**Course**: CPSC310

**Name**: Tyler Brennan

**Date**: November 5, 2021

**Homework Assignment 2**

YES I have followed the academic honesty rules as outlined in the course syllabus.

YES I have carefully reviewed the details of the assignment.

YES I unit tested and integration tested my solutions.

YES I have documented and commented my code.

YES My code follows clean code principles.

YES My code compiles without errors.

YES My code runs without errors and behaves as expected.

**Errors:** N/A

**Part 1: Object-Oriented Concepts**

**Abstraction**

Definition: The act of generalizing a class to make it usable in many different instances.

Example: Separating GuitarSpecifications from Guitar, as GuitarSpecifications provides a level of abstraction and can be used in a different situation other than being a part of Guitar.

**Encapsulation**

Definition: The act of hiding funtionality from any user. It leads to methods called becoming a black box, as the user does not need to know how the output is generated to utilize it.

Example: The functions of GutiarSpecifications are encapsulated away from the sorting method found within InventorySystem. This is helpful because it puts the burden of comparing two GuitarSpecification objects onto the class itself, leading to easier maintenance as any adjustments to the comparison only takes place within the GuitarSpecifications class.

**Inheritance**

Definition: When a class (subclass) builds upon the functionality of another class (superclass). The subclass gains access to all of the superclasses protected and public methods and can further extend the functionality. This leads to high coupling between classes, as any adjustments to the superclass quickly ripple down to any instance of a subclass and could lead to many unintended consequences.

Example: I have no instance of inheritance within my homework.

**Polymorphism**

Definition: When two objects are subclasses of the same parent class, they can be used in place of each other when they are of type superclass when calling public methods of the superclass. Because subclasses inherit all public and protected functionality of the superclass, they will always have an implementation of each public method of the superclass.

Example: When creating an instance of a list of Guitars within InventorySystem, I create an object of type List<Guitar> before assigning it the value of LinkedList<Guitar> which is a subclass of the generic list.

**Composition**

Definition: The act of a class referencing other objects. This allows the class to access all the public methods without having to contain them. Introduces a 'has-a' relationship between two classes.

Example: One example of composition within my code is using GuitarSpecifications within Guitar. This lets Guitar to use all of GuitarSpecifications public methods.

**Delegation**

Definition: Passing responsibility of one action to another class that has the necessary methods to handle it.

Example: Search within InventorySystem delegates the comparisons between GuitarSpecification objects to the class GuitarSpecifications.

**Part 2: Functional Improvements**

In this homework assignment, I made multiple functional improvements throughout the course of working on it. One such improvement was the adjustment of the search method within InventorySystem. I adjusted the search to take in a GuitarSpecifications object to be able to search on multiple different attributes at the same time as opposed to just one. This change also affected the GuitarSpecifications class, as it had to implement a compareTo mehod to compare itself to another instance of GuitarSpecifications within the search method. Another functional improvement in my code was changing my implementation of an array to an enumerated object to store wood types, brand types, and guitar types.

**Part 3: Refactoring**

In this homework assignment, I refactored my code when separating the GuitarSpecifications class from the Guitar class. This had no impact on the functionality of the code, but it also allows my code to better adhere to object-oriented principles. Another example of refactoring within my code was adjusting the constructor of the Guitar class in order to take in an instance of a GuitarSpecifications class as opposed to taking in all the attributes associated with that class like before. This abstracts the functionality of GuitarSpecifications from the Guitar class, so that it does not need to know how GuitarSpecifications works in order to utilize it.

**Part 4: Testing**

For the testing, I wrote many different methods in the GuitarInventoryTester class

duplicateSerialTest: Tried to add two different guitars with the same serial into InventorySystem

removalIndexTest: Removed a guitar from InventorySystem by the serial number and making sure it got removed

removalGuitarSpecificationsTest: Removed a guitar from InventorySystem by the GuitarSpecifications and making sure it got removed

nullAdditionTest: Tried to add a guitar into InventorySystem with null attributes involved in GuitarSpecifications

modifyTest: Modifying aspects of a guitar and making sure that the changes occur

multipleItemSearchTest: Inserted multiple different types of guitars into InventorySystem and searched on many of them to see how the search works

negativePriceTest: Attempted to add a negatively priced guitar into the system

**Part 5: UML Class Diagram** Diagram

Description automatically generated