Reflective Report: Learning Swift with TinyMart

Abstract

Swift is a powerful, open-source programming language developed by Apple, first introduced in 2014. It's designed for safety, performance, and modern syntax, making it suitable for iOS/macOS apps, server-side development, and general-purpose programming. For this assignment, I used Swift to implement the "TinyMart" product catalog system, a console-based application showcasing OOP principles like inheritance and polymorphism. Swift's characteristics—such as strong typing, optionals, and protocol-oriented design—shaped my experience as I explored a new language beyond my prior exposure to languages like Python and Java.

Approach

I approached the TinyMart assignment by first designing the class hierarchy based on the provided specifications. I sketched a UML diagram manually before coding to visualize the inheritance relationships: Product as an abstract base class, with AudioProduct, VideoProduct, and BookProduct as direct subclasses, and EBook and PaperBook inheriting from BookProduct. My implementation used Swift's class and struct features, leveraging SPM (Swift Package Manager) for a simple build process. I chose a straightforward array for the Cart's purchasedItems to manage products, avoiding complex data structures since the focus was on OOP. I relied on Xcode for coding and debugging, with the terminal for building and running via swift run. The test driver in main.swift was hardcoded with sample data to meet the output requirements.

Key Learnings

- **Classes and Inheritance**: Swift's class keyword and override mechanism for polymorphism were intuitive, similar to Java, but I learned to use final to prevent unintended subclassing.
- **Structs vs. Classes**: I discovered Swift's struct for value types (e.g., NameType) versus reference types with classes, a distinction new to me compared to Python's uniform object model.
- **Optionals**: Swift's optional types (String?) forced me to handle nil values explicitly with ?? or if let, enhancing safety but adding complexity.

- **Enums**: I used enum with associated values for GenreType and FilmRateType, a powerful feature for type-safe categorization.
- **Protocols**: While not fully utilized here, I explored how protocols could simulate abstract classes, a shift from traditional inheritance-based abstraction.

Unique or Distinctive Features

- **Optionals**: Unlike Java's null or Python's None, Swift's optionals require explicit handling, reducing runtime errors. This influenced my Cart design by ensuring purchasedItems couldn't accidentally hold nil references.
- **Protocol-Oriented Programming**: Swift favors protocols over deep inheritance hierarchies, though I stuck to classes for this assignment. It's a paradigm I'd like to explore further.
- **Concise Syntax**: Swift's omission of semicolons and use of type inference (e.g., var price: Double vs. explicit typing) streamlined coding compared to C++.
- **Property Observers**: Features like willSet and didSet (not used here) caught my eye as a unique way to monitor property changes, unlike getter/setter boilerplate in other languages.

Likes and Dislikes

- **Likes**:
    - **Readability**: Swift's clean syntax (e.g., if isCartFull() { return false }) felt elegant and reduced visual clutter.
    - **Safety**: Optionals and strong typing caught errors early, like invalid price values in the Product constructor.
    - **SPM**: The ease of swift build and swift run made compilation straightforward, unlike manual Makefile setups I've used.

- **Dislikes**:
    - **Verbosity with Optionals**: Unwrapping optionals repeatedly (e.g., for NameType fields) felt cumbersome compared to Python's duck typing.

- **Learning Curve**: Swift's mix of OOP and functional features overwhelmed me at times, especially reconciling it with C++-like expectations from the assignment.
- **Error Messages**: Some compiler errors (e.g., around protocol conformance) were cryptic, requiring Stack Overflow detours.

Any Additional Notes

One challenge was mimicking an abstract class in Swift, since it lacks a direct abstract keyword. I used a base Product class with fatal errors in unimplemented methods (e.g., getProdTypeStr), which worked but felt hacky. I was surprised by how much Swift's type system encouraged upfront design, contrasting with Python's flexibility. Debugging in Xcode was a bonus—breakpoints and variable inspection sped up fixing Cart's removeItem logic. I also noticed Swift's performance felt snappy even for a small program, hinting at its optimization under the hood.

Conclusion

Swift impressed me with its balance of safety, modernity, and expressiveness, making it a compelling choice for OOP beyond just Apple ecosystems. This experience deepened my appreciation for how language design influences code structure—Swift's optionals and enums pushed me to think more about edge cases than Java ever did. While the learning curve was steep, I enjoyed the process and see Swift as a language I'd use again, especially for projects needing robustness. This assignment reinforced that OOP's benefits (e.g., polymorphism in displayProdInfo) shine brightest when paired with a language that supports them elegantly.