Classes, Objects and Inheritance

1. Starting with the employee class you created in the previous assignment, create a manager class. This class should use the previously created employee class as the base class and make a derived class called manager. This class should include a long term bonus method that compute the long term bonus to be 40% of their salary. Create a program to instantiate the new class and show that it works.

**Create a derived class called Executive.**

Add a method called ExecutiveBonus computed to be 200% of their annual salary.

Override the long term bonus method to be 50% of their annual salary.

Instantiate and test your object.

|  |  |  |
| --- | --- | --- |
| Input | Process | Output |
| Employee’s first, last name and salary | Class employee:  Def\_init\_(self, first, last, salary):  self.first = first  self.last = last  self.pay = salary  self.email = self.first + '.' + self.last + '@company.com' | Employee’s first, last name, email, and salary  Manager’s first, last name, and email, salary, and long-term bonus  Executive’s first, last name, email, salary, and executive bonus |
| Manager’s first, last name and salary | Def fullname (self):  return '{} {}'.format(self.first,self.last)  def bonus (self):  return self.pay x 0.1 |  |
| Executive’s first, last name and salary | Class executive (manager):  Def long term bonus (self):  Return self.pay x 0.4  Class executive (manager):  Def long term bonus (self):  Return self.pay x 0.5  Def executive bonus (self):  Return self.pay x 0.2 |  |
|  | employee1 = Employee("Sam", "White", 50000)  manager1 = Manager("John", "Brown", 60000)  executive1 =Executive("Sarah", "Lee", 70000) |  |
|  | print(f"{employee1.fullname()} ({employee1.email}) has a salary of ${employee1.pay} and a bonus of ${employee1.bonus():.2f}")  print(f"{manager1.fullname()} ({manager1.email}) has a salary of ${manager1.pay}, a 10% bonus of ${manager1.bonus():.2f}, and a long-term bonus of ${manager1.long term bonus():.2f}")  print(f"{executive1.fullname()} ({executive1.email}) has a salary of ${executive1.pay}, a 10% bonus of ${executive1.bonus():.2f}, a long-term bonus of ${executive1.long term bonus():.2f}, and an executive bonus of ${executive1.executive bonus():.2f}") |  |
|  |  |  |
|  |  |  |

1. Create a car class. This should have methods for make, model, sticker price and discount price (90% of sticker price). Then create a derived class called sport. Inherit the car class but add options methods. Set the option to Y to include the option in the updated price method. See table below. Define a method for each option.

**New for this assignment.**

Create a derived class called Luxury. Override the Option Method to include:

Options Option Price

GPS 5,000.00

Self-Driving 10,000.00

Instantiate and test your object.

Options (method) Option Price

SportWheels 1000.00

SportEngine 3000.00

SportInterior 2000.00

For each method set to Y add the amounts to the updated price and display using a method called pricewithoptions.

Write program to instantiate the object and show that it works.

|  |  |  |
| --- | --- | --- |
| Input | Process | Output |
| Car make, model, sticker price | Class car:  Def\_init\_(self, make, model, sticker price)  Self.make = make  Self model = model  Self. Sticker price = sticker price  Def discounted price (self):  Return self. Sticker price x 0.9  Def display information (self):  Return f"{self.make} {self.model} - Sticker Price: ${self.sticker\_price:.2f}, Discounted Price: ${self.discounted\_price():.2f}" | Make, model, sticker price, and discounted price |
| Sports car | Class sport (car):  Def\_init\_( self, make, model, sticker price):  self.sport\_wheels\_price = 1000.00  self.sport\_engine\_price = 3000.00  self.sport\_interior\_price = 2000.00 | Sport car price with selected options |
| Luxury car | Def price with options self, sport wheels="N", sport engine="N", sport interior="N"):  total price = self.discounted\_price() if sport wheels.upper() == "Y": total price += self.sport\_wheels\_price if sport engine.upper() == "Y": total price += self.sport\_engine\_price if sport\_interior.upper() == "Y": total price += self.sport\_interior\_price return total price  def display info(self, sport wheels="N", sport engine="N", sport interior="N"): return f"{self.make} {self.model} - Final Price with Options: ${self.price\_with\_options(sport\_wheels, sport\_engine, sport\_interior):.}" | Luxury car price with selected options |
|  | Class Luxury(Car):  def \_\_init\_\_(self, make, model, sticker price):  super().\_\_init\_\_(make, model, sticker price) self.gps\_price = 5000.00 self.self\_driving\_price = 10000.00  def price\_with\_options(self, gps="N", self driving="N"):  total price = self.discounted\_price() if gps.upper() == "Y": total price += self.gps\_price if self\_driving.upper() == "Y": total price += self.self\_driving\_price  return total price  def display info(self, sport\_wheels="N", sport\_engine="N", sport\_interior="N"):  return f"{self.make} {self.model} - Final Price with Options: ${self.price\_with\_options(sport\_wheels, sport\_engine, sport\_interior):.}" |  |
|  |  |  |
|  | car1 = Car("Toyota", "Camry", 40000)  sport1 = Sport("Ford", "Mustang", 50000)  luxury1 = Luxury("Tesla", "Model S", 90000) |  |
|  | print(car1.display\_info())  print(sport1.display\_info(sport\_wheels="Y", sport\_engine="N", sport\_interior="Y"))  print(sport1.display\_info(sport\_wheels="Y", sport\_engine="Y", sport\_interior="Y"))  print(luxury1.display\_info(gps="Y", self\_driving="N")) print(luxury1.display\_info(gps="Y", self\_driving="Y")) |  |