2025

CS 4402 Comparative Programming Languages

LEARNING JOURNAL 6

SANA UR REHMAN

INSTRUCTOR: JIM CASALE

WHAT I DID

This week, I explored core object-oriented programming (OOP) concepts, including classes, objects, encapsulation, inheritance, dynamic polymorphism, and generics. I began by reviewing the unit readings and video materials to understand how these ideas differ from procedural programming. Then, I completed self-quizzes and a graded quiz to reinforce the theoretical aspects.

My main task was the discussion post, where I researched *Generics* and *Open Recursion*. I synthesized ideas from academic and technical sources and provided examples in Java and C++ to show how generics promote type safety and code reuse, while open recursion supports dynamic polymorphism (Musser & Stepanov, n.d.; Hinze, 2007). Writing and revising this post helped me apply the abstract theory from the lectures into concrete programming examples.

MY REACTIONS AND FEELINGS

Initially, I found the number of OOP concepts introduced this week slightly overwhelming. Concepts such as inheritance and polymorphism were somewhat familiar, but generics and open recursion were more abstract. However, as I progressed through the readings and coding examples, the relationships among these ideas began to make sense. I particularly enjoyed the generics topic because it clarified how modern languages like Java and C++ enable strong typing while maintaining flexibility. I felt a sense of accomplishment when I could write my own generic class that compiled without type errors.

FEEDBACK AND INTERACTION

In the discussion forum, peers commented on how the examples I shared clarified their understanding of type parameterization. I also received feedback suggesting that I further elaborate on potential pitfalls of open recursion, such as invoking overridable methods from constructors. This feedback encouraged me to re-examine how lifecycle management works in object-oriented systems and why careful initialization is essential to avoid subtle runtime errors (Hinze, 2007). These interactions deepened my understanding and highlighted the importance of peer learning in this course.

CHALLENGES AND SURPRISES

One surprising realization was how similar and yet different generics and templates are across languages. Although both provide compile-time type safety, C++ templates generate specialized code, while Java generics rely on type erasure. This difference helped me appreciate the design trade-offs in programming language implementation (cppreference.com, 2025). The most challenging aspect was understanding open recursion. The idea that a base class method can dynamically call an overridden method in a subclass sometimes before the subclass is fully initialized was both fascinating and complex. This challenged me to think critically about initialization order and method binding.

WHAT I LEARNED AND HOW I CAN APPLY IT

Through this unit, I learned how encapsulation protects data integrity, inheritance promotes code reuse, and polymorphism enables flexibility. The study of abstract classes and generics showed me how to design extensible and type-safe APIs. I now recognize that constructors, destructors, and garbage collection are not just technical details but essential components of robust memory management.

In my future projects, I plan to apply these principles by writing modular, reusable code. For instance, I can use generics in Java to design utility classes that operate on multiple data types while maintaining type safety. Understanding open recursion will also help me write safer initialization code by avoiding virtual calls in constructors.

CONCLUSION

This week strengthened my understanding of how OOP principles combine abstraction, safety, and extensibility. I realized that mastering these concepts is crucial not only for programming efficiency but also for building scalable software systems. The most important takeaway for me is that true object-oriented design requires balancing flexibility with discipline something I now aim to practice consciously in my coding journey.

REFERENCES

cppreference.com. (2025). *Templates.* https://en.cppreference.com/w/cpp/language/templates

Hinze, R. (2007). *Closed and Open Recursion (PDF).* University of Oxford. https://www.cs.ox.ac.uk/people/ralf.hinze/talks/Open.pdf

Musser, D. R., & Stepanov, A. A. (n.d.). *Generic Programming (PDF).* https://stepanovpapers.com/genprog.pdf

Wordcount: 578