3 Object-Oriented Programming (rkh23)

(*a*) (*i*) Explain the practical motivation for introducing `default` methods within interfaces in Java. [2 marks]

(*ii*) Compare and contrast a Java interface containing `default` methods with an abstract class. Give a scenario where an abstract class would still be the preferred design choice. [4 marks]

(*b*) You have an existing class hierarchy consisting of an interface `Shape` and two concrete implementations: `Circle` (atomic) and `Group` (composite). A `Group` contains other `Shape`s in a `List<Shape>`. You wish to allow arbitrary operations to be defined **outside** of the `Shape` classes, such that new operations can be added without modifying the `Shape` source code later.

(*i*) Define the additional methods necessary in the `Shape` interface and the `Circle` and `Group` classes to support this new mechanism via the generic interface:

```
public interface ShapeOp<R> {
  R opShape(ShapeCircle s);
  R opGroup(Group g);
}
```

Your answer should define only the additional methods and not the original fields and methods, which you can assume are complete. [5 marks]

(*ii*) Explain why `opCircle` and `opGroup` should not have default implementations. [3 marks]

(*iii*) Using the mechanism you created in part (*b*)(*i*), implement a concrete class `BlueFilter` that implements `ShapeOp<List<Circle>>`. This class must traverse the shape hierarchy and return an immutable list containing only the `Circle` objects that return `Color.BLUE` when their `getColour()` method is called. [5 marks]

(*iv*) Explain why the `ShapeOp` interface cannot be instantiated using a Java Lambda expression (e.g., `x -> ...`). [1 mark]