Practice Problem Set – Inheritance, Association, Polymorphism

1. Recall from class lectures the `PhoneBook` class implemented by extending class `list`. The `PhoneBook` class inherited all methods of the list class including the `pop()` and `remove()` methods. Derive a class from `PhoneBook` called `ProtectedPhoneBook` which does not allow any deletions from the `ProtectedPhoneBook`, instead if the user calls `pop()` or `remove()` methods it prints an appropriate message.

2. Imagine a publishing company that markets both book and audiocassette versions of its works. Create a class `Publication` that stores the `title` (a string) and `price` (type float) of a publication. From this class derive two classes: `Book`, which adds a `pageCount` (type int), and `Tape`, which adds a `playingTime` in minutes (type float). Each of these three classes should have a `getdata()` function to get its data from the user at the keyboard, and a `putdata()` function to display its data. Write a main program to test the book and tape classes by creating instances of them, asking the user to fill in data with `getdata()`, and then displaying the data with `putdata()`.

3. Start with the `Publication`, `Book`, and `Tape` classes of problem 2. Add a base class `Sales` that holds a list of three floats so that it can record the dollar sales of a particular publication for the last three months. Include a `getdata()` function to get three sales amounts from the user, and a `putdata()` function to display the sales figures. Alter the `Book` and `Tape` classes so they are derived from both `Publication` and `Sales`. An object of class `Book` or `Tape` should input and output sales data along with its other data. Write a main program to create a `Book` object and a `Tape` object and exercise their input/output capabilities.

4. Assume that the publisher in problem 3 decides to add a third way to distribute books: on computer disk, for those who like to do their reading on their laptop. Add a disk class that, like `Book` and `Tape`, is derived from `Publication`. The disk class should incorporate the same member functions as the other classes. The data item unique to this class is the `diskType`: either CD or DVD. The user could select the appropriate type by typing c or d. Use a dictionary to store this relationship table between c and CD, and d and DVD. When user enters c or d, the corresponding value is picked from this dictionary and assigned to `diskType`.

5. Repeat problem 3, replacing multiple inheritance by aggregation or composition. Now `Book` or Tape should inherit from `Publication` only; Sales should be attached as a part to them.

6. Define a class called `Date` which stores a date (day, moth, year). Define appropriate setters and getters and override the `__str__` method to print the date in the format dd-mm-yy.

7. Add a member variable `format` to class `Date` in problem 6. This variable stores numbers that represents various date formats; for example: if `format=1`, the date should be printed in dd-mm-yy, if `format=2`, the date is printed in mm/dd/yy, if `format=3` date is printed on mm_name dd, yy. The user of class `Date` should be able to set a specific format for date printing by the `__str__` method.

8. Attach class `Date` developed in problem 7 to the `Publication` class of problem 3 by any relationship you find appropriate. The `Date` class will help store the date of publication of a `Book` or a `Tape`.

9. To implementation of problem 8, add class variables at appropriate places to keep count of all the books and tapes instantiated using this hierarchy.

10. For the implementation of class `Publication` done in problem 9, answer the following questions:
    a. Make class diagram for this implementation.
    b. Identify all class, instance and local variables.
    c. If an object of class `Publication` is instantiated, how many namespaces (including class blueprints) will be created? Trying drawing a map.

11. Write code to develop a class named Vehicles having three private attributes `noOfWheels`, `color` and `modelNo`; define related constructor and public getter/setter methods. Use appropriate types for all attributes, method parameters and return values wherever needed. Instantiate necessary objects to test the functionality of this class.

12. Write code to develop a class `Engine` having private attributes `engineNo` and `dateOfManufacture`, along with appropriate constructor and public getter/setter methods. Associate this class to the class `Vehicles` create in problem 6 as aggregation making the necessary changes. Decide appropriate types for all attributes, method parameters and return values for this new class.

13. Repeat problem 7, changes the type of association to composition.

14. Add to problem 7, implementation of `__str__` method to display the following text when the object is printed:
    Class Vehicle
    Model no.: <modelNo>
    No of wheels: <noOfWheels>
    Engine no: <engineNo>
    Date of manufacture (engine): <dateOfManufacture>

15. Write code to create a class that imitates part of the functionality of the basic data type int. Call the class `Int` (note different capitalization). The only data in this class is an int variable. Include methods to initialize an `Int` to any value (0 being default), to change it anytime and to display it, and to add two `Int` values using the + operator. Write a program that exercises this class by creating one uninitialized and two initialized `Int` values, adding the two initialized values and placing the response in the uninitialized value, and then displaying this result.

16. Write code to create a class called `Time` that has separate member data for hours, minutes, and seconds. Make constructor to initialize these attributes, with 0 being the default value. Override `__str__` method to display time in 11:59:59 format when the object is printed. Add another method `addTime` which takes one argument of `Time` type and add this time to the current time of the *self* object through + operator. Instantiate two objects `t1` and `t2` to any arbitrary values, display both the objects, add `t2` to `t1` and display the result.

17. Define a class called `BasicSalary` having one instance variable `basic` to store basic salary of an employee. Add appropriate `__init__` method and another method `annualBasicSalary` which calculates and returns the sum of basic salaries for twelve months. Now define another class called `Employee` which instantiates an object of `BasicSalary` type in its `__init__` method. It also instantiate an instance variable called `annualBonus`. Add a method `annualNetSalary` to `Employee` which returns the annual net salary of an employee including the annual basic and annual bonus.

18. For the code developed in problem 17, answer the following questions:
    a. What kind of relationship does this code shows between `Employee` and `BasicSalary`?
    b. Draw namespace diagram for the scenario when an object of class Employee is instantiated.

19. Define a class called `Point` that stores two coordinates of a 2D point. Add appropriate methods including initiator, `__str__` and setters. Add any other method if you want to.

20. Define another class `Distance` that instantiates two objects of `Point` type. Add appropriate initializers and setters. Add a method `findDistance` that returns the distance between the two stored points. Comment on the type of relationship between the two classes (`Point` and `Distance`).

21. To the `Distance` class developed in problem 19, add an overloaded − operator that subtracts two distances. It should allow statements like dist3= dist1-dist2. Assume that the operator will never be used to subtract a larger number from a smaller one (that is, negative distances are not allowed).

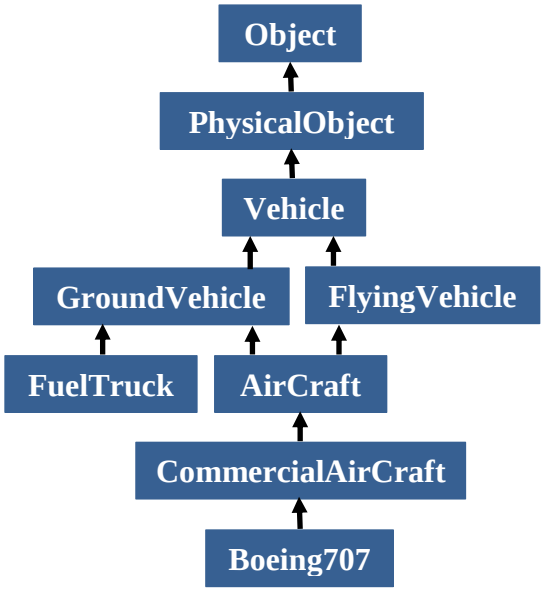
22. Define a mix-in class `DistanceFinder` containing only one method `findDistance(self,p)`. `p` is a `Point` type value and the method returns the distance between `self` and `p`. Now link this class to `Point` class via inheritance so that the current can find its distance from any other point.

23. Define a class `Polygon` to model polygons in two dimensional space. Each polygon is represented as a list of points. Define initiator which initiates an empty list. Add a method `addPoint` which adds a 2D point (using any `Point` class defined before) to the list. Add another method `perimeter` which finds perimeter of the shape by finding distance between all points in the list. For example:
    square = [p1, p2, p3, p4], where p1, p2, p3 and p4 are 4 corners of the square
    perimeter=distance(p1,p2) + distance(p2,p3) + distance(p3,p4) + distance(p4,p1)
    Do keep in mind that different shapes will have different list length, so perimeter method should be generic. You can either use `Distance` mix-in class or can define a static method within class `Polygon` to find the distance between two points.
    Test this class `Polygon` by defining the following shapes and finding their perimeters:
    square = (1,1), (1,2), (2,2), (2,1)
    triangle=(1,0), (2,2), (3,0)
    pentagon=(2,2), (3,3), (6,4), (5,2), (4,0)

24. Add to the `Time` class of problem 16 the ability to subtract two time values using the overloaded `-` operator, and to multiply a time value by a number of type float, using the overloaded `*` operator. The float should be multiplied by the hours component; if the hour component is 0, the float should be multiplied by the minute component; and if the minute component is also zero then the float should be multiplied by the second component.

25. Define a class `Fraction` which stores a fraction in two instance variables: numerator and denominator. Define appropriate initiator, setters and print methods. Override the following operators: `+`, `-`, `*`, `/`, `>`, `<`, `>=`, `<=`, `==`, and `!=`. Add another method `simplify` to simplify the fraction. You overloaded arithmetic methods should return a simplified fraction.

26. Consider the following class hierarchy:
    Find MRO for an object of class:
    a. Boeing707
    b. AirCraft
    c. FuelTruck
    d. GroundVehicle
    e. FlyingVehicle
    f. Vehicle



27. Code the class hierarchy define in problem 26. Just add one method in each class: `__init__`. This method should call the initiator of its parent class using the function `super()`, after which it prints its current class name. Use this code to verify your answers of problem 26.

28. Define a class called `fancyPrint` having one class variable `message` initialized to a `null` string. Define a setter method that sets a value to this class variable (don't convert it to instance variable). Now define a static method called `fancyPrint` which prints the class variable message in uppercase at the output. Write some test code to test this class.

29. Define a decorator called `makeFancy` to decorate the static method `fancyPrint()` defined in problem 28. This decorator should print some patterns of * before and after the message.
For example, consider the following test code and its corresponding output:

Test Code:

```
mssg=fancyPrint()
mssg.setMessage('Congratulations!!')
fancyPrint.fancyPrint()
```

Output:

```
**************************
CONGRATULATIONS!!
**************************
**********************
******************
**************
```