Computer Science Division
Department of Mathematics
Faculty of Science

Level 3- Fall Semester
Course Code: Comp 301
Date: November 4, 2024

Sheet 5

Objective: upon successful completion of this sheet, students should be able to effectively apply one of the fundamental OOP principles: inheritance. This encompasses concepts such as method overriding, method overloading, and polymorphism.

1. Design the "`Point2D`" class that contains the following data members and methods.

   Data members: two private instance variables of type double to represent an $x-$ and a $y-$coordinate.

   Public methods: suitable constructor(s), setters, getters, `setXY()` to set both coordinates, `getXY()` to retrieve both coordinates as an array, `distance(Point2D)` to calculate the Euclidean distance from this point to the specified point, and `distance(double x, double y)` to calculate the Euclidean distance from this point to the specified coordinates.

   The `equals()` and `toString()` methods should be overridden. The former is used for comparing two points based on their $x-$ and $y-$coordinates, while the latter is used to display the point instance in the following format: (x, y).

   - Design also the "`Point3D`" class, which is a subclass of the `Point2D` class and contains the following.

   Data members: private instance variable of type double to represent a $z-$coordinate.

   Public methods: suitable constructor(s), setters, getters, `setXYZ()` to set all the three coordinates, `getXYZ()` to retrieve the three coordinates as an array, `distance(Point3D)` to calculate the Euclidean distance from this point to the specified point, and `distance(double x, double y, double z)` to calculate the Euclidean distance from this point to the specified coordinates.

The `equals()`and `toString()` methods should also be overridden.

- Test the classes with an array of `5` 2D points, two of them are created as 3D points. Then calculate the distances between the 2D points as well as between the 3D points. Check whether the 2D points are equal. Repeat the same checking process for the 3D points.

2. Design the "`Figure`" class which contains the following.
   Data members: two private instance variables, one of type `Point2D` to represent the starting point of the figure, and the other of type String to represent the color of the figure.
   Public methods: suitable constructor(s), setters, getters, and a `display()` method to show the figure's information.

   - Design also the "`ClosedFigure`" class which is a subclass of the `Figure` class and contains the following.

   Data members: three instance variables, two of type float to represent the width and height of the figure, and the third of type boolean to indicate whether the figure is filled with the specified color.
   Methods: suitable constructor(s), setters, getters, and an `area()` method. The `display()` method should be overridden.

   - Design also the "`Rectangle`" class which is a subclass of the `ClosedFigure` class and contains the following.

   Methods:
   - suitable constructor(s), one of them takes an ending point of type `Point2D` as an argument. Then, calculate the width and height of the rectangle accordingly.
   - The `area()` and `display()` methods should be overridden. The `display()` method shows the rectangle's information, including its area.

- Test the classes by defining objects from the three class types and checking the methods for each one of them. Declare objects from the `Figure` class, where some of these objects are created as `ClosedFigure` objects and others as `Rectangle` objects. Check the methods for these different types of objects.

3. Design the "`Vehicle`" class that contains the following data members and methods.

Data members: two instance variables of type String to represent the name and color of a vehicle, a price of type double, a numberOf Cylinders of type integer, and a numberOfVehicles of type integer to keep track of all the vehicles created.

Methods: suitable constructor(s), setters, getters, a `display()` method to show the vehicle's information, and a `getPrice()` method to calculate the total price of the vehicle after the taxes.

- Design the "`Car`" class which is a subclass of the `Vehicle` class and contains the following.

Data members: the number of passengers of type integer. The class also contains a numberOfVehicles of type integer to keep track of all the cars created.

Methods: suitable constructor(s), setters, getters, and override both the `display()` and `getPrice()` methods.

Taxes for cars are calculated according to the following rule:

$$\begin{cases} 15\%\ of\ \boldsymbol{price} & \boldsymbol{numberOfCylinders} \leq 4 \\ 30\%\ of\ \boldsymbol{price} & otherwise \end{cases}$$

- Design also the "`Truck`" class which is a subclass of the `Vehicle` class and contains the following.

Data members: the load capacity in kilograms of type integer. The class also contains a numberOfVehicles of type integer to keep track of all the trucks created.

Methods: suitable constructor(s), setters, getters, and override both the `display()` and `getPrice()` methods.

Taxes for trucks are calculated according to the following rule:

$$\begin{cases} 10\%\ of\ \boldsymbol{price} & \boldsymbol{numberOfCylinders} \leq 6\ and\ \boldsymbol{capacity} < 3500\ kg \\ 20\%\ of\ \boldsymbol{price} & otherwise \end{cases}$$

- Test the classes by creating an array of vehicles, some of which are created as `Car` objects and others as `Truck` objects. Find the average price of the total prices for each type of vehicle.