Lab 13
OOP – BSDS – Spring 2022

Note: There is only one lengthy task, don't go on reading and reading instead read one or two paragraph and write code, continue reading and writing code. Until you read till last. Finally, run driver function and find a lot of error (obviously you have created), so correct them and one by one run all menu options given in driver function.

With reference to inheritance case study discussed in class, you have to design a simple real estate application that allows an agent to manage properties available for purchase or rent. There will be two types of properties: apartments and houses. The agent needs to be able:

- to enter a new property
- list all currently available properties
- list type of specific property
- modify a property

Create class **Property** (with data members *square feet, no of bedrooms & no of* baths). You have to write member functions *init, str & set values*. In function *set values*, you have to take input from user and assign to data members. For this and all coming *str* functions see output (given at the end of the description).

House is child class of Property (with data members *garage, fenced, no of stories*). You have to write member functions *init, str & set values*. The function *set values* is similar to the *set values* function of Property class.

In addition, you have to define two class level members: (required to validate input values)

```
valid_garage = ('attached', 'detached', 'none')
valid_fenced = ('yes', 'no')
```

In the function *set values* call *get_valid_input* function (given in functions.py) to take input with two parameters. First parameter is message for user and second parameter is relevant class level member to validate input.

Apartment is also child class of Property (with data members *balcony, laundry*). You have to write member functions *init, str & set values*. The function *set values* is similar to the *set values* function of Property class.

In addition, you have to define two class level members: (required to validate input values)

```
valid_laundries = ('coin', 'ensuite', 'none')
valid_balconies = ('yes', 'no', 'solarium')
```

In the function *set values* call *get_valid_input* function (given in functions.py) to take input.

Rentals and purchases also seem to require separate representation. Create class **Rental** with data members *furnished, rent & utilities*. Write member functions *init, str & set values*. Create class **Purchase** with data members *price & taxes*. Write member functions *init & str*.

Create class **HouseRental, HousePurchase ApartmentRental & ApartmentPurchase.** Each of these four classes will inherit two parent classes like *HouseRental* will inherit *House & Rental.* In *init* function of *HouseRental & ApartmentRental* just call the *init* functions of the parent classes. In *init function of HousePurchase & ApartmentPurchase* take data members of parent classes (wherever required) and call *init* functions of parent classes.

Finally, write **Agent** class. The only instance data member is list of all properties. However, there is a class level member: (just copy paste)

```
class_dict = {('house', 'rental'): HouseRental, ('house', 'purchase'):
HousePurchase, ('apartment', 'rental'): ApartmentRental,('apartment',
'purchase'): ApartmentPurchase }
```

Write *init* function and initialized list with empty brackets. Copy paste *add property* function given below:

```
def add_property(self):
        property_type = get_valid_input('What type of property? ', ('house',
'apartment')).lower()
        payment_type = get_valid_input( 'What payment type? ', ('purchase',
'rental')).lower()
        item = None
        if property_type == 'house' and payment_type == 'purchase': item =
HousePurchase()
        elif property_type == 'house' and payment_type == 'rental': item =
HouseRental()
        if property_type == 'apartment' and payment_type == 'purchase': item
= ApartmentPurchase()
        elif property_type == 'apartment' and payment_type == 'rental': item
= ApartmentRental()
        self.property_list.append(item)
```

Add functions *show all properties, show match properties and modify properties* in class *Agent*. In show all function, just traverse list and print objects.

In *show match properties*, take input property type and payment type and print all match properties. You have to use class level dictionary object here.

In *modify properties* function take property no as input. The user is supposed to give the number one to length of list. If number is invalid show message of wrong input, otherwise call set values function already defined in the classes.

Finally, run driver function, you may run driver function even before the completion of all functions of class *Agent*. In the menu option, select only those functions, which are created. A sample run is given here:

Output:

```
1. Add new property
2. Show specific property
3. Show all property
4. Modify property
5. Quit from system
Enter Your Choice: 1
What type of property? (house, apartment) house
What payment type? (purchase, rental) purchase
What is the selling price? 10000000
What are the estimated taxes? 50000

1. Add new property
2. Show specific property
3. Show all property
4. Modify property
5. Quit from system
Enter Your Choice: 1
What type of property? (house, apartment) apartment
What payment type? (purchase, rental) purchase
What is the selling price? 5000000
What are the estimated taxes? 5000

1. Add new property
2. Show specific property
3. Show all property
4. Modify property
5. Quit from system
Enter Your Choice: 1
What type of property? (house, apartment) house
What payment type? (purchase, rental) rental

1. Add new property
2. Show specific property
3. Show all property
4. Modify property
5. Quit from system
Enter Your Choice: 1
What type of property? (house, apartment) apartment
What payment type? (purchase, rental) rental

1. Add new property
2. Show specific property
3. Show all property
4. Modify property
5. Quit from system
Enter Your Choice: 3
PROPERTY DETAILS
================
square footage: 100
bedrooms: 1
bathrooms: 1
HOUSE DETAILS
# of stories: 1
garage: YES
fenced yard: YES
PURCHASE DETAILS
selling price: 10000000
estimated taxes: 50000
PROPERTY DETAILS
================
square footage: 100
bedrooms: 1
bathrooms: 1
APARTMENT DETAILS
laundry: YES
has balcony: YES

PURCHASE DETAILS
selling price: 5000000
estimated taxes: 5000
PROPERTY DETAILS
================
square footage: 100
bedrooms: 1
bathrooms: 1
HOUSE DETAILS
# of stories: 1
garage: YES
fenced yard: YES
RENTAL DETAILS
rent: 20000
estimated utilities: YES
furnished: YES

1. Add new property
2. Show specific property
3. Show all property
4. Modify property
5. Quit from system
Enter Your Choice: 4
Enter property no to modify: 1
Is the yard fenced? (yes, no) yes
Is there a garage? (attached, detached, none) attached
How many stories? 3

1. Add new property
2. Show specific property
3. Show all property
4. Modify property
5. Quit from system
Enter Your Choice: 3
PROPERTY DETAILS
================
square footage: 100
bedrooms: 1
bathrooms: 1
HOUSE DETAILS
# of stories: 1
garage: YES
fenced yard: YES
PURCHASE DETAILS
selling price: 10000000
estimated taxes: 50000
PROPERTY DETAILS
================
square footage: 100
bedrooms: 1
bathrooms: 1
APARTMENT DETAILS
laundry: YES
has balcony: YES

PURCHASE DETAILS
selling price: 5000000
estimated taxes: 5000
PROPERTY DETAILS
```

```
================
square footage: 100
bedrooms: 1
bathrooms: 1
HOUSE DETAILS
# of stories: 1
garage: YES
fenced yard: YES
RENTAL DETAILS
rent: 20000
estimated utilities: YES
furnished: YES

1. Add new property
2. Show specific property
3. Show all property
4. Modify property
5. Quit from system
Enter Your Choice: 5
Thank, for using the system
```