**Assignment 5: Team  Write Up**

Master of Science:

Information and Communications Technology

Michelle Agustin, Hamdi Ali, Kalika Browder, Jake Collins

University of Denver University College

October 12, 2025

Faculty: Nirav Shah, M.S , MBA

Director: Cathie Wilson, M.S

Dean: Bobbie Kite, PhD

**The Power of Generics and Collections in Java**

Generics and Collections have significantly changed the way we think about programming and code organization as a group. Many of us struggled with repetitive and rigid code structures before exploring these concepts. It was common to write similar logic for different data types multiple times, leading to unnecessary duplication and potential errors. Using the concept of generics has  changed our perspective, and we began to see how type flexibility and reusability made our code safer and cleaner.

Working together, we have not only developed a rhythm to how we operate, but on this assignment in particular, we have discovered that implementing generics allowed us to divide tasks efficiently. Each team member could focus on different aspects of revision for the Month class and collections while also maintaining consistent logic. This collaboration reinforced how generics help teams write reusable and predictable code, which reduces the likelihood of miscommunication or inconsistent implementations.

Generics introduced us to the concept of type abstraction, where we can design a single method or class that operates on multiple data types. This was particularly useful when we created the Month<T> class, which is able to store both integers (for month numbers) and strings (for month names) without having to create two separate implementations. Compiler type checking prevented many runtime issues that would otherwise be difficult to diagnose by catching type mismatches early. Our approach to building classes and data structures has been changed by the concept of compile-time safety.

In exploring Collections, we discovered how versatile and essential they are for managing data effectively. Java's Collections Framework provides a more dynamic and flexible way to store, retrieve, and manipulate data than basic arrays. As an example, ArrayLists are excellent for storing ordered data that changes frequently, while HashSets are ideal for storing unique elements. In our group work, one example stood out when checking for duplicate user IDs. In the beginning, using a list made lookups slow since every element had to be scanned through for each search. By switching to a HashSet, performance was dramatically improved because of its constant-time lookup capability.

In addition, we appreciated how HashMap simplifies working with key-value pairs. The use of a HashMap(Integer, String) made our code much more readable and efficient. Rather than writing nested loops to find matches, we were able to retrieve values by their keys instantly. It is especially important to achieve this efficiency when working with large applications or datasets, where performance and clarity go hand in hand.

As part of our implementation process, we also made key design choices as a team. We decided to use separate arrays and ArrayList for month numbers and names to clearly demonstrate how different collection types can work together. Throughout the implementation the team iteratively tested and refined our approach to ensure correctness and efficiency. This iterative teamwork not only helped us catch potential errors with the assignment, but also allowed us to collectively understand the practical advantages of combining generics with collections.

Together, we realized that generics and collections reinforce good software design principles as well. Rather than focusing on getting code to “work,” we now prioritize scalability and maintainability. Writing methods with generic types has inspired us to build reusable tools that can adapt to multiple contexts, an approach that mirrors real-world software development practices.

The team also has recognized that using generics and collections together reinforces consistency in our coding standards. By agreeing on how data structures should be implemented and accessed, we reduced redundancy, improved maintainability, and ensured that everyone could understand and extend the code without confusion.

The learning curve for generics can seem steep at first, especially when working with bounded types or wildcards, but the rewards are substantial in the long run. As a team, we agree that mastering these concepts has made us better programmers who are more thoughtful, efficient, and capable of writing adaptable code. In summary, we have learned how generics and collections can be combined to create robust, scalable, and professional applications. In summary, we have learned how generics and collections can be combined to create robust, scalable, and professional applications.

**References**

GeeksforGeeks. 2025. "Generics with Collections."*GeeksForGeeks*. Accessed October 6, 2025. <https://www.geeksforgeeks.org/advance-java/generics-with-collections/>

Jenkov, Jakob. 2024.  Java Generics Tutorial.  *Jenkov Tech & Media Labs.* Accessed October 6, 2025. <https://jenkov.com/tutorials/java-generics/index.html>