Comp 2140 - Data Structures

Lab 1 - Object Oriented Programming in Java.

“When I went to school (University of Manitoba), they (Instructor of the Data Structure course) asked me what I wanted to be when I grew up (graduate from the university). I wrote down ‘happy’. They told me I didn’t understand the assignment (the lab), and I told them (anonymously, on Piazza) they didn’t understand life…”

John Lennon (almost)

The **objective** of this lab is to review object-oriented programming design principles in Java.

Please take the following steps in this lab.

- Download the source-file from . There are multiple .java files each containing a public class. These classes form a class hierarchy of two-dimensional geometric shapes (typically found in drawing programs).

- Sketch the tree structure for the Shape class hierarchy. Put an asterisk (*) beside the abstract classes. Recall that abstract classes are those which require some of their methods be implemented in concrete (non-abstract) class extensions.

<table>
<thead>
<tr>
<th>Answer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class <strong>Shape</strong> has <strong>PositionalShape</strong> and <strong>MyRectangle</strong> as its children. <strong>PositionalShape</strong> has <strong>MyCircle</strong> as its child. <strong>MyCircle</strong> has indeed <strong>MyDonut</strong> as its child.</td>
</tr>
</tbody>
</table>

- One of the advantages of a class hierarchy is that common code can be placed in the upper levels of the hierarchy, which avoids repeating code in multiple classes. As an example of this, in the process of constructing a new Donut object, list all the classes in which a constructor is invoked.

<table>
<thead>
<tr>
<th>Answer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Besides the <strong>donut</strong> constructor, those of <strong>MyCircle</strong>, <strong>PositionalShape</strong>, and <strong>Shape</strong> are called through <strong>super</strong> operator.</td>
</tr>
</tbody>
</table>

---

1Let’s be ignorant and spell it as ‘donut’ instead of a ‘doughnut’.
• Note that the Shape hierarchy contains two abstract methods (getArea in class Shape and getBoundingBox in PositionalShape). Why can’t we just include concrete (non-abstract) methods in MyRectangle, MyCircle, and MyDonut classes?

**Answer:**
MyCircle, MyRectangle, and MyDonut are not completely unrelated; as shapes, they share having an ‘area’ and as positional shapes they have a position and bounding box. If we overlook these shared elements and include non-abstract methods in these classes, we cannot exploit the commonalities between them. As an example, we can compare the areas of these shapes (as we do in method printSmallerShapes), and we can compare their positions (as we do in printLeftShapes) because their differences in concrete classes are ignored and their commonalities in the abstract classes are exploited.

• Note that a ‘donut’ is formed by a small circle (a hole) inside a larger one. In this implementation, the donut is defined through the larger circle as an extension of the class MyCircle, while the smaller one is a field (attribute) in the extended class. Explain why this implementation is preferred over the one which defines the donut through the smaller circle and uses the larger circle as an attribute.

**Answer:**
The alternative explanation requires unnecessary overwrite of the getBoundingBox method. This is because the bounding box of the donut is the same as the larger circle and not the smaller one.

• MyRectangle class is an extensions of Shape. Change the hierarchy and the code so that a rectangle becomes ‘positional’, i.e., it has a position in the plane (like MyCircle does). Note that, since getBoundingBox is defined in the abstract class PositionalShape, it should be defined in the new MyRectangle class.

• In file lab1.java, a test program for the shape hierarchy. In the main function, 10 instances of the PositionalShape class are initiated but commented. Note that you can’t instantiate a PositionalShape object (because it is an abstract class), but you can use a PositionalShape variable to refer to any concrete object in the Shape hierarchy. Ensuring that MyRectangle is updated, uncomment the instantiation and run the program (no output at this point; just ensure it compiles and runs).

A function named printSmallerShapes is included in the lab1 class. Uncomment the call to this function in the main method and run your code. The information (strings) associated with objects smaller than the first shape in the array allShapes (which is c1) should be printed. Show your working program to the lab TA.

**The space for the TA to comment:**
**Answer:**

• Another function, named printLeftShapes, is called which is aimed to print the positional shapes on the left of its first parameter. Complete the implementation of this function (currently empty), and run your code to report points on the left of the first object in the array (which is c1). Show your working program to the lab TA.

**The space for the TA to comment:**
**Answer:**