CS 4402-01 Comparative Programming Languages

Instructor: Professor Rajesh Soni

Name: Ryohei Hayashi

Learning Journal 6

**Weekly Overview**

This week, I delved deeply into the foundational concepts and applications of Object-Oriented Programming (OOP). The main topics included essential elements of OOP, such as classes and objects, encapsulation, inheritance, and dynamic polymorphism. In addition, I explored methodologies for structuring classes, including abstract classes, generics, and multiple inheritance. I also gained a detailed understanding of the roles of constructors, destructors, and garbage collection mechanisms. This knowledge forms a critical foundation for practically understanding how OOP enhances flexibility and reusability in programming.

**Personal Reflections**

While revisiting OOP concepts, I particularly recognized the importance of information hiding during the design phase. For example, I learned how generics ensure type safety while enabling dynamic behaviors through polymorphism, significantly improving the maintainability and reusability of code. On the other hand, discussing the complexities of multiple inheritance and the performance impact of garbage collection emphasized the importance of thoughtful choices in OOP design. The decision of "what should be an object" was especially enlightening, as it greatly influences the system’s stability and maintainability.

Additionally, while reading Chapter 14 on the flexibility offered by object-oriented design, I was reminded of the significance of adaptability in real-world software development. The examples in this chapter highlighted how abstract classes can maximize code reusability, a principle I found directly applicable to my own projects.

**Key Learnings**

Through my research on generics, I explored how to ensure type safety while enhancing code reusability. For instance, in Java, I learned specific methods for managing lists that handle various data types safely. Additionally, by comparing C++ templates and Ada generic packages, I evaluated the strengths and weaknesses of type parameterization features in different programming languages.

Regarding open recursion, I studied its relationship with dynamic polymorphism and recursive algorithm design. For example, I examined how virtual functions in C++ enable flexibility through dynamic binding. This mechanism allows derived classes to add unique implementations while leveraging functionalities of base classes.

Furthermore, I explored the workings of garbage collection, comparing its implementations in C++ and Java. While C++ requires explicit destructors, Java simplifies memory management through automatic garbage collection. These comparisons deepened my understanding of selecting appropriate languages and methodologies based on project requirements and scale.

**Future Challenges and Goals for Next Week**

As a future challenge, I aim to design and implement programs using generics and open recursion to develop practical skills. By doing so, I hope to bridge the gap between theory and practice. Additionally, I plan to further investigate design methods that address the challenges of multiple inheritance and optimization techniques for garbage collection. Next week, I intend to deepen my understanding of these concepts and actively participate in discussions with fellow students to gain new perspectives.

Word Count: 447

References

1. Ben-Ari, M. (2006). *Understanding programming languages.* Weizman Institute of Science.