05-Design-Patterns

November 21, 2019

1 Design patterns

1.1 Singleton

```
In [25]: class Singleton:
             def __new__(cls,*a, **b):
                 if hasattr(cls,'_inst'):
                     return cls._inst
                 else:
                     cls._inst=super().__new__(cls,*a,**b)
                     return cls._inst

         class Counter(Singleton):
             def __init__(self):
                 if not hasattr(self,'val'):
                     self.val = 0
             def get(self):
                 return self.val
             def incr(self):
                 self.val +=1
```

```
In [26]: print(dir(Counter))
         #print(dir(Counter[_inst]))
         a=Counter()
         print(dir(Counter))
```

```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
```

```
In [27]: print(Counter._inst)
         b=Counter()
         a.incr()
         b.incr()

         c=Counter()
         c.incr()
```

```
        print(b,c)
        print(a.get(),b.get(),c.get())
        Counter().incr()
        Counter().get()
```

```
<__main__.Counter object at 0x7f2f95513240>
<__main__.Counter object at 0x7f2f95513240> <__main__.Counter object at 0x7f2f95513240>
3 3 3
```

```
Out[27]: 4
```

```
In [42]: def Singleton(cls):
                 '''generic python decorator to make any class
                 singleton.'''
                 _instances = {}          # keep classname vs. instance
                 def getinstance():
                         '''if cls is not in _instances create it
                         and store. return the stored instance'''
                         if cls not in _instances:
                                 _instances[cls] = cls()
                         return _instances[cls]
                 return getinstance


         @Singleton
         class Config:
             def __init__(self):
                 self.vals = {}
             def __setitem__(self,k,v):
                 self.vals[k]=v
             def __getitem__(self,k):
                 return self.vals[k]


         a=Config()
         a['username']='onur'
         b=Config()
         print(b['username'])
         Config()['filename']='445.txt'
         Config()['database']='mysql://localhost/ceng445'
         Config().vals
```

```
onur
```

```
Out[42]: {'database': 'mysql://localhost/ceng445',
          'filename': '445.txt',
          'username': 'onur'}
```

1.2 Observer

```
In [43]: class OSubject:
             def __init__(self):
                 self.observers = []
             def register(self,obs):
                 self.observers.append(obs)
             def unregister(self,obs):
                 self.observers.remove(obs)
             def notify(self):
                 for obs in self.observers:
                     obs.update()

         class Clock(OSubject):
             def __init__(self):
                 self.value = 0
                 super().__init__()
             def get(self):
                 return self.value
             def tick(self):
                 self.value +=1
                 self.notify()

         class Person:
             def __init__(self,name,clock):
                 self.name = name
                 self.clock = clock
                 clock.register(self)

             def update(self):
                 print('Updated:',self.name, self.clock.get())
```

```
In [44]: c=Clock()

         p1=Person('ali',c)
         p2=Person('veli',c)
```

```
In [45]: c.tick()
```

```
Updated: ali 1
Updated: veli 1
```

```
In [46]: c.tick()
```

```
Updated: ali 2
Updated: veli 2
```

```
In [47]: c.unregister(p2)

In [48]: c.tick()

Updated: ali 3
```