Assignment 7: Team Write Up

Master of Science:

Information and Communications Technology

Michelle Agustin, Hamdi Ali, Kalika Browder, Jake Collins

University of Denver University College

October 26, 2025

Faculty: Nirav Shah, M.S , MBA

Director: Cathie Wilson, M.S

Dean: Bobbie Kite, PhD

Interfaces and Abstract Classes in Java

This assignment on interfaces and abstract classes helped us strengthen our understanding of inheritance and polymorphism in Java, as well as our collaboration and problem-solving skills. The task was approached by dividing responsibilities, evaluating each other's code, and meeting to discuss design decisions to ensure consistency across implementations. Ultimately, this assignment allowed us to apply theoretical concepts to real-world code and to gain insight into when interfaces are preferred over abstract classes.

It was relatively easy to create the player interface and define the required methods to acquire the name, statistics, and sport type of a player. It was clear how to structure the NFL and NBA player classes to fulfill these method requirements due to interfaces being straightforward contracts that classes must implement. It was also manageable to unit test the interface implementation, since we could verify that the values returned by the concrete classes were what we expected.

Designing the abstract class version was the most challenging part for us. In the beginning, we needed time to distinguish conceptually between the functionality that should be implemented directly in the abstract class and the functionality that should remain abstract for subclasses to define. As a result of our discussion, we agreed that the getName method belongs in the abstract class since every player, regardless of sport, has a name. It was better to leave the getStats and getSport methods abstract to give each subclass flexibility in specifying stats and sports. The decision helped us better understand abstraction and code reusability and why an abstract class is appropriate when related child classes share behavior.

Working through the abstract class portion of the project deepened our group’s understanding of how abstraction supports cleaner and more maintainable code. We learned that collaboration and consistent communication were essential for aligning our design choices and avoiding overlap in implementation. As we compared our approaches and tested our code, we gained a clearer appreciation for the importance of planning class structures early, maintaining naming consistency, and adhering to object-oriented principles. This process helped us see how thoughtful design decisions can make large projects more manageable and cohesive.

Testing frequently was one of the things that greatly helped us during development. We were able to catch small inconsistencies early on, such as mismatched method names and minor typos, before they became larger problems. Additionally, we found that both direct and polymorphic references to the interface or abstract parent class were beneficial for testing. Throughout this process, we gained a deeper understanding of how Java supports polymorphism and how to write code that is flexible and not tightly coupled with details of implementation.

We wish we knew about this sooner so we could organize our project structure to avoid classpath issues when compiling and running tests. In the beginning, we had trouble running the program from the command line because we did not specify the correct package path. By learning how to compile with javac and run with java -cp using the package name, we were able to resolve this issue and become more comfortable with setting up Java projects.

Implementation decisions were based on clarity, reusability, and object-oriented principles. We chose meaningful class names, kept methods concise, and maintained consistency across interface and abstract class versions. In order to validate behavior and confirm design decisions, we utilized unit testing.

Ultimately, this assignment strengthened our teamwork, communication, and ability to collaborate on Java development tasks, skills essential for real-world software engineering.

References

Baeldung. 2020. “Using an Interface vs. Abstract Class in Java.” *Baeldung*. https://www.baeldung.com/java-interface-vs-abstract-class Baeldung on Kotlin

GeeksforGeeks. 2024. “Difference Between Abstract Class and Interface in Java.” *GeeksforGeeks*. https://www.geeksforgeeks.org/java/difference-between-abstract-class-and-interface-in-java/ GeeksforGeeks+1

GeeksforGeeks. 2024. “Polymorphism in Java.” *GeeksforGeeks*. https://www.geeksforgeeks.org/polymorphism-in-java/

InfoWorld. 2013. “When to use abstract classes vs. interfaces in Java.” *InfoWorld*. https://www.infoworld.com/article/2171958/when-to-use-abstract-classes-vs-interfaces-in-java.html InfoWorld