BLG332E
Object Oriented Programming

Practice Session 2

Exercise 1

Task 1: Change your working directory to `chap03/exercise1`

Vehicle
+load : double +maxLoad : double
+Vehicle(max_Load : double) +getLoad() : double +getMaxLoad() : double

Task 2: Create a class `Vehicle` that implements the UML diagram given above.

a) Include two public attributes: `load` "the current weight of the vehicle's cargo" and `maxLoad` "the vehicle's maximum cargo weight limit."
b) Include one public constructor o set the `maxLoad` attribute.
c) Include two public access methods: `getLoad` to retrieve the `load` attribute and `getMaxLoad` to retrieve the `maxLoad` attribute.

All the data is assumed to be in kilograms.

Task 3: Read the TestVehicle.cpp code. The programs gets into trouble when the last box is added to the vehicle's load because the code does not check if adding this box exceeds the `maxLoad`.

Task 4: Compile Vehicle.cpp and TestVehicle.cpp.

Task 5: Run the program. The output generated should be:

```
Creating a vehicle with a 10000kg maximum load
Add box #1 (500kg)
Add box #2 (250kg)
Add box #3 (5000kg)
Add box #4 (4000kg)
Add box #5 (300kg)
Vehicle load is 10050.0 kg
```

Exercise 2

To solve the problem from the first version, you hide the internal data (`load` and `maxLoad`) and provide a method, `addBox`, to perform the proper checking that the vehicle is not being overloaded.

Task 1: Change your working directory to `chap03/exercise2`

Vehicle
-load : double -maxLoad : double
+Vehicle(max Load : double) +getLoad() : double +getMaxLoad() : double +addBox(weight : double) : boolean

Task 2: Create a class `Vehicle` that implements the above UML diagram

a) Modify the `load` and `maxLoad` attributes to be private.
b) Add the `addBox` method. This method takes a single argument, which weight of the box in kilograms. The method must verify that adding the box will not violate the maximum load. If a

violation occurs, the box is rejected by returning the value of `false`; otherwise the weight of the box is added to the vehicle load and the methods returns `true`.

Task 3: Read the TestVehicle.cpp code. The code cannot modify the `load` attribute directly, but now must use the `addBox` method. This method returns a `true` or `false` value, which is printed to the screen.

Task 4: Compile Vehicle.cpp and TestVehicle.cpp.

Task 5: Run the program. The output generated should be:

```
Creating a vehicle with a 10000kg maximum load
Add box #1 (500kg) : true
Add box #2 (250kg) : true
Add box #3 (5000kg) : true
Add box #4 (4000kg) : true
Add box #5 (300kg) : false
Vehicle load is 9750.0 kg
```

Exercise 3

Now suppose that you were going to write some calculations that determine the wear on the vehicle's engine and frame. These calculations are easier if the weight of the load is measured in newtons.

Task 1: Change your working directory to `chap03/exercise3`

Vehicle
-load : double -maxLoad : double
+Vehicle(max Load : double) +getLoad() : double +getMaxLoad() : double +addBox(weight : double) : Boolean -newsToKilo(weight : double) : double -kiloToNews(weight : double) : double

Task 2: Create a class `Vehicle` that implements the above UML diagram.

You can copy the Vehicle.cpp and Vehicle.h files you created in Exercise 2.

a) Modify the constructor, `getLoad`, `getMaxLoad`, and `addBox` methods to use a conversion from kilograms to newtons. (1 kilo is equal to 9.8 newtons)

Task 3: Read the TestVehicle.cpp code.

Task 4: Compile Vehicle.cpp and TestVehicle.cpp.

Task 5: Run the program. The output generated should be:

```
Creating a vehicle with a 10000kg maximum load
Add box #1 (500kg) : true
Add box #2 (250kg) : true
Add box #3 (5000kg) : true
Add box #4 (4000kg) : true
Add box #5 (300kg) : false
Vehicle load is 9750.0 kg
```

You should see no change in the output of the program. This demonstrates that the private internal changes to the Exercise 3 `Vehicle` class did not change the code of the client class TestVehicle.cpp.