Lab5 Solutions

Problem 1.A

In an earlier lesson, it was mentioned that Java's `ArrayList` implements 6 interfaces and extends one class. What are they?

Solution:

- **Class:** AbstractList
- **Interfaces:** Serializable, Clonable, Iterable<E>, Collection<E>, List<E>, RandomAccess

Problem 1.B-D

Parts B – D of this Problem refer to code in package `lesson7.labs.prob1`, in which you are trying to remove duplicates from a List and then test that your output is correct. All three attempts to solve this problem are incorrect in some way (when you run the code, output message indicates that the procedure fails). Explain, in each case, what is wrong with the solution. Place each of your answers in a text file in the relevant package.

Solution:

Part B: The method equals from the class Object was not overridden.

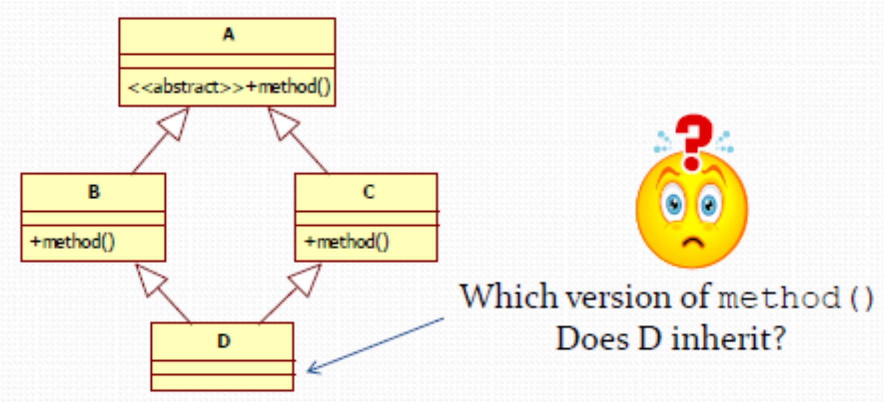
Part C: HashMap needs method hash code overridden from Object, and it was missing in the Employee class.

Part D: The following code **`tracker.get(e).setVisited(true);`** was breaking the logic of the method **`removeDuplicates`** because it was changing the value visited in the object in the **`HashMap`**, and this value is used in **`equals`** method and **`hashCode`**.

Problem 1.E

Lesson 5 introduced the Diamond Problem that must be handled by any language that supports multiple inheritance. Java SE 8 now supports "behavioral" multiple inheritance (but not "data" multiple inheritance). Explain how features of Java 8 handle the Diamond Problem by considering two scenarios:

i. ***When the type D is a class and A, B, C are interfaces***
ii. ***When the type D is an interface also***



Solution:

Should override the inherited method **or** create an abstract method with the same signature.

Problem 2

The Lesson 5 Demo in `lesson5.lecture.intfaces2` shows how to polymorphically compute the average perimeter of a list of geometric objects by requiring each to implement the ClosedCurve interface. Notice that when a closed curve happens to be a polygon, computing the perimeter is especially easy – you just add up the lengths of the sides.

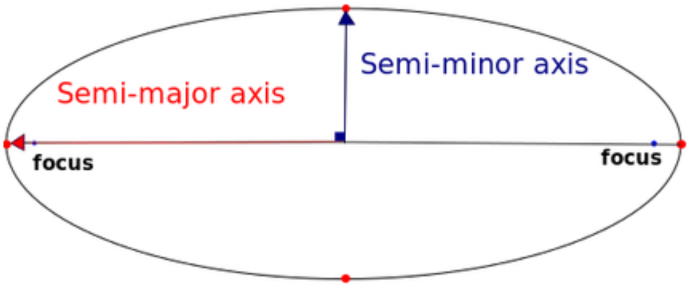
If we create an interface Polygon having method double[] getSides() (which will return the length of each side of the polygon in an array), we could replace ClosedCurve in our example with an interface Polygon – if we didn't have to take into account the computation of the perimeter of non-polygons, like Circles. In this problem, you will find a way to make use of both ClosedCurve and Polygon.

Startup code for this problem is in the package lesson7.labs.prob2; it contains classes Circle and

Rectangle, the interface ClosedCurve, and a DataMiner class that contains a main method that loads a few of these geometric objects into an array and computes the averagePerimeter. Begin by creating a new Polygon interface. Then think of a way to make use of both ClosedCurve and Polygon so that, when computePerimeter is called on one of the geometric objects that implements the Polygon interface (like Rectangle), the side lengths are added up, but when the object is not a polygon, a different computation of perimeter is done (as in the case of a Circle). Hint. Create a default method in Polygon. The idea is that you try to use the generic computation for computing perimeter, available in Polygon, whenever it is possible.

Expand your code by adding two new ClosedCurves to your package: EquilateralTriangle and Ellipse (an equilateral triangle is a triangle in which all side lengths are equal). Modify DataMiner so that it includes in the objects list instances of these new classes.

Hint. The perimeter (or circumference) of an ellipse is 4aE where a is the length of the semi-major axis and E is the value of the elliptic integral evaluated at the ellipse's eccentricity. You do not need to know these technical concepts; just include a and E as instance variables in your class, of type double, and include them as arguments to the Ellipse constructor.



Coding Solution:
Attached in eclipse file name ***'MPP-Lab7'*** inside package `'lesson7.labs.prob2'`.

Problem 3

In the lesson7.labs.prob3 package, there is a class called ForEachExample that specifies, in its main method, a list of Strings. Use the Java 8 forEach method within the main method to print out the list so that all Strings are in upper case. To do this, you will need to define your own implementation of the Consumer interface.

Coding Solution:
Attached in eclipse file name ***'MPP-Lab7'*** inside package **`'lesson7.labs.prob3'`**.

Problem 4

Rework the Duck Application of Lab 5, Problem 1 so that Flyable and Quackable interfaces are used after all, but now use Java 8 interfaces. Rewrite your code with this approach. Hint. Recall that the reason why we chose not to use interfaces to solve the Duck problem was that it would require us to provide the same implementation of methods like quack() and fly() whenever a class implements one of the interfaces. How does the use of default methods avoid this problem of code redundancy?

Coding Solution:
Attached in eclipse file name ***'MPP-Lab7'*** inside package **`'lesson7.labs.prob4'`**.