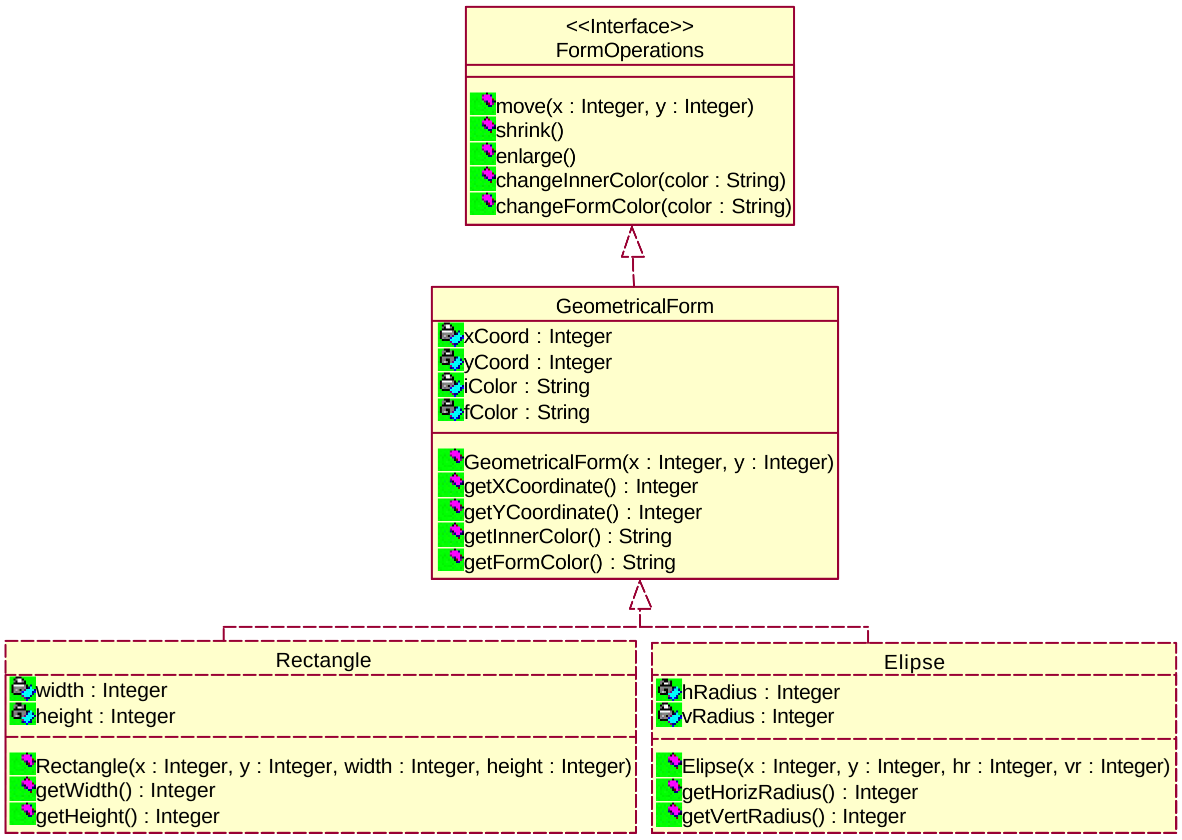
Interfaces

Problem 1

Let us consider the following hierarchy of classes and interface.



<<Interface>>
FormOperations
move(x : Integer, y : Integer)
shrink()
enlarge()
changeInnerColor(color : String)
changeFormColor(color : String)
GeometricalForm
xCoord : Integer
yCoord : Integer
iColor : String
fColor : String
GeometricalForm(x : Integer, y : Integer)
getXCoordinate() : Integer
getYCoordinate() : Integer
getInnerColor() : String
getFormColor() : String
Rectangle
width : Integer
height : Integer
Rectangle(x : Integer, y : Integer, width : Integer, height : Integer)
getWidth() : Integer
getHeight() : Integer
Elipse
hRadius : Integer
vRadius : Integer
Elipse(x : Integer, y : Integer, hr : Integer, vr : Integer)
getHorizRadius() : Integer
getVertRadius() : Integer

Requirements:

1. Implement the above class diagram.

2. Write a test class that creates an objects array with objects from the classes GeometricalForm, Rectangle and Ellipse call the methods declared in the CollectionOperations interface and the methods defined in classes.

Interfaces

Problem 2

A car dealer sales domestic and foreign vehicles like trucks, buses, and minibuses. To compute a vehicle's price the dealer has to add some requested taxes to the vehicle's base price. The taxes the dealer's sales management system should consider are the followings:

a) VAT: 19% from vehicle's price

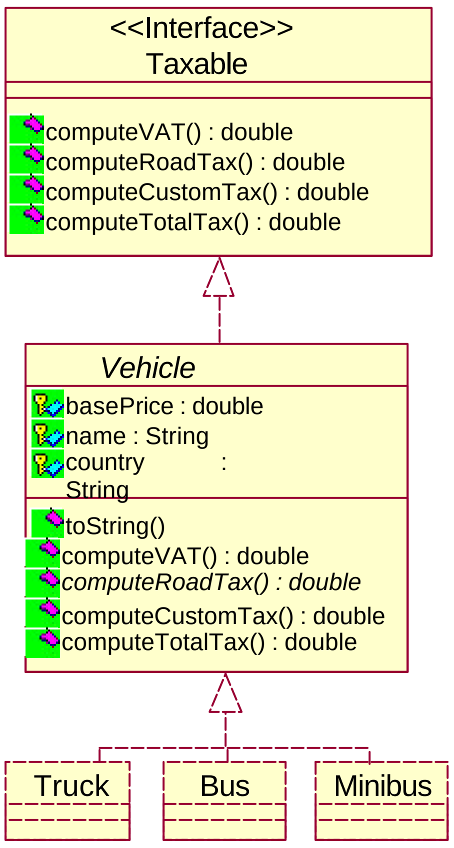
b) Road tax: 3% for minibuses, 4% for buses and 5% for trucks

c) The Customs tax: 10% for the vehicles made abroad.

In order to reuse the code, the system's model introduces the class Vehicle. It abstracts three classes: Truck, Bus, and Minibus by factorizing the common data structure and behavior of them.

The system behavior regarding tax application is modeled as a four methods interface `Taxable`.

The class diagram of the problem is presented in the following figure:



<<Interface>>
Taxable
computeVAT() : double
computeRoadTax() : double
computeCustomTax() : double
computeTotalTax() : double
Vehicle
basePrice : double
name : String
country : String
toString()
computeVAT() : double
computeRoadTax() : double
computeCustomTax() : double
computeTotalTax() : double
Truck
Bus
Minibus

Vehicle implements three methods and offers the fourth, `computeRoadTax()`, as an abstract method. This way it gives up the method implementation to its subclasses. From this reason, the class Vehicle is an abstract class.

Requirement. Implement the above class diagram and write a test class that prints for at least 3 vehicles the following information:

- vehicle's type: "Truck", "Bus" or "Minibus"
- manufacture's name
- manufacture country
- base price of the vehicle
- the total sum of taxes that have to be paid

Problem 3

Using the objects created in the previous problem, sort in ascending order after the total sum of taxes and when these values are equals, then sort the objects in descending order of the price of the vehicle. Hint: You may use the interface `Comparable` that contains the method `compareTo(o: Object)`.