10 PATRONES de diseño (ejemplos)Python

#Investigación 4

**Command Pattern**

Ena situación regular de oficina donde, para hablar a un Director de ventas de una empresa, el Cliente hace una llamada primero al recepcionista de la oficina de director de ventas y luego el Recepcionista pasa la llamada al director. En este caso, Director de ventas sería el Sujeto a quien el Cliente quiere hablar a y el Recepcionista sería el Poder que protege el Sujeto de hablar directamente a los Clientes.

Ampliando este ejemplo, nosotros podríamos considerar 'al Director de ventas' como el Verdadero Sujeto y crear una Clase común Sustancial 'Directores' llamados de lo cual 'el Director de ventas' 'y el Recepcionista' pueden ser sacados.

**class Switch:**

**""" The INVOKER class"""**

**def \_\_init\_\_(self, flipUpCmd, flipDownCmd):**

**self.\_\_flipUpCommand = flipUpCmd**

**self.\_\_flipDownCommand = flipDownCmd**

**def flipUp(self):**

**self.\_\_flipUpCommand.execute()**

**def flipDown(self):**

**self.\_\_flipDownCommand.execute()**

**class Light:**

**"""The RECEIVER Class"""**

**def turnOn(self): print "The light is on"**

**def turnOff(self): print "The light is off"**

**class Command: """The Command Abstract class"""**

**def \_\_init\_\_(self): pass #Make changes**

**def execute(self): #OVERRIDE pass**

**class FlipUpCommand(Command): """The Command class for turning on the light"""**

**def \_\_init\_\_(self,light): self.\_\_light = light**

**def execute(self): self.\_\_light.turnOn()**

**class FlipDownCommand(Command): """The Command class for turning off the light"""**

**def \_\_init\_\_(self,light): Command.\_\_init\_\_(self) self.\_\_light = light**

**def execute(self): self.\_\_light.turnOff()**

**class LightSwitch: """ The Client Class"""**

**def \_\_init\_\_(self): self.\_\_lamp = Light()**

**self.\_\_switchUp = FlipUpCommand(self.\_\_lamp)**

**self.\_\_switchDown = FlipDownCommand(self.\_\_lamp)**

**self.\_\_switch = Switch(self.\_\_switchUp,self.\_\_switchDown)**

**def switch(self,cmd): cmd = cmd.strip().upper() try:**

**if cmd == "ON": self.\_\_switch.flipUp()**

**elif cmd == "OFF": self.\_\_switch.flipDown()**

**else: print "Argument \"ON\" or \"OFF\" is required."**

**except Exception, msg: print "Exception occured: %s" % msg # Ejecuta esto si el archive corre como un script y no como un modulo importado**

**if \_\_name\_\_ == "\_\_main\_\_":**

**lightSwitch = LightSwitch()**

**print "Switch ON test."**

**lightSwitch.switch("ON")**

**print "Switch OFF test"**

**lightSwitch.switch("OFF")**

**print "Invalid Command test"**

**lightSwitch.switch("\*\*\*\*")**

Observer Pattern

**class Publisher:**

**def \_\_init\_\_(self):**

**#MAke it uninheritable**

**pass**

**def register(self):**

**#OVERRIDE**

**pass**

**def unregister(self):**

**#OVERRIDE**

**pass**

**def notifyAll(self):**

**#OVERRIDE**

**pass**

**class TechForum(Publisher):**

**def \_\_init\_\_(self):**

**self.\_listOfUsers = []**

**self.postname = None**

**def register(self, userObj):**

**if userObj not in self.\_listOfUsers:**

**self.\_listOfUsers.append(userObj)**

**def unregister(self, userObj):**

**self.\_listOfUsers.remove(userObj)**

**def notifyAll(self):**

**for objects in self.\_listOfUsers:**

**objects.notify(self.postname)**

**def writeNewPost(self, postname):**

**# User writes a post.**

**self.postname = postname**

**# When submits the post is published and notification is sent to all**

**self.notifyAll()**

**class Subscriber:**

**def \_\_init\_\_(self):**

**#make it uninheritable pass**

**def notify(self):**

**#OVERRIDE pass class User1(Subscriber):**

**def notify(self,postname):**

**print 'User1 notfied of a new post %s' % postname**

**class User2(Subscriber):**

**def notify(self, postname):**

**print 'User2 notfied of a new post %s' % postname**

**class SisterSites(Subscriber):**

**def \_\_init\_\_(self):**

**self.\_sisterWebsites = ["Site1","Site2","Site3"]**

**def notify(self, postname):**

**for site in self.\_sisterWebsites: # Send updates by any means**

**print "Sent nofication to site: %s" % site**

**if \_\_name\_\_ == "\_\_main\_\_":**

**techForum = TechForum()**

**user1 = User1()**

**user2 = User2()**

**sites = SisterSites()**

**techForum.register(user1)**

**techForum.register(user2)**

**techForum.register(sites)**

**techForum.writeNewPost("Observer Pattern in Python")**

**techForum.unregister(user2)**

**techForum.writeNewPost("MVC Pattern in Python")**

Facade Pattern

**#Complex Parts import time class TC1:**

**def run(self):**

**print "###### In Test 1 ######"**

**time.sleep(1)**

**print "Setting up"**

**time.sleep(1)**

**print "Running test"**

**time.sleep(1)**

**print "Tearing down"**

**time.sleep(1)**

**print "Test Finished\n"**

**class TC2:**

**def run(self):**

**print "###### In Test 2 ######"**

**time.sleep(1)**

**print "Setting up"**

**time.sleep(1)**

**print "Running test"**

**time.sleep(1)**

**print "Tearing down"**

**time.sleep(1)**

**print "Test Finished\n" class TC3:**

**def run(self):**

**print "###### In Test 3 ######"**

**time.sleep(1)**

**print "Setting up"**

**time.sleep(1)**

**print "Running test"**

**time.sleep(1)**

**print "Tearing down"**

**time.sleep(1)**

**print "Test Finished\n" #Facade**

**class TestRunner:**

**def \_\_init\_\_(self):**

**self.tc1 = TC1()**

**self.tc2 = TC2()**

**self.tc3 = TC3()**

**def runAll(self):**

**self.tc1.run()**

**self.tc2.run()**

**self.tc3.run() #Client**

**if \_\_name\_\_ == '\_\_main\_\_':**

**testrunner = TestRunner()**

**testrunner.runAll()**

**Mediator Pattern**

**import time**

**class TC:**

**def \_\_init\_\_(self): self.\_tm = tm self.\_bProblem = 0**

**def setup(self): print "Setting up the Test"**

**time.sleep(1) self.\_tm.prepareReporting()**

**def execute(self):**

**if not self.\_bProblem: print "Executing the test"**

**time.sleep(1)**

**else: print "Problem in setup. Test not executed."**

**def tearDown(self):**

**if not self.\_bProblem: print "Tearing down"**

**time.sleep(1) self.\_tm.publishReport()**

**else: print "Test not executed. No tear down required."**

**def setTM(self,TM): self.\_tm = tm**

**def setProblem(self, value): self.\_bProblem = value**

**class Reporter: def \_\_init\_\_(self): self.\_tm = None**

**def prepare(self): print "Reporter**

**Class is preparing to report the results"**

**time.sleep(1)**

**def report(self): print "Reporting the results of Test" time.sleep(1)**

**def setTM(self,TM): self.\_tm = tm**

**class DB: def \_\_init\_\_(self): self.\_tm = None**

**def insert(self): print "Inserting the execution begin status in the Database" time.sleep(1)**

**#Following code is to simulate a communication from DB to TC**

**import random**

**if random.randrange(1,4) == 3: return -1**

**def update(self): print "Updating the test results in Database" time.sleep(1) def setTM(self,TM): self.\_tm = tm**

**class TestManager:**

**def \_\_init\_\_(self): self.\_reporter = None self.\_db = None self.\_tc = None**

**def prepareReporting(self): rvalue = self.\_db.insert()**

**if rvalue == -1: self.\_tc.setProblem(1) self.\_reporter.prepare()**

**def setReporter(self, reporter): self.\_reporter = reporter**

**def setDB(self, db): self.\_db = db def publishReport(self): self.\_db.update() rvalue = self.\_reporter.report()**

**def setTC(self,tc): self.\_tc = tc**

**if \_\_name\_\_ == '\_\_main\_\_': reporter = Reporter()**

**db = DB()**

**tm = TestManager()**

**tm.setReporter(reporter)**

**tm.setDB(db)**

**reporter.setTM(tm)**

**db.setTM(tm)**

**# For simplification we are looping on the same test. # Practically, it could be about various unique test classes and their objects while (1): tc = TC() tc.setTM(tm) tm.setTC(tc) tc.setup() tc.execute() tc.tearDown()**

**Factory Pattern**

**class Person:**

**def \_\_init\_\_(self):**

**self.name = None**

**self.gender = None**

**def getName(self): return self.name**

**def getGender(self): return self.gender**

**class Male(Person):**

**def \_\_init\_\_(self, name): print "Hello Mr." + name**

**class Female(Person):**

**def \_\_init\_\_(self, name): print "Hello Miss." + name**

**class Factory:**

**def getPerson(self, name, gender):**

**if gender == 'M':**

**return Male(name)**

**if gender == 'F':**

**return Female(name)**

**if \_\_name\_\_ == '\_\_main\_\_':**

**factory = Factory()**

**person = factory.getPerson("Chetan", "M")**

**Proxy Pattern**

**import time**

**class Manager(object):**

**def work(self):**

**pass**

**def talk(self):**

**pass**

**class SalesManager(Manager):**

**def work(self):**

**print "Sales Manager working..."**

**def talk(self):**

**print "Sales Manager ready to talk"**

**class Proxy(Manager):**

**def \_\_init\_\_(self):**

**self.busy = 'No'**

**self.sales = None**

**def work (self):**

**print "Proxy checking for Sales Manager availability"**

**if self.busy == 'Yes':**

**self.sales = SalesManager()**

**time.sleep(2);**

**self.sales.talk()**

**else:**

**time.sleep(2);**

**print "Sales Manager is busy"**

**if '\_\_name\_\_' == '\_\_main\_\_':**

**p = Proxy() p.work()**