Matthew Hall

Pseudocode // Notes:

Classes:

CompoundInvestment // stores inputs and performs calculations

Private Data:

openAmount // initial lump sum investment

depositAmount // monthly deposit

annualInterest // interest rate in percent

numYears // years for the investment to grow

Public Setters:

SetOpenAmount(value)

SetDepositAmount(value)

SetAnnualInterest(value)

SetNumYears(value)

Public Getters:

GetOpenAmount()

GetDepositAmount()

GetAnnualInterest()

GetNumYears()

Public Method:

BuildYearlyReport(includeDeposit) // returns vector of {year, balance, interest}

Struct YearRow: //creates the structure for returning {year, balance, interest}

year

balance

interest

Function:

// We’re creating a vector to store data using the YearRow structure, for class CompoundInvestment.

// The function is named BuildYearlyReport and the parameters () receives a true/false value.

vector<YearRow>CompoundInvestment::BuildYearlyReport(bool includeDeposit)

monthlyRate = (annualInterest / 100) / 12 // creating the monthlyRate

currentBalance = openAmount

vector<YearRow> rows // rows is an empty vector

for year = 1 to numYears

yearEndInterest = 0

for month = 1 to 12

if includeDeposit == true, then

currentBalance = currentBalance + depositAmount

end if

monthlyInterest = currentBalance \* monthlyRate

currentBalance = currentBalance + monthlyInterest

yearEndInterest = yearEndInterest + monthlyInterest

end month for loop

append rows with {year, currentBalance, yearEndInterest}

end year for loop

return rows

Start Program:

Declare CompoundInvestment account // creating the object that’ll use the class functions.

Main Loop:

Clear console // clearing console for new loops.

Display first “Data Input” screen with blank input fields

“Initial investment amount: “

“Monthly deposit amount: “

“Annual interest: “

“Number of years: “

“Press any key to continue. . .” (pause) // can use the ‘Enter’ key easily instead.

Request data input for each field and set it: // Validate each input being a positive number > 0

“Enter the initial investment amount: “

Read openAmount, validate input

account.SetOpenAmount(openAmount)

“Enter the Monthly deposit amount: “

Read depositAmount, validate input

account.SetDepositAmount(depositAmount)

“Enter the Annual interest: “

Read annualInterest, validate input

account.SetAnnualInterest(annualInterest)

“Enter Number of years: “

Read numYears, validate input

account.SetNumYears(numYears)

“Press any key to continue…” (pause)

Clear console

Display second “Data Input” screen showing the four entered values

Initial investment amount: (account.GetOpenAmount)

Monthly deposit amount: (account.GetDepositAmount)

Annual interest: (account.GetAnnualInterest)

Number of years: (account.GetNumYears)

Press any key to continue. . . (pause) // can use the ‘Enter’ key easily instead.

Clear console

// Display two static screens showing the build reports

// Display has three columns/categories: Year | Year End Balance | Year End Earned Interest

reportNoDeposit = account.BuildYearlyReport(false)

reportWithDeposit = account.BuildYearlyReport(true)

// Table 1 (without / NoDeposit)

Print screen display: Balance and Interest Without Additional Monthly Deposits

Print table row: Year | Year End Balance | Year End Earned Interest

For each row in reportNoDeposit

Print row.year | row.balance | row.interest

End for loop

// Table 2 (WithDeposit)

Print screen display: Balance and Interest With Additional Monthly Deposits

Print table row: Year | Year End Balance | Year End Earned Interest // column headings

For each row in reportWithDeposit

Print row.year | row.balance | row.interest

End for loop

output “Run again? (y/n): ”

Read userChoice

if userChoice is ‘y’ or ‘Y’ then repeat main loop

else if ‘n’ or ‘N’, exit program

Input validation: if invalid input, request input again. Use ‘y’ or ‘n’ for yes or no.

Exit program