Java Evolving Interfaces

Objective(s)

- Learn how to properly change the definition of interfaces

Lab setup

1. Create a new Java project
2. Add the interface `Interstellar` and its implementing classes to the project

Making changes to an interface

Ideally, you will have anticipated all the possible uses for your interface and the specification of it would be complete from the beginning.

Changing the interface will break the code of all developers who rely on it. This is easy to forget in an academic setting where you are usually working with your own code.

Suppose we need to add a new method to the `Interstellar` interface.

1. Add a new method to the `Interstellar` interface. We will use the checkStatus method shown in Figure 1.

```java
boolean checkStatus(int limit)
```

Figure 1 checkStatus method signature

   You should notice now Eclipse is throwing errors for each of the classes which implement `Interstellar`. We could fix this by adding the required implementation, but our goal is to mitigate the impact of changing this specification on the code.

   Remove the new method from the interface.

We have a couple options to add to this interface without breaking code. The first solution is to define a new interface which extends `Interstellar`

Graded Submission Task 1

1. Add a new interface to the project which extends `Interstellar`, call it `Interstellar_v2`.

Add the new method from Figure 1 to `Interstellar_v2` interface. Eclipse should not be throwing errors about this. Now it is up to the developer which interface they want to implement.

2. Change the one of the implementing classes to implement `Interstellar_v2` and implement the new method.

** STOP **

Ensure you have saved the source files in your project. Make a copy of your Eclipse project at this point (you can do this from your system's file explorer). Name the file labelling it as Task 1. You will submit this copy at the end of this lab.

The second solution we have is to define the new method as a `default` method. If done in this way, developers would not need to change their code to accommodate the new methods.

Graded Submission Task 2

1. Add a `default` method to the `Interstellar` interface.

```
default boolean chargeImpulse (int charge){
    return (charge > 9);
}
```

Figure 2 chargeImpulse method

You should notice that Eclipse does not throw any errors for this.

2. In the driver, add calls to `chargeImpulse` for each of the classes which implement `Interstellar` or `Interstellar_v2`

** STOP **

Ensure you have saved the source files in your project. Make a copy of your Eclipse project at this point (you can do this from your system's file explorer). Name the file labelling it as Task 2. You will submit this copy at the end of this lab.

Lab wrap-up

In this lab we discussed evolving interfaces. Changing the definition of an interface carries the risk of breaking the code of anyone that is using your interface. We have two options to mitigate the effect of changing interface definitions. First, we can extend the interface with a new interface. This new interface will contain the additions we want to make to the old one. This method leaves the onus to the programmers using your interface. They can choose to implement the new interface or remain with the old one. The second method we have is to add the new functionality as a default method. The default method will be available to anyone using your interface without needing to change anything.

Submission notes for graded tasks:

- Submit a zipped folder containing the Eclipse projects for each graded task.
- For each project, ensure you have included the src folder containing the .java file, the .classpath file, and the .project file.
- Ensure that each project is named appropriately to correspond with each task.
- Zip the projects together and name the zipped folder "Lab11_firstName_lastName".
- Submit the zipped file to the respective Canvas link.

Rubrics	
Task 1 (points allotted) - Compilation (2) - Interstellar and Interstellar_v2 defined. (2) - Interstellar_v2 has new method. (2) - Interstellar_v2 extends Interstellar (4)	Task 2 (points allotted) - Compilation (2) - Default method is defined in Interstellar(2) - Implementing classes compile without errors (6)