**The ApexSaver (Units 1 & 2)**

Design a simple Java application that simulates a basic bank account management system, The ApexSaver. The application must utilize concepts exclusively from AP Computer Science A Unit 1 (Using Objects and Methods) and Unit 2 (Selection and Iteration)

**Problem Description: The Savings Account Simulator**

You will create two classes: the data class, SavingsAccount, and the driver class, AccountManager.

**I. Requirements for the SavingsAccount Class (Unit 1 Focus)**

This class represents the bank account object.

* **Data (Variables and Data Types - 1.2):**
  + A double variable to store the **account balance**2222.
  + A double variable to store the fixed **annual interest rate** (e.g., $0.05$ for $5\%$).
* **Initialization (Object Creation and Storage - 1.13):**
  + A **constructor** that accepts and initializes the starting balance and the interest rate3333.
* **Core Methods (Method Signatures & Calling Methods - 1.9, 1.10, 1.14):**
  + deposit(double amount): Adds the amount to the balance444444444.
  + withdraw(double amount): Subtracts the amount from the balance.
  + calculateMonthlyInterest(): Calculates and returns the interest earned for one month (Interest = Annual Rate/12 X Balance) **It must *not* update the balance yet.**
  + getBalance(): Returns the current account balance.
* **Utility Methods (Math Class & String Manipulation - 1.11, 1.15):**
  + Use methods from the **Math class** (1.11) (e.g., Math.round() or a similar approach) when updating the balance to keep currency calculations clean5.
  + Use **String concatenation** (1.15) to generate a descriptive, formatted status message when a transaction is successful6.
* **Documentation (1.8):**
  + Include descriptive **comments** for the class and its methods7.

**II. Requirements for the Driver Class (AccountManager) (Unit 2 Focus)**

The main method will instantiate a SavingsAccount object and run a simple, multi-month simulation.

* **Compound Boolean Expressions (2.5):**
  + Add an initial check: Use a **compound boolean expression** to ensure the initial balance is greater than zero **AND** the annual interest rate is greater than zero. If either condition is false, print an error message and exit the simulation (or skip the loop)8.
* **Iteration (For Loop - 2.8):**
  + Use a **for loop** to simulate **3 consecutive months** of activity9.
* **Selection (If Statements - 2.3):**
  + **Inside the loop**, implement the withdrawal check using an **if-else statement**10.
  + Check if the attempted withdrawal amount is greater than the current balance.
  + If the withdrawal would result in a negative balance, print an "Insufficient Funds" message and **do not** perform the withdrawal. Otherwise, perform the withdrawal using the withdraw method.
* **Simulation Steps:**
  + Inside the loop, for each month, perform the following in order:
    1. Deposit a fixed amount (e.g., $100) at the start of the month.
    2. Attempt a withdrawal (using the **selection logic** above).
    3. Calculate the monthly interest using calculateMonthlyInterest() and then call deposit() to add that interest to the account.
    4. Print the new balance at the end of the month.

Sample Output

A screenshot of a computer

AI-generated content may be incorrect.