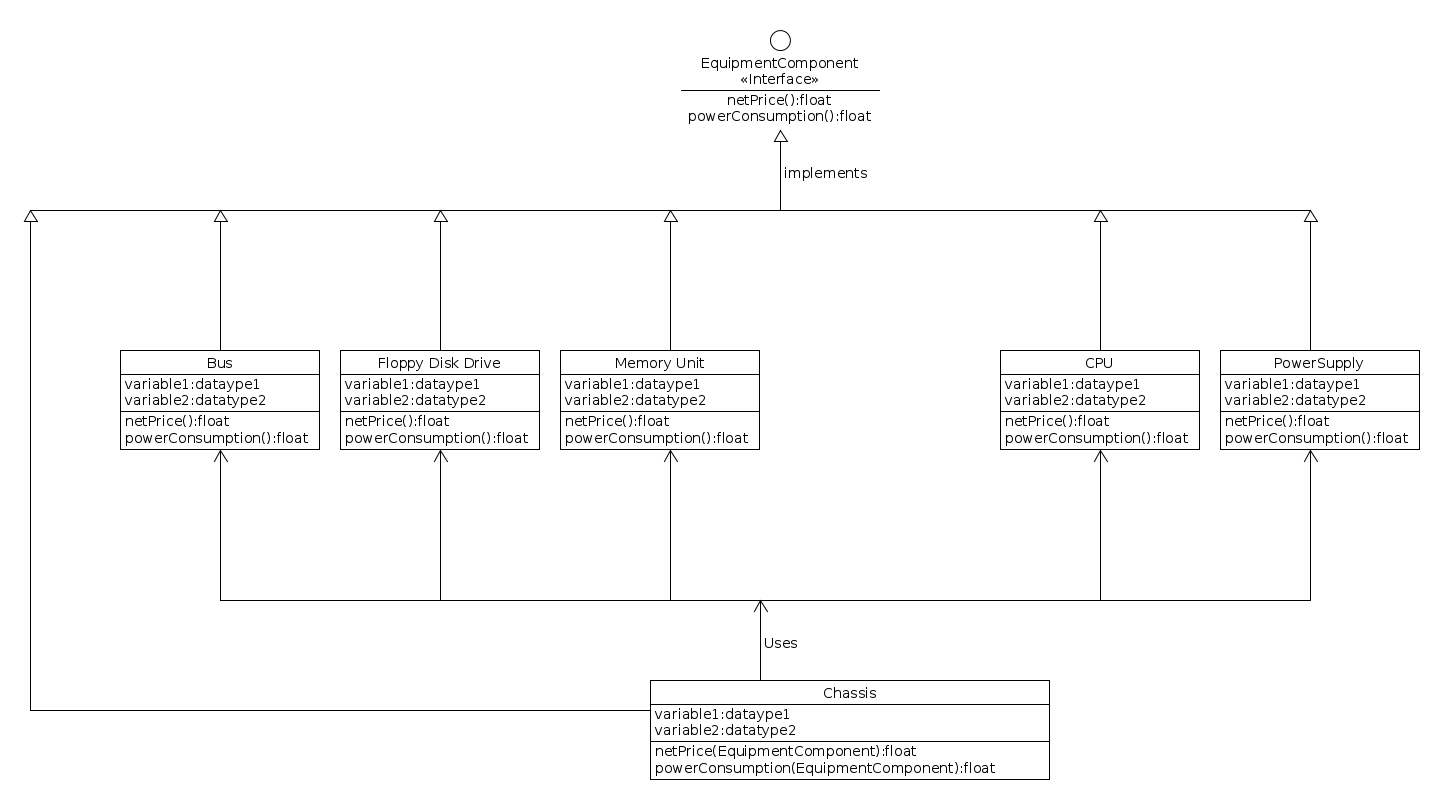
Design Pattern Assignment Solution

1. a) Builder Design Pattern.



b) Algorithm for above Diagram

/\*Interface class comtaining basic operations needed to implement by all components\*/

public interface EquipmentComponent{

public float netPrice();

public float powerConsumption();

}

public class Bus implements EquipmentComponent{

//may or may not have specific semantics(variables and methods)

//inherited method

public float netPrice(){

//return price of bus

}

public float powerConsumption(){

//return power Consumption by bus

}

}

public class FloppyDiskDrive implements EquipmentComponent{

//may or may not have specific semantics(variables and methods)

//inherited method

public float netPrice(){

//return price of FloppyDiskDrive

}

public float powerConsumption(){

//return power Consumption by FloppyDiskDrive

}

}

public class MemoryUnit implements EquipmentComponent{

//may or may not have specific semantics(variables and methods)

//inherited method

public float netPrice(){

//return price of MemoryUnit

}

public float powerConsumption(){

//return power Consumption by MemoryUnit

}

}

public class CPU implements EquipmentComponent{

//may or may not have specific semantics(variables and methods)

//inherited method

public float netPrice(){

//return price of CPU

}

public float powerConsumption(){

//return power Consumption by CPU

}

}

public class PowerSupply implements EquipmentComponent{

//may or may not have specific semantics(variables and methods)

//inherited method

public float netPrice(){

//return price of PowerSupply

}

public float powerConsumption(){

//return power Consumption by PowerSupply

}

}

public class Chassis implements EquipmentComponent{

//may or may not have specific semantics(variables and methods)

Bus b ;

FloppyDiskDrive f;

MemoryUnit m;

CPU c;

PowerSupply p;

public Chassis(bus b,FloppyDiskDrive f,MemoryUnit m,CPU c,PowerSupply p){

this.b = b;

this.f = f;

this.m = m;

this.c = c;

this.p = p;

}

//inherited method

public float netPrice(EquipmentComponent e){

//return price of e

return e.netPrice();

}

public float powerConsumption(){

//return power Consumption of e

return e.powerConsumption();

}

}

public class Cabinet{

public static void main (String[]){

Chassis ch = new Chassis(new Bus(),new FloppyDiskDrive(),new MemoryUnit(),new CPU(),new PowerSupply());

//want to find price of bus

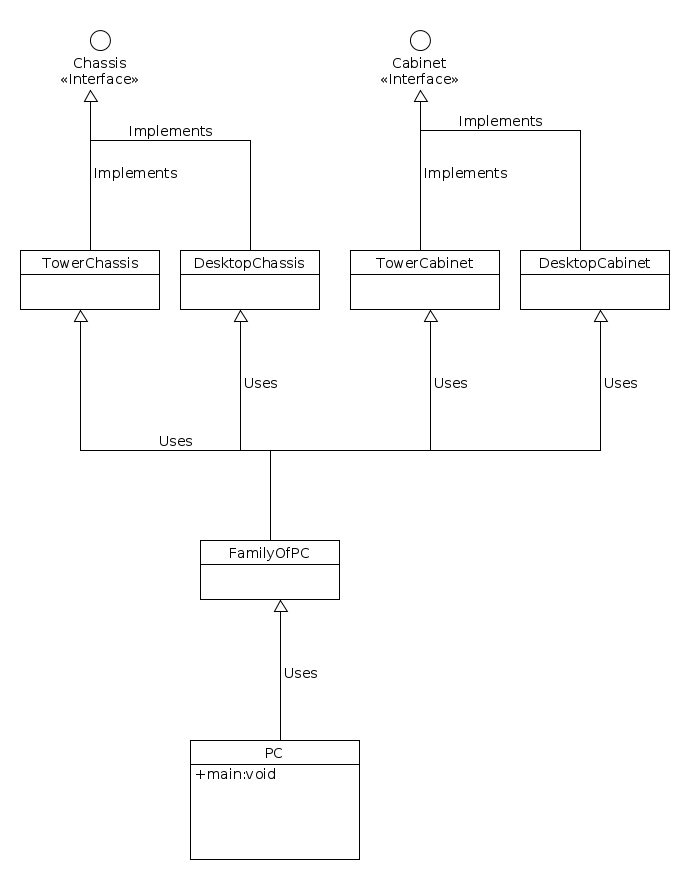
display ch.netPrice(ch.b);

display ch.powerConsumption(ch.b);

}

}

2) a)filter Design Pattern



2) b)

interface Chassis{

//may or may not have variable and methods

}

interface Cabinet{

//may or may not have variable and methods

}

class TowerChassis implements Chassis{

// implements Chassis methods and have Tower members

}

class DesktopChassis implements Chassis{

//implements Chassis methods and have Desktop members

}

class TowerCabinet implements Cabinet{

//implements Cabinet methods and have Tower members

}

class DesktopCabinet implements Cabinet {

//implements Cabinet methods and have Desktop members

}

class FamilyOfPC{

getTowerChassis(){

return new TowerChassis();

}

getTowerCabinet(){

return new TowerCabinet();

}

getDesktopChassis(){

return new DesktopChassis();

}

getDesktopCabinet(){

return new DesktopCabinet();

}

}

class PC {

main(){

FamilyOfPC pc= new FamilyOfPC();

Chassis towerChassis = pc.getTowerChassis();

}

}

3) Decorator Pattern

4) a) Strategy Pattern

b)

interface SortAlgorithm {

void sort();

}

class Merge implements SortAlgorithm{

void sort(){.....};

}

class Insertion implements SortAlgorithm{

void sort(){...};

}

class StrategyPattern {

SortAlgorithm s;

THRESHOLD={constant value};

static void sorting(arrray,uppervalue,lowervalue){

if( array size is below THRESHOLD){

s = new Insertion();

}else{

s=new Merge();

}

s.sort(array,uppervalue,lowervalue);

}

}

class DemoClass{

main(){

StrategyPattern demo = new StrategyPattern();

demo.sorting(array,uppervalue,lowervalue);

}

}