Unit H Lab Assignments

Lab H1 *Diplomas*

Define a class `Diploma` and its subclass `DiplomaWithHonors`, so that the statements

```
Diploma diploma1 = new Diploma("Murray Smith", "Gardening")
System.out.println(diploma1);
System.out.println();
Diploma diploma2 = new DiplomaWithHonors("Lisa Smith", "Psychology")
System.out.println(diploma2);
System.out.println();
```

display

```
This certifies that
Murray Smith has completed a course in Gardening

This certifies that
Lisa Smith has completed a course in Psychology
*** with honors ***
```

Make your class definitions consistent with the information-hiding principle and avoid duplication of code.

Lab H2 *Shapes*

Define an abstract class `Shape`. It should have no fields and two abstract methods, one to return the area of the shape, and one to return the perimeter.

Define two concrete classes, `Rectangle` and `Ellipse`, that inherit from `Shape`. Give them appropriate fields, constructors, accessor methods and implementations for area and perimeter. The area of an ellipse is $ab\pi$ and the perimeter of an ellipse can be approximated by $\pi\left[3(a+b)-\sqrt{(3a+b)(a+3b)}\right]$ where a and b are the lengths of the semi-major axes.

Define two more concrete classes, `Square` and `Circle` that inherit from `Rectangle` and `Ellipse`, respectively. Write appropriate constructors and override methods as necessary (consider the circumference of the circle to be the perimeter. Give `Circle` a method `diameter`

Write a test class and create a `Shape` array with at least one instance of each of the classes you created. First notice that it doesn't compile because you can't create an instance of `Shape`! OK, create the array with one instance of each of the concrete classes. Loop through the array and print the `toString`, `area` and `perimeter` of each object (notice I didn't have you write them. Go back and write them as you think is appropriate). Try to print the diameter of the circle and see what happens!

Lab H3 *Poetry*

Define an abstract class `Poem` with one field, `numlines`

```
private int numLines;          //the number of lines in the poem
```

a constructor to set `numLines` and the following methods:

```
public int getNumLines()    //returns the number of lines in the poem

public abstract int getSyllables(int k)   //returns the number of syllables
                                          //in the k-th line

public void printRhythm()      // shows the rhythm of the poem.  For example, haiku
                               // has 3 lines of 5, 7, 5 syllables, so for haiku, the
                               // printRhythm method should print
                               // ta- ta- ta- ta- ta-
                               // ta- ta- ta- ta- ta- ta- ta-
                               // ta- ta- ta- ta- ta-
```

A limerick has 5 lines of 9, 9, 6, 6, and 9 syllables. Define `Haiku` and `Limerick` as subclasses of `Poem`, and make the `printRhythm` method in each of them work without duplicating any code.

Write a test class. Create a `Poem` array with a `Haiku` and a `Limerick` object. Loop through the array and use the `printRhythm method` to print the rhythm from each of the poems.

Now create a class called `Sonnet` as a subclass of `Poem`. Sonnets are fourteen lines and usually follow a rhyming patterning, not adhering to a strict syllable rule. Implement the `Sonnet` class, overriding `printRhythm` so that it prints “no generalized syllable pattern.”

In your test class, add a `Sonnet` instance to your array and rerun your program. As long as `Sonnet` is properly implemented, you should have to make no more changes to the code to get the desired output!

Lab H4 *Rental Agency*

This lab is to be done individually, without help from peers or any person other than Mr. Saxton. You may NOT discuss the project on ANY level with ANYONE. You may not share your code orally or written in any form. You may not look at each other's computer screens. You may not help each other will compiler errors. If you are not sure what is allowed, ask me immediately.

You run a vehicle rental agency. You rent three makes of cars: Ferrari, Chevy and VW, and three kinds of trucks: a pickup, a dump truck and a semi.

The interesting characteristic of cars is their max speed in mph (Ferrari - 200, Chevy - 100, VW - 75). All cars have a maximum carrying capacity of 0.5 cubic yards.

For trucks it is their carrying capacity in cubic yards that varies (pickup - 4, dump truck - 10, semi - 30). All trucks have a maximum speed of 55 mph.

An important aspect of all vehicles is their operating cost per mile (for gas, oil, etc):

- Ferrari: $1.00
- Chevy: $.25
- VW: $.10
- pickup: $.12
- dump truck: $.80
- semi: $1.50

All vehicles also have an insurance cost. For trucks it is simply $0.01 per cubic yard of actual cargo per mile traveled. For cars it varies with vehicle type, as follows:

- Ferrari: maximum speed * miles-traveled / 1000
- Chevy: 0.05 * miles-traveled
- VW: 10% of total operating cost

You are to write an automated rental program. The customer will enter how much cargo she needs to carry (in cubic yards), the distance she needs to go (in miles), and how much time she has to get there (in hours). The program will present a list of suitable vehicles.

For each valid vehicle found for the given input, the program will display

- the specific vehicle type
- total operating cost for the trip
- total insurance cost for the trip
- total cost for the trip
- capacity (trucks only)
- speed (cars only)
- estimated trip duration

The program will inform the user if there are no suitable vehicles.

Notes

- The user interface will be up to you. It should be user friendly.
- Your program should maintain an array of vehicles that is populated with one of each type of car and truck. Your program should service multiple users; i.e., when finished with one user, it should request input for the next.
- All users should be accommodated from the same array of vehicles; assume there is an infinite supply of all vehicle types.
- Your vehicles should have a method that returns, as a string, all information pertaining to a trip. Your program should loop through the array of vehicles and, for each one valid for the trip, print the information returned by the method.

Turn In

- The first step is your design. You must draw a picture of your program's design, using good OOP principles. Each class should include the fields, methods and constants, and whether it is concrete or abstract.
- The second step is to implement the design. *Please **zip** your .java files and upload to Haiku.*

Lab H5 *Comparable*

The interface `Comparable` is part of the `java.lang` library.

```
public interface Comparable
{
  public int compareTo(Object other);
}
```

Write two classes that implement `Comparable`: `PlayingCard` and `Person`. First, design the two classes. Determine the constructors, fields and methods. The design and functionality of the classes is up to you.

Write a class `Sorter`.

1) It should have a static method, `printInOrder` that takes two `Comparable` objects and prints the objects' `String` representations in ascending order.

2) It should have a `main` method that
   (a) Creates two `Card` objects and uses `printInOrder` to print them in order
   (b) Creates two `Person` objects and uses `printInOrder` to print them in order
   (c) Creates two `String` objects and uses `printInOrder` to print them in order. Does this one work? Explain.

Lab H6 *Same Distance*

Write the following interface:

```
public interface Place
{
    //Precondition: other is not null
    //Postcondition: returns the distance from this object to other
    public int distance(Place other);
}
```

Write two classes that implement `Place`.

1) Class `Point` represents a Cartesian coordinate pair. It has two fields, `x` and `y`.
2) Class `Time` represents a moment on a digital clock. It has three fields, `hour`, `minute`, `second`.

Write a client class to test your classes. Your class should have the following method:

```
//returns true if p1 is equidistant from p2 and p3
public static boolean sameDistance(Place p1, Place p2, Place p3)
{
    return p1.distance(p2) == p1.distance(p3);
}
```

Write a `main` method that creates several `Point` objects and calls `sameDistance` for different triplets. Then repeat, using instances of `Time`.

H7 *Employees*

Create an abstract class called `Employee` with an interface as follows:

```
String getName()
String getAddress()
double weeklySalary()
```

Decide which methods should be `abstract`. Create two classes called `HourlyEmployee` and `SalariedEmployee` that inherit from `Employee`. The constructors for the two classes are:

```
//numHours is the hours worked per week
HourlyEmployee(String name, String address, int numHours, double HourlyWage)

//yearlySalary is assumed to be for 52 weeks
SalariedEmployee(String name, String address, double yearlySalary)
```

Add one method to `HourlyEmployee` that would not likely be in the `Employee` class. Add one method to `SalariedEmployee` that would not likely be in the `Employee` class.

Write an interface called `Taxable` with methods as follows:

```
String getTaxID()       //use the social security number
double getYearlyTax()   //Let's say it's a flat tax of 20% of yearly earnings
```

Update your classes so that they implement `Taxable` in an appropriate way.

Questions

Consider:

```
public void exer1(Taxable t)
{
  <missing code>;
}
```

Which statement(s) could legally replace `<missing code>` ?

(a) `t.getTaxID()`
(b) `t.getYearlyTax()`
(c) `t.weeklySalary()`

Consider:

```
public void exer2(Employee e)
{
  <missing code>;
}
```

Which statement(s) could legally replace `<missing code>` ?

(a) `e.weeklySalary()`
(b) `e.getYearlyTax()`
(c) `e.getHourlyWage()`