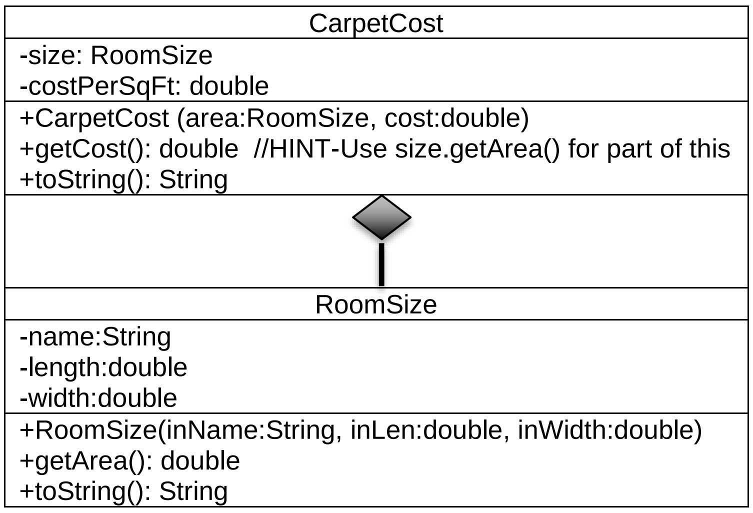
<u>Lab 8</u> (20 points + 5pts Extra Credit)
The purpose of this lab is to apply practical experience to Chapter 8 concepts. There are several learning objectives to this assignment

- Advanced Class / Object Oriented Programming design principles
- Copy Methods and Copy Constructors
- Aggregation
- Enumerations

Empire Today (14pts)

The jingle ad for Empire Today Carpet is stuck in my head, so figured that we do a program to help Empire. Empire Today Carpet has asked you to write an application that calculates the price of carpeting for rectangular rooms. To calculate the price, you multiply the area of the floor (length times width) by the price per square foot of carpet. For example, the area of floor that is 12 feet long and 15 feet wide is 180 square feet. To cover that floor with carpet that costs $7.89 per square foot would cost $1,420.20. (12×15×7.89=1,420.20)

We will use aggregation to help us by having two classes per the UML below – RoomSize and CarpetCost. **RoomSize** has three fields: name, length, and width. RoomSize has a method that returns the area of the room called getArea().
CarpetCost has two fields: size (a RoomSize object) and costPerSqFt (cost of the carpet per square foot). CarpetCost has a method, getCost(), that returns the total cost of the carpet.



```
CarpetCost
-size: RoomSize
-costPerSqFt: double
+CarpetCost (area:RoomSize, cost:double)
+getCost(): double  //HINT-Use size.getArea() for part of this
+toString(): String

RoomSize
-name:String
-length:double
-width:double
+RoomSize(inName:String, inLen:double, inWidth:double)
+getArea(): double
+toString(): String
```

After creating both classes, you will need to create an application (CarpetCostDemo *[or Main if using Replit]*). The application should ask the user to input the name of the room, length, width, and cost per square foot. The first three can be used to create a RoomSize object called temp. Then use temp and cost to create a CarpetCost object or call the RoomSize constructor along with cost. **Ensure that any instance vars that are objects via Aggregation are based on 'safe' objects through copy constructors (ie RoomSize).** ***<u>Even though the RoomSize UML does not have a copy constructor, you will need to add one since aggregation will require you to make a safe copy for the CarpetCost constructor</u>*** **(See pg 514).**

CarpetCost toString() displays the room name, length, width, cost per square foot and total cost. It is best to use the RoomSize toString() to display name, length, and width using size in the CarpetCost toString(), then displays cost per sq ft and calculate / display the cost of the carpet via the getCost() which is the product of the cost x instance file RoomSize size's area via getArea(). Both **cost** and **total cost** should be **rounded out to 2 decimal places**. Using String.format() will be helpful. **HINT – for CarpetCost toString() you can simply have size + remaining items cost and total cost. This will prevent re-writing all of the RoomSize toString() in CarpetCost.**

Continued on next page

Test the following Use Cases in CarpetCostDemo (or Main if using Replit)

1) Create a masterRoom CarpetCost object with name "Master Room", length=8.3ft, and width=10.5ft with cost of $6.97 / sq ft. Consider creating a RoomSize temp object and using temp for the CarpetCost constructor call – new CarpetCost(temp, cost).
2) Print masterRoom
3) ***Create a String var assigned to <ScannerObject, ie keyboard>.nextLine(); to clear out the carriage return from the last nextDouble().***
4) Create a livingRoom object with name "Living Room", length=12.4ft, and width=15.6ft with cost of $9.76 / sq ft. Consider using CarpetCost constructor without temp by calling the RoomSize constructor – new CarpetCost(new RoomSize(name, length, width), cost).
5) Print livingRoom

```
Enter Room Name: Master Room
Enter room length: 8.3
Enter room width: 10.5
Enter cost per square foot: 6.97

Room Name: Master Room
Length: 8.3
Width: 10.5
Area: 87.15
The cost per sq ft is: $6.97
The total cost is: $607.44

Enter Room Name: Living Room
Enter room length: 12.4
Enter room width: 15.6
Enter cost per square foot: 9.76

Room Name: Living Room
Length: 12.4
Width: 15.6
Area: 193.44
The cost per sq ft is: $9.76
The total cost is: $1,887.97
```

WorkEnum (6pts)

Create an enumeration called WorkEnum that has each workweek day starting with Monday and ending with Friday. Since enum values are considered a collection of constants, they should be defined as {MONDAY,TUESDAY, WEDNESDAY, THURSDAY, FRIDAY}. This enum can either be a separate class file or included in WorkWeek class.

Create a class called WorkWeek that has a private field var named `day` that is type `WorkEnum`. **Two constructors – a default constructor that sets the day instance field to MONDAY and another constructor that accepts a WorkEnum param to set the day instance field.**

WorkWeek
- day: WorkEnum
+ WorkWeek() //set day to WorkEnum.MONDAY + WorkWeek(day:WorkEnum) **//can use *this.day=day* since day is an instance field** + daysAreSame(inWorkWeek:WorkWeek): void + compareDays(inWorkWeek:WorkWeek): void + printDay(): void + getDay(): WorkEnum

WorkWeek also has three public void methods in addition to a getter for the field var day. These methods are daysAreSame, compareDays, and printDay. daysAreSame() and compareDays() have one WorkWeek parameter. printDay() does not have any params. **Pages 531-539 should be helpful.**

1) daysAreSame () prints out "the days are the same" if the object and param are the same or "the days are different" if the object and param are NOT the same. Hint - use day along with **`equals`** to compare the object to the param.
2) compareDays() can be found based on comparing the object ordinal value vs the param ordinal value or using compareTo(). If the object's day ordinal is less than the param day's ordinal, print "<day-object> is before <day-param>", if the object's day ordinal is the same as the param day's ordinal, prints "<day-object> is the same as <day-param>", if the object's day ordinal is more than the param day's ordinal, prints "<day-object> is after <day-param>" ***HINT – Use `.ordinal()` or `compareTo()` in your method*** **See output below.**
3) printDay() prints out "The day is " along with the day of the object that invoked (called) printDay()
4) getDay() – returns day

Create a WorkWeekDemo application (***Main if using Replit***) that does the following:

1) create a WorkWeek object called `noArgConst` that creates a default WorkWeek object via the no arg constructor.
2) create a WorkWeek object called `begin` that is set to the WorkEnum MONDAY via the constructor invocation that accepts a WorkEnum.
3) create a WorkWeek object called `mid` that is set to the WorkEnum WEDNESDAY via the constructor invocation that accepts a WorkEnum.
4) create a WorkWeek object called `weekend` that is set to WorkEnum.SATURDAY via the constructor invocation. Attempt to compile, why does the program NOT compile. **PROVIDE A SCREEN CAP SHOWING THIS. Remove the statement creating the weekend WorkWeek object**
5) test to see the relation of `noArgConst` object vs the `begin` object as an argument using daysAreSame()
6) test to see the relation of `noArgConst` object vs the `mid` object as an argument using daysAreSame()
7) test to see the relation of `noArgConst` object comes before or after the `begin` object set as the argument using compareDays().
8) test to see the relation of `begin` object comes before or after the `mid` object set as the argument using compareDays().
9) test to see the relation of `mid` object comes before or after the `begin` object set as the argument using compareDays().
10) print out mid's day value using the printDay()

<u>WorkWeekDemo Output</u>

```
The days are the same
The days are different
MONDAY is the same day as MONDAY
MONDAY comes before WEDNESDAY
WEDNESDAY comes after MONDAY
The day is Wednesday
```

Lab 8 Extra Credit (5pts)

<u>Car Instrument Sim (Prog Chall10) – pg556 (5pts) – *MUST COMPLETE CARPET and WORKWEEK FIRST*</u>

Odometer has two inst vars. (1) a FuelGauge object `gauge` and (2) an int field var `mileage` that tracks current. Mileage. Optionally, you can also add an int field var, `milesSinceAddingFuel`, that is incremented for each mile driven. `milesSinceAddingFuel` can be used in conjunction with %24 == ?? to determine when to decrement `gauge` fuelAmount by 1 gal. See the book for details. HINT – You will need to create a driver / application (class with a main()) such as OdometerDemo to run the below Use Cases

Test the following Use Cases in a driver / app that you create

1) Start with fuelLevel@6 gal, 1,000 miles, FILL UP THE TANK, and then run the tank to empty
2) Start with fuelLevel@8 gal, 999,850 miles, FILL UP THE TANK, and then run the tank to empty

When you FILL UP THE TANK, the fuel should be set to MAX_FUEL=15. Use this fuelGage object along with miles as arguments when creating the Odometer object. Don't forget to create a copy constructor for FuelGauge since you a passing a FuelGauge object into the Odometer constructor **See output file in Canvas**.

Submitting your work
For all labs you will need to provide a copy of all .java files. **No need to provide .class files. I cannot read these.** ***NOTE – For Replit, please update Main.java to another name such as TempProb.java, ProChall3.java, etc.*** In addition to your .java files, you will need to provide output files of your console. The name of the output file should match the class name and have the .txt extension such as TempProbOut.txt, ProChall3Output.txt. For GUIs such as JOptionPane, you will instead need to create screenshots. For Windows users, Snipping Tool is a great way to do this. Chromebook - Shift+Ctrl+Show Windows. Mac OS users, you can see how to take screenshots using the following url - https://support.apple.com/en-us/HT201361.