CSCI 1133
Exercise Set 11

You should complete as many of these problems as you can. Practice is *essential* to learning and reinforcing computational problem solving skills. We encourage you to do these without any outside assistance.

Create a directory named `exercise11` and save each of your problem solutions in this directory. You will need to save your source files in this directory and push them to your repository in order to get help/feedback from the TAs. This requires that you follow a rigid set of naming requirements. Name your individual Python source files using "`ex11`" followed by the letter of the exercise, e.g., "`ex11a.py`", "`ex11b.py`", etc.

For these first problems, include the class definitions in the module.

Automatic testing of your solutions is performed using a GitHub agent, so it is necessary to name your source files precisely as shown and push them to your repository as you work.

A. **Vehicles** *(adapted from Walter Savitch, Absolute C++)*

Create a base class called `Vehicle` that has the manufacturer's name, number of cylinders in the engine, and owner. Then create a class called `Truck` that is derived from `Vehicle` and has additional properties: the load capacity in tons and towing capacity in pounds. Be sure your classes have a reasonable complement of accessor and mutator methods. Write a program that tests all your methods.

B. **People** *(adapted from Daniel Liang, Introduction to Programming with C++)*

In this exercise you need to design a class named `Person`, and its derived classes named `Student` and `Employee`. Create additional `Faculty` and `Staff` classes, derived from Employee. A `Person` has a name, address, phone number, and e-mail address. A `Student` has a class status (freshman, sophomore, junior, or senior). An `Employee` has an office, salary, and a hire date. A `Faculty` member has office hours and a rank (assistant, associate, professor). A `Staff` member has a title. Define the `__repr__` method in the `Person` class and override it in each subclass to display the class name and the individual's name.

Write a test program that creates `Person`, `Student`, `Employee`, `Faculty`, and `Staff` objects and prints each of them.

C. **Rock-Paper-Scissors** *(adapted from MIT Open Courseware)*

In this exercise you need to implement a class named `Tool` that has 2 private instance variables: an integer named `__strength` and a string named `__type`. The `Tool` class should also contain a mutator method named `setStrength()` that updates the strength of the tool. Create three more classes: `Rock`, `Paper`, and `Scissors` that are each derived from `Tool`. These classes each have an initializer method that takes a *single* integer argument containing the object's strength. The initializer should also set its type field according to the following: 'r' for `Rock`, 'p' for `Paper`, and 's' for `Scissors`. Each of these classes must provide a boolean accessor method, `fight(Tool)` that compares their strengths in the following way:

- A rock's strength is doubled (temporarily) when fighting scissors, but halved (temporarily) when fighting paper.
- In the same way, paper has the advantage against rock, and scissors the advantage against paper.
- Do not alter the strength field in the function
- Return `True` if the calling object wins, `False` otherwise.

Write a program that creates a tool of `Rock`, `Paper`, and `Scissors`, and tests their fight methods.

D. **Guessing Game** (*adapted from W. Savitch, "Problem Solving with C++", 8th edition*)

Part 1: Listed below is code to play a guessing game. In the game two players attempt to guess a number. Your task is to extend the program with objects that represent either a human player or a computer player.

```
def guessingGame(player1, player2):
    answer = random.randint(0,100)
    while(True):
        print(player1.getName()+"'s turn to guess: ", end="")
        guess = player1.getGuess()
        if checkForWin(player1,guess,answer):
            return
        print(player2.getName()+"'s turn to guess: ", end="")
        guess = player2.getGuess()
        if checkForWin(player2,guess,answer):
            return

def checkForWin(player,guess,answer):
    print(player.getName(),"guesses", guess)
    if answer == guess:
        print("You're right! You win!")
        return True
    elif answer < guess:
        print("Your guess is too high.")
    else:
        print("Your guess is too low.")
    return
```

First, define a `Player` class with a single attribute for the player's name, an accessor method: `getName` that will return the player's name and a second accessor method: `getGuess` that will simply return zero.

Next, define a class named `HumanPlayer` derived from `Player`. Override the base-class `getGuess` method to prompt the user to enter a number, then return the entered value as an integer.

Next, define a class named `ComputerPlayer` derived from `Player`. Override the base-class `getGuess` method to return a number between 0 and 100.

Finally, construct a `main` program that will do the following:

- invoke `guessingGame()` using two instances of a `HumanPlayer` (human versus human)
- invoke `guessingGame()` using an instance of a `HumanPlayer` and `ComputerPlayer` (human versus computer)
- invoke `guessingGame()` using two instances of `ComputerPlayer` (computer versus computer).

You will need to include the random module and as well as the code listed above.

Part 2: The computer player in Part 1 does not play very well in the number guessing game since it only makes random guesses. Modify the program so that the computer plays a more sophisticated game. The specific strategy is up to you, but you will need to add method(s) to the `Player` and `ComputerPlayer` classes so that the `guessingGame` function can provide the results of a guess back to the player. In other words, the computer must be told if its last guess was too high or too low. The computer can then use this information to revise its next guess.

E. **Used Car Lot** (*adapted from T. Gaddis, Starting out With Python, 2nd ed.*)

This is a multi-part problem in which you will construct several object classes and a short demonstration program. Submit the entire program as your solution. Be sure to thoroughly test each part before moving on to the next.

A used car dealership maintains an inventory of several types and models of vehicles. There are three kinds of vehicles: Cars, Trucks and SUVs. Regardless the type, the dealership maintains the following information for every vehicle:

- Make
- Model
- Year
- Mileage
- Price

Additional information is maintained for each individual vehicle depending on its type.

For Cars:	Number of doors (2 or 4)
For Trucks:	Drive type (2-wheel drive or 4-wheel drive)
For SUVs:	Passenger Capacity

Part 1

Construct a base class named `Vehicle` to maintain the common vehicle data. The class should include a constructor that will initialize all 5 instance variables and separate accessor and mutator methods for each data attribute.

Part 2

Construct three additional classes named `Car`, `Truck` and `SUV` to represent Cars, Trucks, and SUVs respectively. Each of these classes should be derived from the `Vehicle` class and extend it by adding the attributes unique to the type of vehicle. Each class should provide a constructor to initialize its attribute(s) as well as the attributes of the parent class. Provide accessor and mutator methods for each class to get and set the attributes for the particular class.

Part 3

Add a method named `Display` to each of the four classes to print out the individual vehicle information. For the `Vehicle` class, the `Display` method should print the following (one element per line):

```
Make:  vehicle_make
Year:  vehicle_year
Model: vehicle_model
Miles: vehicle_mileage
Price: vechicle_price
```

Each vehicle-type class (Car, Truck, SUV) should print out the information specific to its own type in addition to the vehicle information (Hint: use the superclass' `Display` method). Here is an example of the output for a `Car`:

```
Inventory unit: Car
Make:  Audi
Year:  2009
Model: A8
Miles: 40000
Price: 27300.00
Number of doors: 4
```

Part 4

Write another class named `Inventory` that will maintain a list of vehicles in inventory. The constructor should start with a Null list. Add the following methods to the `Inventory` class:

- `addVehicle(`*vehicle*`)` : A mutator that will add the *vehicle* object to the inventory list
- `Display( )` : An accessor that will print out the vehicle information for every vehicle in the inventory. Separate each vehicle's information with two blank lines

Part 5

Finally, write a non-pure function named `main` that will solicit inventory information from the user (keyboard input). The program should prompt the user to enter a vehicle type (car, truck or SUV) and then input all the information appropriate to that vehicle type. It should then construct an appropriate vehicle instance and add it to the inventory. This process should continue until the user indicates he/she is done entering vehicle data. The program should then print out the entire inventory by calling the inventory `Display` method.