

Program 5 – Savings Calculator

For this program we will be applying our knowledge of JavaFX programming to create a savings calculator using compound interest to show the future value of a monthly savings plan assuming an annual interest rate and term in years. The user will be able to enter the monthly deposit amount, the annual interest rate, and the number of years. The calculator will compute the future value of the savings plan, the total amount deposited, and the earned interest for the entire term.

1 User Interface Specifications

The program should be written to meet the following User Interface specifications. This section corresponds to Item 3 of the “Standard Program Rubric”.

REMEMBER THIS SECTION DESCRIBES HOW THE PROGRAM SHOULD LOOK TO THE USER. IT DOES NOT DESCRIBE HOW TO WRITE THE PROGRAM.

The user interface will be implemented using JavaFX according to the screen mockup shown in Figure 1. The user interface will consist of the following components.

The project will be evaluated on the completeness, functionality, and appearance of all features.

1.1 Window and Title bar

A window like that shown in the screen mockup at right should be displayed with the title bar text set to “Savings Calculator”

1.2 Controls

There will be three groups of controls including 1) Inputs, 2) Outputs and 3) a Solve Button

1.2.1 Input Controls Group

The Input Controls Group will consist of four text input boxes with associated labels as shown including "Initial Deposit: \$", "Monthly Deposit: \$", "Interest Rate (Annual): %", and "Term (Years)".

1.2.2 Output Controls Group

The Output Controls Group will consist of three labelled output text items including: 1) “Future Value: \$” to indicate the future value of the savings plan using a monthly compound interest calculation, 2) "Total Deposits: \$" to indicate the total amount of money deposited over the term and, 3) “Earned Interest: \$” to indicate the total compounded interest earned by the savings plan over the term.

Initial Deposit: \$	Future Value: \$
<input type="text" value="1000.0"/>	39,022.30
Monthly Deposit: \$	Total Deposits: \$
<input type="text" value="100.0"/>	25,000.00
Interest Rate (Annual): %	Earned Interest: \$
<input type="text" value="4.0"/>	14,022.30
Term (Years)	
<input type="text" value="20"/>	
<input type="button" value="Solve"/>	

Figure 1. User Interface

1.2.3 Button Group

This group will contain a single button called “Solve” that will update the Output Controls Group values based on the current user inputs.

1.3 Usage

Once the user enters appropriate data for each input field, pressing the Solve button should correctly compute the Future Value of the investment at the end of the term, the Earned Interest by the end of the term, and the First-Year Interest.

2 Java Implementation

The program should be implemented in Java using the Eclipse IDE as specified below. This section corresponds to Item 4 of the “Standard Program Rubric”. This program will require a single Java file (.java) that must be uploaded to Blackboard.

2.1 Main.java

The Main class will be the only class written for this project. It should be implemented according to the following specifications to create a JavaFX User Interface Program.

2.1.1 Import Statements

To make things easier as you program, here are some import statements that can be at the top of your file. These are the ones I used in my solution, and you may use them to help get started if you wish.

```
import javafx.application.Application;
import javafx.geometry.*;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.stage.Stage;
```

2.1.2 Class Header

The header of the main class should be updated to inherit from the JavaFX Application class as described in the class programming notes and lectures by appending “extends Application” to the class header.

2.1.3 Fields

This class does not need any fields.

2.1.4 Constants

Create a private final static int constant called COMPOUND_FREQUENCY and set it to 12.

2.1.5 Static Method – main

Create the usual public static void main method. The only code inside this method should be

```
launch(args);
```

2.1.6 Private Static Method – `getFutureValue`

Create a new static method called `getFutureValue` that computes the future value of an amount deposited at a given frequency or number of times per year, using a compound interest rate calculation. Write the method header based on the following UML method notation.

```
+getFutureValue(initialDeposit : double, monthlyDeposit : double,  
interestRateAnnual : double, term : double) : double
```

Use the following PSEUDOCODE algorithm to compute the future value. Convert this pseudocode to correct Java inside your `getFutureValue` method.

Algorithm – `getFutureValue`

```
interestRateAnnualActual = interestRateAnnual / 100.0  
rateFactor = interestRateAnnualActual / COMPOUND_FREQUENCY  
rateFactor += 1.0  
futureValue = initialDeposit  
  
for (i=0; i<(term * COMPOUND_FREQUENCY); i++)  
    futureValue = (futureValue + monthlyDeposit) * rateFactor  
endFor  
return futureValue
```

2.1.7 Static Method – `start`

Create the “start” method required by the Application class.

```
+start(primaryStage : Stage) : void
```

This method will contain all code for implementing the user interface. Use the “@Override” annotation above the method header since this method overrides the start method of Application.

The table below identifies the Java classes to use for a particular UI feature.

Control	Java Class
Non-editable Text	Label (page 766)
Editable Text	TextField (page 799)
Solve Button	Button (page 792)

Vertical Groups	VBox (pp. 782-784)
Horizontal Groups	HBox (pp. 777-782)
Padding around controls	Insets (pp. 780)
Alignment	Pos.Center, Pos.Top_Center (pp. 770-771)

2.1.7.1 Input Section

Inside the start method, write the code required to complete the user input section below. Use the initialDepositInput and initialDepositGroup as examples for the other input controls.

```
//-----
// Input Section
//-----
TextField initialDepositInput = new TextField(Double.toString(1000));
VBox initialDepositGroup = new VBox(2, new Label("Initial Deposit: $"), initialDepositInput);
initialDepositGroup.setPadding(new Insets(5));

// TODO: Complete the code to create the monthlyDepositInput and monthlyDepositGroup based on example
// above
TextField monthlyDepositInput;

// TODO: Complete the code to create the interestRateInput and interestRateGroup based on example above
TextField interestRateInput;

// TODO: Complete the code to create the termInput and termGroup based on example above
TextField termInput;

// Create the input VBox
// TODO: Add monthlyDepositGroup, interestRateGroup, and termGroup to the VBox below
VBox inputGroup = new VBox(8, initialDepositGroup);
```

2.1.7.2 Output Section

Inside the start method and after the Input Section, write the code required to complete the output section below. Use the futureValueOutputLabel and futureValueGroup as examples for the other output controls.

```
//-----
// Output Section
//-----
Label futureValueOutputLabel = new Label();
VBox futureValueGroup = new VBox(2, new Label("Future Value: $"), futureValueOutputLabel);
futureValueGroup.setPadding(new Insets(5));

// TODO: Complete the code to create the totalDepositOutputLabel and totalDepositOutputGroup
// based on example above
Label totalDepositOutputLabel;

// TODO: Complete the code to create the earnedInterestOutputLabel and earnedInterestOutputGroup
// based on example above
Label earnedInterestOutputLabel;

// Create the output VBox
// TODO: Add totalDepositOutputLabel, earnedInterestOutputLabel to the VBox below
VBox outputGroup = new VBox(20, futureValueGroup);
```

2.1.7.3 Input/Output Group Section

Use an HBox to create an inputOutputGroup control and add the inputGroup and outputGroup VBox controls created above. Use the Control class .setAlignment to center the controls on the screen.

```
//-----  
// Create the Input/Output group  
//-----  
HBox inputOutputGroup; // TODO: create an HBox and add inputGroup and outputGroup to it.
```

2.1.7.4 Solve Button Section

Create a solve button and use a lambda expression to attach an event handler to it. Use the Event handler below and complete the TODO items within it.

```
//-----  
// Create the solve button and its event handler  
//-----  
// TODO: Create the solveButton using a Button control.  
// TODO: Give the button an appropriate label. (i.e. "Solve", or "Go", etc.)  
Button solveButton = null;  
  
// Handle the Solve Button  
solveButton.setOnAction((event) -> {  
  
    // Capture input from the user for the deposit amount  
    // interest rate, and term of the savings plan  
    double initialDeposit = Double.parseDouble(initialDepositInput.getText());  
    // TODO use Double.parseDouble to convert monthlyDeposit input to double  
    double monthlyDeposit;  
  
    double rate; // TODO use Double.parseDouble to convert interestRateInput input to double  
    double term; // TODO use Double.parseDouble to convert termInput input to double  
  
    // Compute the future value, earned interest and first year interest  
    double futureValue = getFutureValue(initialDeposit, monthlyDeposit, rate, term);  
    double totalDeposits = initialDeposit + (monthlyDeposit * term * COMPOUND_FREQUENCY);  
    double earnedInterest = futureValue - totalDeposits;  
  
    // Post the results to the output labels  
    futureValueOutputLabel.setText(/*TODO: convert futureValue to an appropriate String */);  
    totalDepositOutputLabel.setText(/*TODO: convert totalDeposits to an appropriate String */);  
    earnedInterestOutputLabel.setText(/*TODO: convert earnedInterest to an appropriate String */);  
});
```

2.1.7.5 Root Node, Scene, and Stage

At the end of the start method, do the following:

1. Create a Scene instance and using the outermost VBox container, a preferred width of 400, and a preferred height of 330. See pages 767 and 770.
2. Set this scene for the Stage ("primaryStage" parameter reference variable or use the parameter variable name in your start method). See page 768.
3. Set the Title for the stage. See page 769 for an example.
4. Show the window. See page 769 for an example.

```
//-----  
// Tie everything together  
//-----  
// Root node of the scene graph
```

```
// TODO: Crate a VBox and add the inputOuptGroup, and solveButton to it
VBox root; // TODO: Crate a VBox and add the inputOuptGroup, and solveButton to it
// TODO: Set any padding or alignment required for the VBox

// Create the Scene
// TODO: Create a Scene object, add the root VBox to it and initialize to 400 wide by 330 high.
Scene scene;

// TODO: Add the scene to the Stage (i.e., primaryStage)

// TODO: Set the title of the Stage (i.e., primaryStage)

// TODO: Show the Stage by calling its .show() method
```

2.2 Comments

Your programs must always include a project comment at the top of your program file. You must use the documentation comments demonstrated at the end of Section 2.11 of the Textbook to create the project comment for this program. For example.

```
/**
 * Your Name Goes Here
 * CPS141 – Summer 2023
 * The date you started the project goes here
 * Instructor: Adam Divelbiss
 * Assignment: Program10
 * Purpose: YOUR DESCRIPTION GOES HERE
 */
```

Also be sure to use a single line comment to document/describe each major section in the code.

PLEASE NOTE THAT DOCUMENTATION COMMENTS ARE REQUIRED FOR ALL METHODS. See Section 6, Item 4, and Section 7 Item 7.

Documentation comments are required for all methods and requested for all class headers.

3 Deliverables

The deliverable files will be uploaded to the assignment in Blackboard. Deliverables consist of the following .java files. DO NOT SEND THE .class FILES. I CANNOT USE THEM.

- Main.java

The due date/time is:

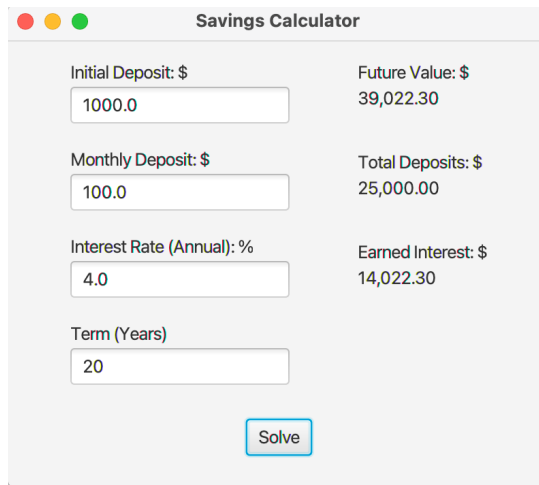
Saturday, July 1 at 11:59PM.

PROVIDED FILES:

No Java files are provided for this project.

4 Sample Output

The screenshot shows one possible UI layout with a correct calculation.



The screenshot displays a window titled "Savings Calculator" with a standard macOS-style title bar (red, yellow, and green buttons). The window contains several input fields and calculated values arranged in two columns. On the left, the input fields are: "Initial Deposit: \$" with a value of 1000.0, "Monthly Deposit: \$" with a value of 100.0, "Interest Rate (Annual): %" with a value of 4.0, and "Term (Years)" with a value of 20. On the right, the calculated values are: "Future Value: \$" at 39,022.30, "Total Deposits: \$" at 25,000.00, and "Earned Interest: \$" at 14,022.30. A "Solve" button is located at the bottom center of the window.

Input	Value
Initial Deposit: \$	1000.0
Monthly Deposit: \$	100.0
Interest Rate (Annual): %	4.0
Term (Years)	20
Future Value: \$	39,022.30
Total Deposits: \$	25,000.00
Earned Interest: \$	14,022.30

5 General Submission Policy

You may submit your code multiple times up until the due date/time. Any submission after the due date/time will be subject to the late assignment policy described in the course Syllabus. In all cases the last code you submit will be used for the grade.

6 General Requirements

All code you write for this course should meet the following general requirements:

1. All code should be your own work and not copied from another student, the internet, or any solution guide for the textbook.
2. Code from the following sources may be used or modified to complete the project:
 - a. Any code provided by me in the class notes.
 - b. Any code provided by the textbook itself (excluding any solution guides).
 - c. Any code written by you for this or another course.
3. All Java classes should have a documentation comment including, author, date, course (i.e., CPS141-Summer 2023), and a purpose or brief description statement.
4. All methods must have properly formatted and descriptive documentation comments. The only exception is methods that override well documented interface or superclass methods. Private methods must also be commented.
5. Any method with more than just a few lines of code should have additional single-line comments describing important parts of the algorithm.
6. The use of break, continue, or exit statements inside loops is not acceptable in the implementation of any class. If found points will be subtracted from Item 5 of the Programming Rubric.

NOTE: The above requirements should be followed only where applicable.

7 Program Rubric

Your grade for this assignment will be computed using the following rubric.

Item	Description	Percent of Grade
1	<u>Compilation</u> : Compiles with no syntax errors.	10
2	<u>Execution</u> : Runs with no crashes or runtime errors.	10
3	<u>External Operation</u> : Written to meet all requirements for input and output as specified or implied by the problem.	20
4	<u>Internal Operation</u> : Written to meet all requirements for internal operation as specified as specified or implied by the problem statement.	40
5	<u>Coding Style</u> : Written and formatted in accordance with best practices presented in class (indentation, class names, variable names, method names, etc.).	5
6	<u>Project Comment</u> : Contains a documentation comment with the assignment name, author, date, class (i.e., CPS141-Summer 2023), and purpose/description at the top of each file.	5
7	<u>Documentation</u> : All methods, classes, and interfaces have documentation comments . Methods that override well-documented interface method and the main method are exempt.	5
8	<u>Comments</u> : Includes sufficient other regular comments to describe important parts of the code.	5
Total		100