CSCI3180 Asg2 Report

Ling Leong 1155062557 Unix id: llling5

Task 2 Q1

1. More generic code can be written. In other words, functions can be defined to apply on arguments of different types.

Example 1 - Python:

```
class Hotel:
    @staticmethod
    def move_room(obj_list, destination):
        for obj in obj_list:
            obj.move_to(destination)

if __name__ == '__main__':
    peter = Attendant("Peter", 58, 10000, 7)
    mary = Attendant("Mary", 45, 90000, 1)
    tony = Attendant("Tony", 60, 1000000, 1)
    washer1 = Washer("Washer 1", 55, 1, 5000)
    refrigerator1 = Refrigerator("Refrigerator 1", 50, 1, 5000, 3)
    table1 = Table("Table 1", 20, 1, peter, True)
    chair1 = Chair("Chair 1", 8, 1, peter, False)
    old_room_stuff = [mary, tony, washer1, refrigerator1, table1, chair1]
    old_room = Room("Room 2046", 1)
    new_room = Room("Room 2047", 5)
    dining_hall_1 = DiningHall("Dinning hall 1", 12, 80)
    Hotel.move_room(old_room_stuff, new_room)
    peter.move_to(dining_hall_1)
```

In the above code, hotel has a generic method move_room(obj_list, destination), where obj_list is an python list storing anything object that has move_to method defined. It doesn't matter what is the type of the obj in the list, it can be Attendant, Washer, Table or Chair, as long as that type has a move_to method. Same for the move_to method, it can accept any object type as long as it has the position property. This methodology is also called duck typing and dynamic typed language like Python make duck typing easy to implement. The code above is much cleaner than the normal way of writing in static typed language, where you may need to do type checking first and then cast the object type to call their corresponding move_to method. And your move_to method may need to be overloaded multiple time for different object type. However, it is worth mention that in Java there is Interface which can help us simulate duck typing. For example, in this case, we could have defined an interface named Moveable, which defines a move_to method. So, any class can implement the Moveable interface and we can implement a method that accept Moveable type object. This way our method can accept object of any type if they implemented Moveable. The advantage of dynamic typing here is that it doesn't have those complications like interface, it will just work.

2. Possibilities of mixed type collection data structures.

Dynamic typing also allows us to have mixed type collection data structures, for example, we can still refer to the example 1 above, in python, we can have the List data type which can store object of any type. In the above code, old_room_stuff is a list that stored [mary, tony, washer1, refrigerator1, table1, chair1], which consist of Attendant, Washer, Table, Chair, etc. They can all be stored in a single list without type casting. In Java, you will need to cast them to object when storing and cast them back to their original type when using them.

Disadvantages: No type checking at compile time or in IDE, hard to debug and discover the error early. Type checking is done in runtime which slow down performance.

Task 3 Q2

Example 2 -Java

```
peter.MoveTo(dining_hall_1);
```

```java
public double MoveTo(Room Destination){
...
}
public double MoveTo(Fixture Destination){
...
}
public double MoveTo(Room Destination, Fixture Obj){
...
}
```

Example 2 – Python

```python
def move_to(self, destination, obj=None)
```

In Java, polymorphism is done by inheritance. For example, in MoveTo, we can move to different fixture type like table or washer, this is done by having them inherit the Fixture base class so that they can be treated as the same class. We can also see that peter can move to different type of room, this is done by method overloading and having DiningHall inherit the Room base class.

In Python, we do polymorphism by duck typing where we simply define the necessary method or property for each class. For example, we can call move_to with any object that has the position property, we do not need the destination to inherit a specific class.

Task 3 Q3

Example 3 -Java

```java
public class Hotel {
    public static void MoveRoom(Object[] Obj_list, Room Destination){
        for(Object obj: Obj_list){
            if(obj instanceof Fixture){
                ((Fixture) obj).MoveTo(Destination);
            }
            else if(obj instanceof Attendant){
                ((Attendant) obj).MoveTo(Destination);
            }
        }
    }
```

The first advantage of Python implementation is that we can have mixed type list and we can have more generic code. We can compare the move_room in example 1 and example 3, in Python we only need 2 lines of code, where in Java we need 6 lines, and we need to do typing checking and type casting. Python one is obviously cleaner.

For second advantage, we can refer to example 2, due to dynamic typing and duck typing we can use one single method move_to(self, destination, obj=None) to handle any type of object, but in Java we need to overload it for different type of object, and most of the code is repeated. We need to do the same thing many times.

Task 4 Q4

Example 4 - Java

```java
public DumbFixture(String Name, double Weight, int Position, Attendant Person_in_charge) {
    super(Name, Weight, Position);
    this.person_in_charge = Person_in_charge;
}

public DumbFixture(String Name, double Weight, int Position, Robot Robot_in_charge) {
    super(Name, Weight, Position);
    this.robot_in_charge = Robot_in_charge;
}
```

In task 4, we see another advantage for duck typing, which is extensibility. When we need to add new class, or change type to exist code, in Python we can simply add the new class without any changes to existing code, but for Java, we need to add new overloaded method to handle new class, create new constructor for existing class to accept the new class, or modify the parameter type. If the code base is large this can be a huge issue. Python avoid most of this issue.