**Design Pattern Answers**

1. **Assignment -1**

a) **Composite Design Pattern**

b) Pc

class Pc

{

Float price, power;

Float netPrice()

{

Return price;

}

Float netPowerConsumption()

{

Return power;

}

}

Cabinet

class Cabinet extends Pc

{

Float netPrice()

{

Return price;

}

Float netPowerConsumption()

{

Return power;

}

}

Chasis

class Chasis extends Cabinet

{

Float netPrice()

{

Return price;

}

Float netPowerConsumption()

{

Return power;

}

}

Bus

class Bus extends Chasis

{

Float netPrice()

{

Return price;

}

Float netPowerConsumption()

{

Return power;

}

}

Floppy Disk Drive

class FloppyDiskDrive extends Chasis

{

Float netPrice()

{

Return price;

}

Float netPowerConsumption()

{

Return power;

}

}

Memory Unit

class MemoryUnit extends Chasis

{

Float netPrice()

{

Return price;

}

Float netPowerConsumption()

{

Return power;

}

}

CPU

class Cpu extends Chasis

{

Float netPrice()

{

Return price;

}

Float netPowerConsumption()

{

Return power;

}

}

Power Supply

class PowerSupply extends Chasis

{

Float netPrice()

{

Return price;

}

Float netPowerConsumption()

{

Return power;

}

}

Network Card

class NetworkCard extends Bus

{

Float netPrice()

{

Return price;

}

Float netPowerConsumption()

{

Return power;

}

}

1. **Assignment -2**

a)**Abstract Factory Method**

b)Pseudocode:

class DesktopPc

{

//all class variables and member function declaration and definition.

}

Class DesktopCabinet extends DesktopPc

{

//all class variables and member function declaration and definition.

}

Class DesktopChasis extends DesktopCabinet

{

//all class variables and member function declaration and definition.

}

And so on further class hierarchy for desktop pc for its further components like desktop bus, desktop floppy drive, desktop Memory Unit, desktop CPU, desktop Power Supply, desktop network card, etc.

class TowerPc

{

//all class variables and member function declaration and definition.

}

Class TowerCabinet extends TowerPc

{

//all class variables and member function declaration and definition.

}

Class TowerChasis extends TowerCabinet

{

//all class variables and member function declaration and definition.

}

And so on further class hierarchy for desktop pc for its further components like tower bus, tower floppy drive, tower Memory Unit, tower CPU, tower Power Supply, tower network card, etc.

Abstract class AbstractFactory

{

//all class variables and member function declaration and definition.

}

Class DesktopPcFactory extends AbstractFactory

{

//all class variables and member function declaration and definition.

}

Class TowerPcFactory extends AbstractFactory

{

//all class variables and member function declaration and definition.

}

class FactoryCreator {

      static AbstractFactory getFactory(String choice){

**if**(choice.equals(“DesktopPc")){

         return new DesktopPcFactory();

      } else if(choice.equals ("TowerPc")){

         return new TowerPcFactory();

      }

      return null;

   }

Class AbstractFactoryPatternExample

{

//Take Input for which pc user needs Desktop or Tower Pc.

AbstractFactory ob=FactoryCreator.getFactory(user\_choice);

}

**3. Assignment -3**

**Decorator Design Pattern**

**4. Assignment -4**

a) **Strategy Design Pattern**

b)Pseudocode:

public interface SortingInterface

{

Public void algo(float performance\_value[],int size)

{

}

}

Class SortingAlgorithm\_1 implements SortingInterface

{

Public void algo(float performance\_value[],int size)

{

}

Public String timeComplexity()

{

//return time complexity

}

Public String spaceComplexity()

{

//return spaceComplexity

}

}

Class SortingAlgorithm\_2 implements SortingInterface

{

Public void algo(float performance\_value[],int size)

{

}

Public String timeComplexity()

{

//return time complexity

}

Public String spaceComplexity()

{

//return spaceComplexity

}

}

Class SortingAlgorithm\_3 implements SortingInterface

{

Public void algo(float performance\_value[],int size)

{

}

Public String timeComplexity()

{

//return time complexity

}

Public String spaceComplexity()

{

//return spaceComplexity

}

}

Class SortingAlgorithmSelector

{

Public String chooseAlgorithm()

{

//find the least space and time complexities of various sorting algorithms and choose the one having minimum complexity

// ruin the algorithm having minimum time and space complexity

// call that particular algorithm

}

}