Assignment 1

Instructions and Grading Criteria

- This is an **individual** assessment. Please review the college's **Academic Integrity Policy** to ensure that you are completing your work in an academically honest manner.
- In addition to the required functionality, learners are expected to use the coding conventions demonstrated in class, meaningful variable naming, and clearly organized code. Comments are helpful but not required.

Academic Integrity

- This is an individual assessment.
- Permitted activities: Usage of Internet to search for syntax only; usage of course materials
- Not permitted:
  - Communication with others (both inside and outside the class)
  - Discussion of solution or approaches with others; sharing/using a "reference" from someone
  - Searching the internet for full or partial solutions
  - Sharing of resources, including links, computers, accounts
  - Usage of AI

Problem Description

Create a command-line personal finance application in Swift. Your application will allow users to manage bank accounts, track financial transactions, and generate basic reports.

Platform: Command-line Swift application (not iOS/SwiftUI)

Learning Objectives

By completing this assignment, you will demonstrate your ability to:

- Apply Swift variables, constants, and data types appropriately
- Implement control flow structures to drive application logic
- Create and use functions with parameters and return values
- Handle Optionals safely using proper unwrapping techniques
- Design meaningful structs with properties and methods
- Build class hierarchies using inheritance and polymorphism
- Create enumerations with raw and associated values
- Define and implement custom protocols
- Write clean, readable, and well-structured code

Technical Requirements

Your application **must** include all of the following components:

Required Data Structures:

1. Transaction Struct

- Properties for storing transaction details (amount, description, date, etc.)
- At least one computed property or method
- Should be Identifiable and/or Equatable as appropriate

2. Account Class Hierarchy

- Base BankAccount class with common properties and methods
- At least two derived classes (e.g., CheckingAccount, SavingsAccount)
- Demonstrate method overriding in child classes
- Show meaningful inheritance relationships

3. Enumerations

- Create at least one enum (e.g., transaction categories)

4. Protocol Implementation

- Define at least one custom protocol
- Have multiple types conform to your protocol(s)
- Demonstrate protocol methods being called

Required Programming Elements:

5. Variables and Constants

- Demonstrate proper use of let vs var
- Show understanding of when immutability is appropriate
- Use meaningful variable names following Swift conventions

6. Control Flow

Your application must include **at least 3** of the following:

- switch statements with multiple cases
- for loops (for-in or traditional)
- while or repeat-while loops
- if-else statements with complex conditions
- guard statements for early returns

7. Functions

- Create functions with parameters and return values
- Show functions that take optional parameters
- Demonstrate functions with default parameter values
- Include functions that return optionals

8. Optionals Handling

Use **at least 2** different optional handling techniques:

- if let optional binding
- guard let statements
- Nil coalescing operator (??)
- Optional chaining
- Force unwrapping (only where absolutely safe)

Application Requirements

Your personal finance tracker should provide the following functionality:

Core Features:

1. **Account Management**
    - Create different types of bank accounts
    - View account details and balances
    - Switch between multiple accounts
2. **Transaction Processing**
    - Add income and expense transactions
    - Categorize transactions appropriately
    - View transaction history
3. **Basic Reporting**
    - Display account summaries
    - Show transactions by category
    - Calculate totals and balances
4. **User Interface**
    - Menu-driven command-line interface
    - Input validation and error handling
    - Clear, user-friendly prompts and output

Program Structure Requirements

- **Logical organization** of code across multiple files
- **Clear separation** between data models and application logic
- **Consistent architecture** throughout the application
- **Proper error handling** for invalid user inputs

Code Quality Requirements

- **Consistent formatting** and indentation
- **Meaningful names** for variables, functions, and types
- **Appropriate comments** explaining complex logic
- **Swift naming conventions** (camelCase, etc.)
- **No compiler warnings** where avoidable

Sample Program Flow

Your application should present a professional interface similar to:

```
========================================
    Personal Finance Tracker v1.0
========================================

Select an option:
1. Account Management
2. Add Transaction
3. View Transactions
4. Generate Reports
5. Exit

Enter your choice (1-5): _
```

Note: The exact interface design is up to you, but it should be intuitive and professional.

Getting Started Tips

1. **Plan your data model first** - sketch out your structs, classes, and enums before coding
2. **Start simple** - implement basic functionality before adding advanced features
3. **Test incrementally** - verify each component works before moving to the next
4. **Read the rubric carefully** - ensure you're addressing each requirement
5. **Use meaningful example data** while developing and testing

Assessment Criteria

Your assignment will be evaluated using the provided rubric (15 total marks):

Criteria	Full Marks	Partial Marks	No Marks
Variables/Constants (1pt)	Correct and consistent use of let/var	Mostly correct usage	Misused or missing
Control Flow (2pts)	Multiple structures used properly	One structure used correctly	Absent or incorrect
Functions (2pts)	Well-designed functions with parameters/returns	Basic functions present	Minimal or incorrect
Optionals (1pt)	Safe unwrapping techniques used	Used but not handled properly	Not used or incorrect
Structs (2pts)	Meaningful structs with properties/methods	Basic struct(s) defined	Missing or incorrect
Classes/Inheritance (2pts)	Proper hierarchy with overrides	Class defined, limited inheritance	Missing or irrelevant
Enums (1pt)	Raw/associated values used appropriately	Enum present but basic	Not present or incorrect
Protocols (1pt)	Custom protocol with conformance	Present but incomplete	Missing or incorrect
Program Structure (2pts)	Well-organized, clear logic flow	Functional but lacks clarity	Disorganized or broken
Code Quality (1pt)	Clean formatting, proper naming, comments	Minor issues, limited comments	Poor formatting, no comments

Assignment Submission:

- Compress (.zip) your app.
- Video 1: Application Demonstration
    - Create an application demonstration screen recording **(minimum 2 mins, maximum 5 mins)** that starts by showing your program in action. Begin by stating your name and briefly introducing your application, then systematically demonstrate all core functionality including user interactions, key features, navigation flow, and error handling scenarios. Ensure your demonstration covers all assignment requirements by showing the program working correctly with various inputs and edge cases. Maintain clear screen capture with a minimum resolution of 1080p with audible narration explaining each action as you perform it, ensuring the demonstration flows logically through your application's capabilities to prove that all required programming concepts function properly in practice.
- Video 2: Code Explanation
    - Create a comprehensive code walkthrough recording **(minimum 7 mins, maximum 10 mins)** where you systematically explain your implementation by showing your actual source code. Start by introducing yourself and providing an overview of your project structure and file organization, then walk through each required component specified in the assignment: demonstrate your use of language features, explain your design decisions, show how different concepts are implemented, highlight key functions and data structures, and point out how you handled challenging aspects of the assignment. Maintain clear screen capture with a minimum resolution of 1080p with audible narration while providing clear explanations that demonstrate understanding rather than just reading code aloud.

Check Notion Docs for resources on:

- How to do a proper app demonstration and code explanation for the submission videos
- How to upload videos to YouTube and set them as "unlisted"

Submit the following:

- **Project zip file**
- **Video 1: App Demonstration Video URL**
- **Video 2: Code Explanation Video URL**
    - upload the videos to YouTube, set them as "unlisted"
    - Share the links in submission comments

Note:

- **All items listed above must be submitted for your submission to be considered complete/valid.**
- **Incomplete/Invalid submissions will not be accepted and will receive a grade of zero (0).**

Important Note:

- **NO LATE SUBMISSIONS** for assignments. Late assignment submissions will not be accepted and will receive a **grade of zero (0)**.
- Submitted assignments must run locally, ie: start up errors causing the assignment/app to fail on startup will result in a **grade of zero (0)** for the assignment.

END OF ASSESSMENT