

**Usman Institute of Technology**

**Department of Computer Science Fall 2022**

Name: Muhammad Waleed

Roll no: 20B-115-SE

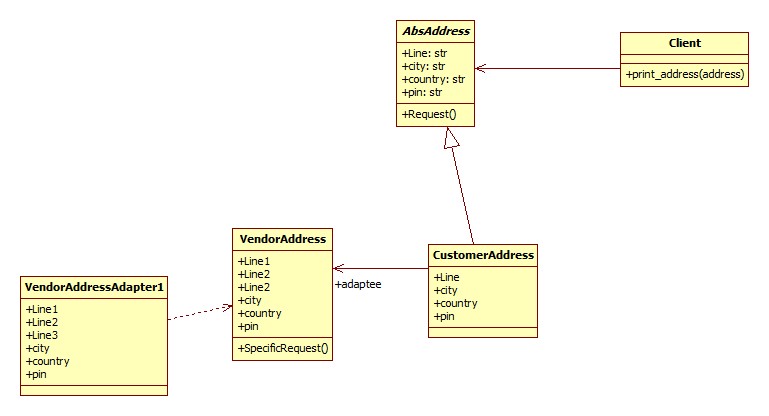
Course: Software Design and Architecture (SE-308)

Course Instructor: Misbah ud Din

Date: 24-Nov-2022

# Lab Tasks:

## AddressAdapter:



Code:

from abc import ABC

class AbsAddress(ABC):

    line: str

    city: str

    country: str

    pin: str

class VendorAddress:

    def \_\_init\_\_(self, line1, line2, line3, city, country, pin):

        self.line1 = line1

        self.line2 = line2

        self.line3 = line3

        self.city = city

        self.country = country

        self.pin = pin

class CustomerAddress(AbsAddress):

    def \_\_init\_\_(self, line, city, country, pin):

        self.line = line

        self.city = city

        self.country = country

        self.pin = pin

class VendorAddressAdapter:

    def \_\_init\_\_(self, vendor\_address):

        self.line = f'{vendor\_address.line1}, {vendor\_address.line2}, {vendor\_address.line3}'

        self.city = vendor\_address.city

        self.country = vendor\_address.country

        self.pin = vendor\_address.pin

# client

def print\_address(address):

    print(f'{address.line}, {address.city}, {address.country}, {address.pin}')

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

    cust\_address = CustomerAddress("House No", "A. B C Road", "Karachi", 74550)

    vend\_address = VendorAddress("Home # 1", "Apartment 1", "Street 4", "A. B C Road", "karachi", 45700)

    vend\_address\_adapt = VendorAddressAdapter(vend\_address)

    for address in [cust\_address, vend\_address\_adapt]:

        print\_address(address)

Output:



## Web Page Bridge:

Diagram

Description automatically generated

Code:

class WebPage:

    def \_\_init\_\_(self, theme):

        self.theme = theme

    def getContent(self):

        pass

class About(WebPage):

    \_theme = None

    def \_\_init\_\_(self, theme):

        self.theme = theme

    def getContent(self):

        return "About page in " + self.theme.getColor()

class Careers(WebPage):

    \_theme = None

    def \_\_init\_\_(self, theme):

        self.theme = theme

    def getContent(self):

        return "Careers page in " + self.theme.getColor()

class Theme:

    def getColor(self):

        pass

class DarkTheme(Theme):

    def getColor(self):

        return 'Dark Black'

class LightTheme(Theme):

    def getColor(self):

        return 'Off White'

class AquaTheme(Theme):

    def getColor(self):

        return 'Light Blue'

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

    darkTheme = DarkTheme()

    lightTheme = LightTheme()

    about = About(darkTheme)

    careers = Careers(darkTheme)

    aboutLight = About(lightTheme)

    careersLight = Careers(lightTheme)

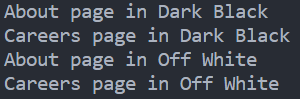
    print(about.getContent())

    print(careers.getContent())

    print(aboutLight.getContent())

    print(careersLight.getContent())

Output:



## Department Composite:

Diagram

Description automatically generated

Code:

from abc import ABC, abstractmethod

class BaseDepartment(ABC):

    @abstractmethod

    def \_\_init\_\_(self, num\_of\_employees):

        pass

    @abstractmethod

    def print\_department(self):

        pass

class Accounting(BaseDepartment):

    def \_\_init\_\_(self, num\_of\_employees):

        self.num\_of\_employees = num\_of\_employees

    def print\_department(self):

        print(f"Accounting employees: {self.num\_of\_employees}")

class Development(BaseDepartment):

    def \_\_init\_\_(self, num\_of\_employees):

        self.num\_of\_employees = num\_of\_employees

    def print\_department(self):

        print(f"Development employees: {self.num\_of\_employees}")

class Management(BaseDepartment):

    def \_\_init\_\_(self, num\_of\_employees):

        self.num\_of\_employees = num\_of\_employees

        self.childs = []

    def print\_department(self):

        print(f"Management base employees: {self.num\_of\_employees}")

        total\_emp\_count = self.num\_of\_employees

        for child in self.childs:

            total\_emp\_count += child.num\_of\_employees

            child.print\_department()

            print(f'Total employees: {total\_emp\_count}')

    def add\_child\_dept(self, dept):

        self.childs.append(dept)

    def delete\_child\_dept(self,dept):

        self.child.pop(dept)

#

acc\_dept = Accounting(200)

dev\_dept = Development(500)

management\_dept = Management(50)

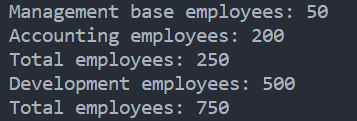
management\_dept.add\_child\_dept(acc\_dept)

management\_dept.add\_child\_dept(dev\_dept)

# print dept

management\_dept.print\_department()

Output:



# Home Tasks:

## Socket Adapter:

class Socket:

    def \_\_init\_\_(self):

        self.voltage = 220

        self.type = '2Pins'

    def get\_voltage(self):

        return self.voltage

    def get\_type(self):

        return self.type

class Charger:

    def \_\_init\_\_(self):

        self.voltage = 220

        self.type = '3Pins'

    def get\_voltage(self):

        return self.voltage

    def get\_type(self):

        return self.type

class Adapter:

    def \_\_init\_\_(self, socket):

        self.socket = socket

    def get\_voltage(self):

        return self.socket.get\_voltage()

    def get\_type(self):

        return self.socket.get\_type()

class Device:

    def \_\_init\_\_(self, adapter):

        self.adapter = adapter

    def charge(self):

        if self.adapter.get\_type() == '3Pins':

            print('Charging...')

        else:

            print('Cannot charge. Adapter type is not compatible.')

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

    print('Socket Type: 2Pins')

    print('Socket Voltage: 220V')

    print('Mobile Charger Type: 3Pins')

    print('Mobile Charger Voltage: 220V')

    print('Mobile Socket Type: 3Pins')

    charger = Charger()

    adapter = Adapter(charger)

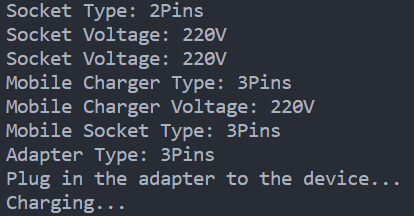
    print('Adapter Type: {}'.format(adapter.get\_type()))

    device = Device(adapter)

    print('Plug in the adapter to the device...')

    device.charge()

Output:



## Encryption Bridge:

class Encryption:

    def encrypt(self):

        pass

class AES(Encryption):

    def encrypt(self):

        print('Encrypting with AES')

class Blowfish(Encryption):

    def encrypt(self):

        print('Encrypting with Blowfish')

class EncryptionBridge:

    def \_\_init\_\_(self, encryption):

        self.encryption = encryption

    def encrypt(self):

        self.encryption.encrypt()

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

    aes = AES()

    blowfish = Blowfish()

    aesBridge = EncryptionBridge(aes)

    blowfishBridge = EncryptionBridge(blowfish)

    aesBridge.encrypt()

    blowfishBridge.encrypt()

Output:



## PCComposite:

# composite design pattern example

class PC:

    def \_\_init\_\_(self, name):

        self.name = name

        self.parts = []

    def add(self, part):

        self.parts.append(part)

    def remove(self, part):

        self.parts.remove(part)

    def get\_price(self):

        price = 0

        for part in self.parts:

            price += part.get\_price()

        return price

    def get\_name(self):

        return self.name

class Part:

    def \_\_init\_\_(self, name, price):

        self.name = name

        self.price = price

    def get\_price(self):

        return self.price

    def get\_name(self):

        return self.name

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

    pc = PC('PC')

    cpu = Part('CPU', 100)

    ram = Part('RAM', 50)

    pc.add(cpu)

    pc.add(ram)

    print('Total price of {} is {}'.format(pc.get\_name(), pc.get\_price()))

Output:

