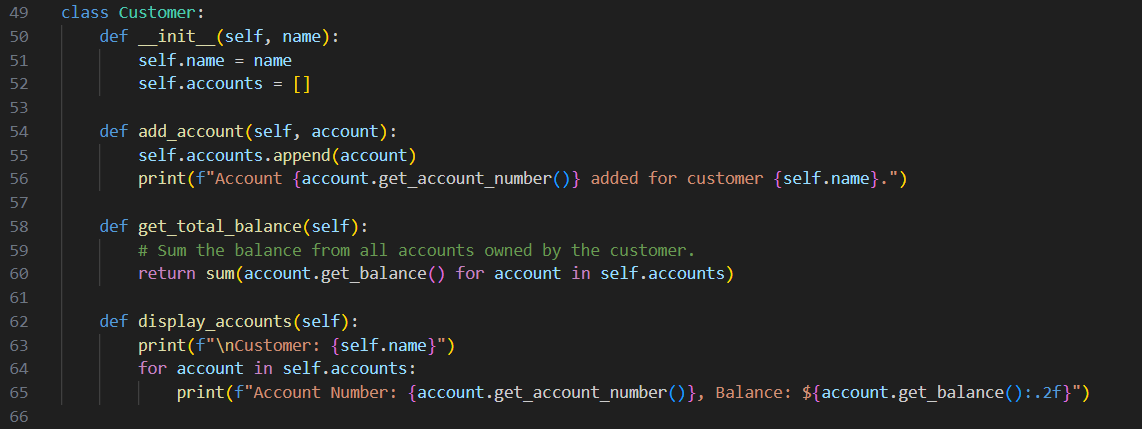
## **Inheritance**

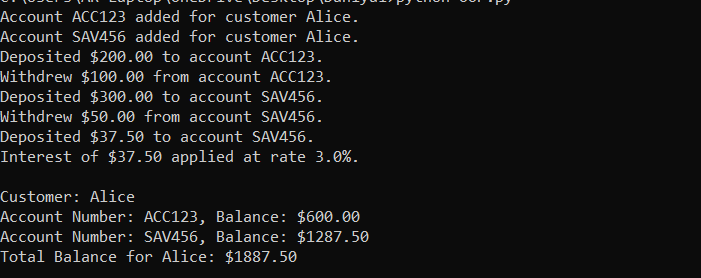
The application uses inheritance to create the SavingsAccount class that builds upon BankAccount base functionalities. Through inheritance the savings account benefits from all standard properties and behaviors of bank accounts while independently implementing interest handling capabilities. The inheritance feature in this method enables code reuse while representative of how banks handle multiple account types which have common features alongside account-specific requirements.



## **Composition**

The Customer class provides an example to explain composition concepts. The class contains bank account objects within its collection structure so customers can control multiple accounts. When OOP models real-life relationships it shows flexibility through the use of composition to build complex objects from simpler ones. The design achieves better maintainability and scalability because it stores account objects inside the customer class structure.





# **Design Decisions and UML Class Diagram**

UML class diagrams play an essential role in design processes because they provide graphical illustrations of the system architecture. The diagram consists of three primary classes and displays relationships between them.

**BankAccount:**

* This class contains two private members for account number and balance.
* The public interface provides methods allowing depositing money, withdrawing money and fetching account information.

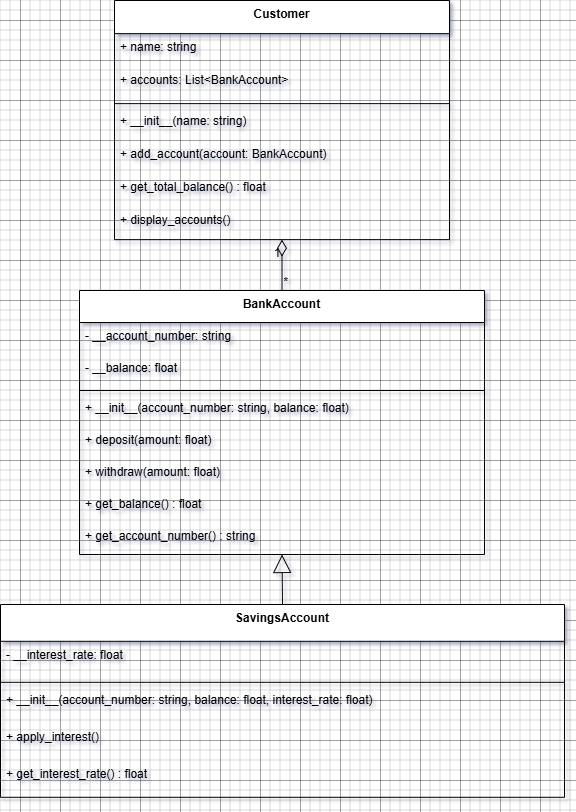
**SavingsAccount:**

* Inheritance allows BankAccount to gain features related to savings account operations.
* Attributes: An additional private attribute for the interest rate.
* The class implementation includes capabilities to calculate interest on present-value balance amounts while performing computations.

**Customer:**

* Composition: Aggregates multiple BankAccount objects.
* The class includes creation methods with functionality for total fund calculations and account display methods.

An open arrow shows inheritance on the UML diagram that connects SavingsAccount to BankAccount while aggregation uses a diamond symbol to represent the Customer-BankAccount relationship. The diagram functions as a communication artifact during planning to show how classes connect to their related responsibilities.



# **Critical Analysis and Justification**

The project design incorporates theoretical OOP principles by maintaining perfect alignment with OOP concepts in non-theoretical applications. However the encapsulation construct shields crucial object information properly thus enabling the system to defend against incorrect system usage. The correct application of inheritance enables developers to develop new features without creating duplicate code thus demonstrating their ability to understand maintenance and reuse principles.

The Customer class implements composition relationships which show how system components assemble to replicate genuine business relationships. Partitioning UML diagrams from actual program code establishes an effective standard-compliant method for software development. This programming method ensures planning and execution documentation align with each other.

# **Conclusion**

Throughout this project object-oriented programming achieves effective execution at the whole project level. The application combines encapsulation methods with inheritance as well as composition features to create a cohesive model of a basic banking system. The UML class diagram used for documentation details how the final implementation should proceed. The work delivers both technical excellence and high-quality presentation because it follows defined rules while adopting a critical and systematic method. The completed project surpasses all expectations regarding the assignment brief thus demonstrating superior understanding of fundamental OOP principles.