Object-Oriented Programming
Programming Assignment

Programming Assignment -- Retirement Investment Calculator

Overview

You will build a console-based Java application that simulates the growth of a retirement account (e.g., a 401(k) without employer match). The program will accept user input, validate it, perform financial calculations, use conditional logic and loops, and print a clear year-by-year growth report.

This assignment is intentionally top-down: write one console program (with helper methods) and do not design your own classes/objects beyond what Java requires. A future assignment will refactor this solution into an object-oriented design.

Learning Objectives

By completing this assignment, you will practice:

- Reading user input from the console using Scanner
- Input validation and error handling (try/catch and re-prompt loops)
- Calculations with percentages and compounding
- Conditional logic (menus, optional features, repeated runs)
- Loops (simulate growth over years; simulate compounding periods within each year)
- Output formatting for readable financial tables

Program Requirements

Program Behavior

Your program must:

1. Display a welcome message and brief instructions.
2. Prompt the user for simulation inputs (see Inputs section).
3. Validate all inputs and re-prompt until valid.
4. Run the simulation and print a year-by-year table plus a final summary.
5. Ask the user if they want to run another simulation (Y/N). If yes, repeat.

Object-Oriented Programming

Programming Assignment

Inputs (with Validation Rules)

Prompt for the following inputs and enforce the validation rules. Your program must not crash due to bad input.

Input	Type	Validation / Notes
Current Age	int	18–100 (inclusive)
Retirement Age	int	Must be > Current Age and ≤ 100
Current Balance	double	Must be ≥ 0
Annual Contribution	double	Must be ≥ 0
Annual Interest Rate (APR %)	double	User enters 7 for 7%. Must be 0–30 (inclusive)
Compounding Frequency	menu (int)	1 = Annually, 2 = Monthly, 3 = Daily (365). Must be 1, 2, or 3
Annual Contribution Increase (%)	double	User enters 3 for 3%. Must be 0–20 (inclusive). If 0, contribution stays fixed

Simulation Rules

Time Horizon

Simulate from Current Age up to (but not including) Retirement Age. Example: Current Age 25 and Retirement Age 65 means simulate 40 years (ages 25 through 64).

Contribution Timing

Contributions are deposited evenly based on compounding frequency:

- Annually: deposit once per year (one deposit).
- Monthly: deposit annualContribution / 12 each month.
- Daily: deposit annualContribution / 365 each day.

For each simulated year, use that year's annual contribution value. If Annual Contribution Increase > 0%, then at the start of each new year after the first year, update:

- annualContribution = annualContribution * (1 + increaseRate/100)

Interest / Growth

Use compound growth per period:

- r_period = (annualRate / 100) / periodsPerYear

For each period inside a year, apply this exact order:

- Add the period contribution to the balance.
- Apply interest to the resulting balance: balance = (balance) * (1 + r_period).

Object-Oriented Programming

Programming Assignment

Yearly Interest Earned

For each year, compute interest earned as:

- interestEarned = endBalance - startBalance - totalContributionsThisYear

Output Requirements

Print a year-by-year table with one row per simulated year. Round displayed money values to 2 decimals.

- Each row must include:
- Age at End of Year
- Start Balance
- Total Contributions This Year
- Interest Earned This Year
- End Balance

After the table, print a summary that includes:

- Total Contributions (all years)
- Total Interest Earned (all years)
- Ending Balance at Retirement Age

Example Console Output (Format Guide)

```
Retirement Growth Simulator
---------------------------
Current Age: 25
Retirement Age: 65
Annual Rate: 7.00%
Compounding: Monthly
Annual Contribution (Year 1): $6000.00
Annual Contribution Increase: 3.00%

Year-by-Year Projection
-------------------------------------------------------------------------------
----
Age | Start Balance     | Contributions     | Interest Earned    | End
Balance
-------------------------------------------------------------------------------
----
26  | $0.00             | $6000.00          | $215.34            | $6215.34
27  | $6215.34          | $6180.00          | $647.88            | $13043.22
...
-------------------------------------------------------------------------------
----

Summary
Total Contributions: $XXX.XX
```

Object-Oriented Programming

Programming Assignment

```
Total Interest Earned: $YYY.YY
Ending Balance at Age 65: $ZZZ.ZZ
```

Error Handling Requirements (Mandatory)

Your program must handle all of the following without crashing:

- Non-numeric input where a number is expected (use try/catch and re-prompt).
- Out-of-range numeric values (re-prompt in a loop).
- Invalid menu selections (re-prompt until the user chooses 1, 2, or 3).
- Run-again prompt input other than Y/N (re-prompt).

Required Top-Down Structure

You must implement helper methods (method names may vary) that cover the following behaviors. Do not create your own domain classes (e.g., Account, Investor, Simulation) for this assignment.

- Minimum required helpers:
- readIntInRange(Scanner sc, String prompt, int min, int max)
- readDoubleInRange(Scanner sc, String prompt, double min, double max)
- readCompoundingChoice(Scanner sc) // returns 1, 2, or 3
- runSimulation(...) // performs loops and prints the table + summary
- askRunAgain(Scanner sc) // returns true/false

Implementation Notes / Tips

- Use constants for periods per year: 1, 12, 365.
- Use System.out.printf(...) for aligned output (money: %.2f, percent: %.2f%%).
- Use while(true) loops for input validation.
- Update annual contribution once per year (not once per month/day).
- Keep your code readable: meaningful variable names and brief comments where needed.

Test Cases (You Must Verify These)

Use the following to self-check your program. Values may differ slightly due to rounding, but should be close if you follow the required order.

Test Case A (Simple)

- Current Age: 30
- Retirement Age: 32
- Current Balance: 0
- Annual Contribution: 1200
- Rate: 0%

- Compounding: Annually
- Contribution Increase: 0%

Expected behavior:

- End of age 31: $1200.00
- End of age 32 (after 2 simulated years): $2400.00
- Total interest earned: $0.00

Test Case B (Monthly Growth)

- Current Age: 25
- Retirement Age: 27
- Current Balance: 1000
- Annual Contribution: 6000
- Rate: 6%
- Compounding: Monthly
- Contribution Increase: 0%

Expected behavior:

- Interest earned each year should be positive.
- Ending balance at age 27 should be greater than $13,000.00 (1000 + 2*6000).

Test Case C (Annual Increase)

- Current Age: 40
- Retirement Age: 43
- Current Balance: 5000
- Annual Contribution: 4000
- Rate: 5%
- Compounding: Annually
- Contribution Increase: 10%

Expected behavior:

- Year 1 contribution = 4000
- Year 2 contribution = 4400
- Year 3 contribution = 4840

Submission Requirements

- Ensure that your project is stored in a GitHub repository. You will submit the link to your repository. Do not submit source files in Blackboard.

Object-Oriented Programming

Programming Assignment

- Your GitHub repository must include a Readme file that identifies your name, how to run the program, any assumptions, and which compounding options you tested.

Academic Integrity / Constraints

- You may use the Java standard library (Scanner, Math, etc.).
- Do not use external financial libraries.
- Do not copy another student's code.
- This assignment is intentionally not OOP-focused. Keep it top-down. A later assignment will focus on turning this into an OOP structure.