Development Process I – 6: Objects of the (Programming) World 2

Objectives

In addition to simple object inheritances, objects may do similar things despite not being inherently related to one another. This activity aims to:

- Utilize interfaces in creating a set of rules of what objects should be able to do.
- Differentiate when to use inheritance over interface and vice versa.
- Integrate generics when creating Java programs.

Interfaces

An interface is a template of what an object should be able to do. For example, cars, doors, laptop, and phones can lock themselves but they have different implementations. A car can lock itself like a door does, but a laptop and a phone have a different way of locking themselves for security purposes.

As such, for simplicity's sake, interfaces are utilized whenever there is some set of action(s) that a group of unrelated objects can do.

Interface Example

```
public interface LockingMechanism
{
        public void lock();
        public void unlock();
}
```

Figure 7.1 An example of an interface in Java.

As you can see from Figure 7.1, interfaces have abstract methods that define what a particular object should be able to do. This is useful in cases where we want objects to have a defines set of behavior and mitigate situations where we are unable to accomplish something during runtime because some object did not have any rules of how to do a particular action.

A good example would be comparing a person, a student, and an athlete against their own selves or one another. What does this phrase mean?

Let's say we want to compare two person objects against one another. How are we going to do that? Do we compare them by their surname? Or do we compare them by their height, weight, etc.?

Now, how about a student object. How do we compare students against one another? Is it through their grades? Or is it through the honors they have received?

Next would be how do we compare a student object to a person object? Obviously, we cannot compare them as previously implemented above. There must be a way we can compare these two objects.

Why is this important? Because there are situations in which we must prioritize a student over a person when executing some sort of operation.

The example above is very though provoking and confusing at first. Give it some time to understand or proceed to the ***Program Requirement*** of this process development to further comprehend the importance of interfaces.

Program Overview

A person contains many different fields, or characteristics, such as first name, last name, age, poor or rich, etc. There is so much variation that you could compare two people in many ways. You could probably compare them through their last name and see that Person 1 should go first before Person 2, or maybe we can see the Person 3 is older than Person 5 by comparing their ages, and so on and so forth.

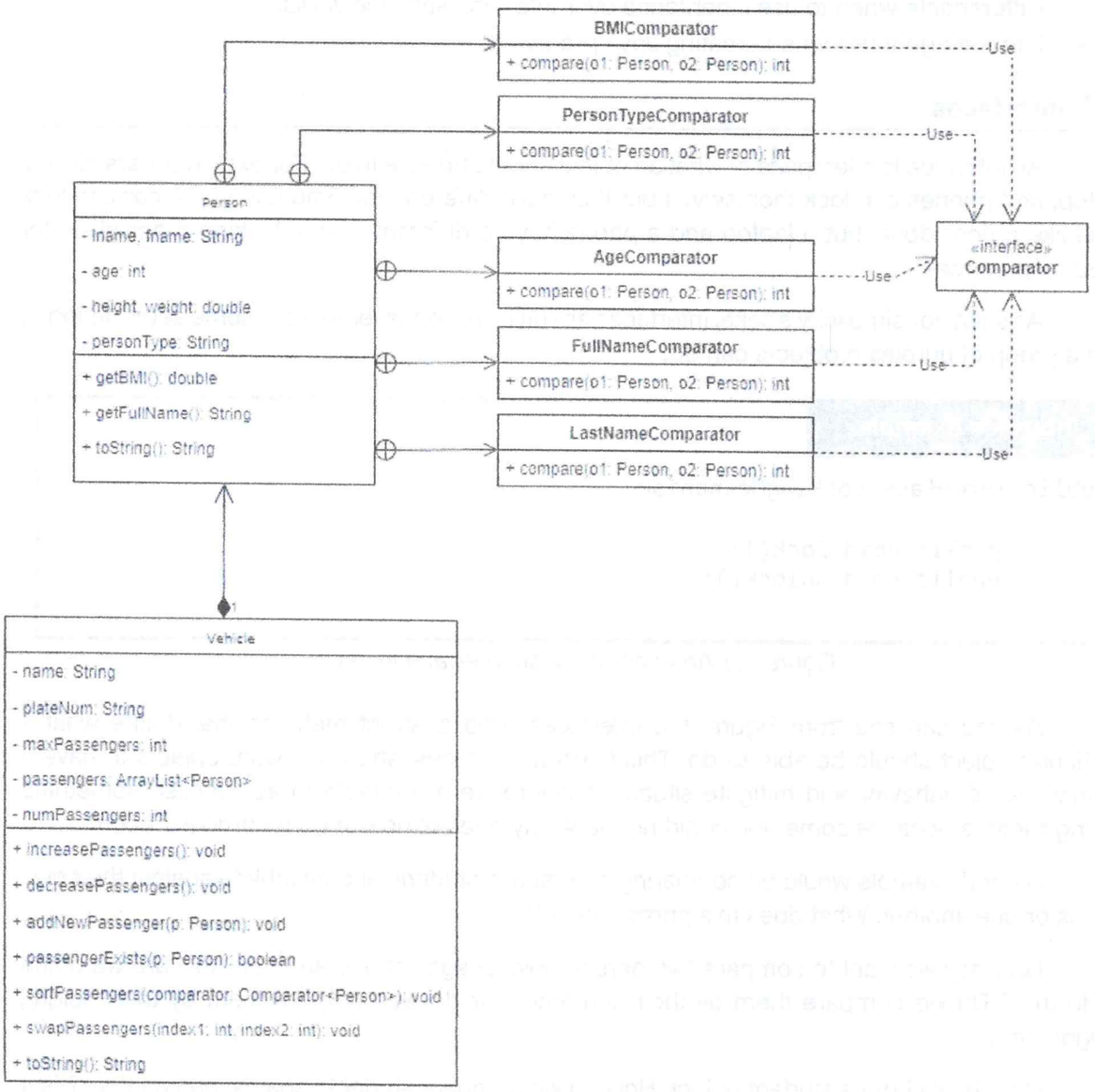

Figure 7.2 UML Class Diagram for this program.

Aside from above, passengers ride vehicles and every vehicle may contain a finite number of passengers at once. The problem now is that the driver of the vehicle wants to sort passengers in various ways. Maybe the driver wants to sort them by age, with the youngest at front, and the oldest at the back. Or maybe the drivers want to sort them alphabetically, or maybe based on their BMI. Refer to the UML Diagram in Figure 7.2

Program Requirements

Your requirement for this activity is to implement the UML class diagram above while following the behavior described in the **Program Overview** section. In addition, there must be a Launcher class on a separate file which will be used to ***test drive*** the two classes in the UML class diagram. Refer to Table 7.1 to determine how scoring will work for this activity.