**Week 5**

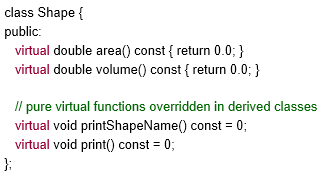
****

# Exercise 1

Define an inheriting system that can inherit a class called “DataStructure”, this class will include virtual functions from the list below to serve different data structures. The suggested data structures are listed below:

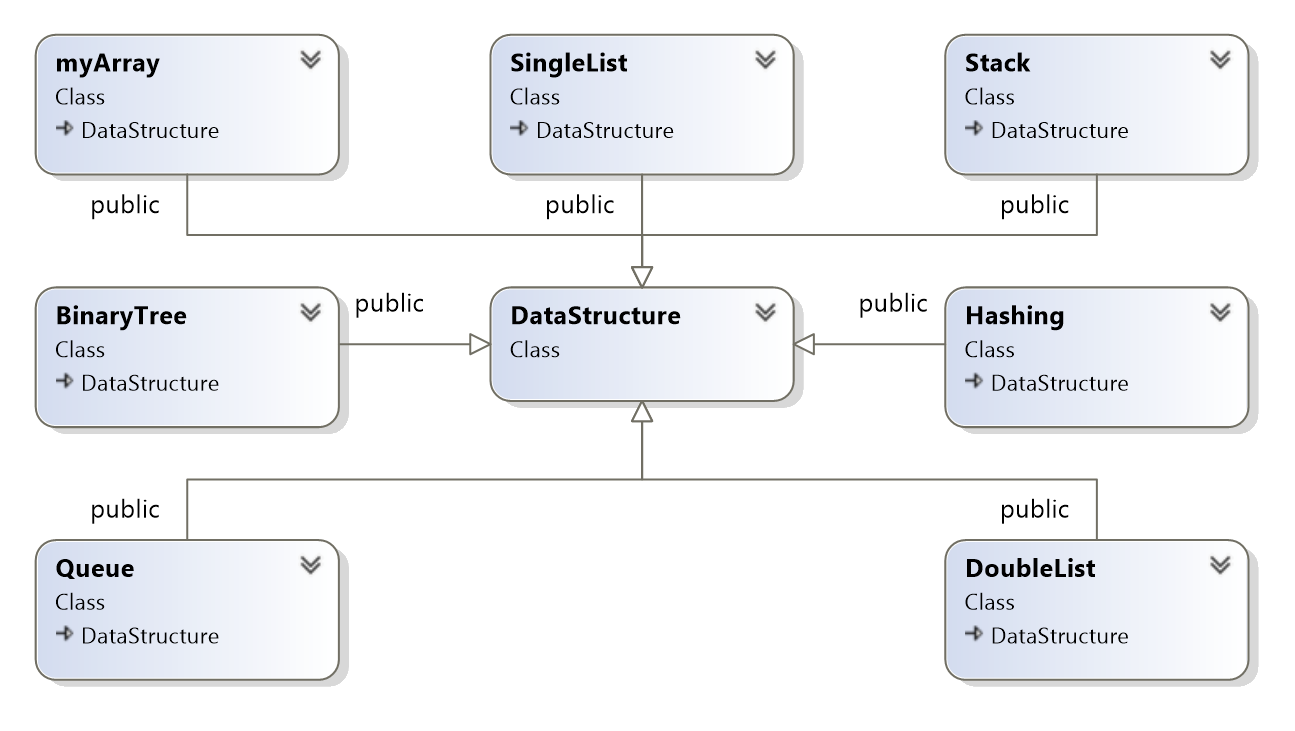
1. MyArray.
2. Stack.
3. Queue.
4. Single Linked list.
5. Doubly linked list.
6. Binary tree.
7. Hashing.

Based on the structured above there are standard functionalities can be used with these structures, some of them can be common methods, and the others can be specific for certain structure(s), the functionalities (Methods) can be as shown below:

1. Initialize
2. IsFull pure virtual
3. IsEmpty pure virtual
4. Insert
5. Delete
6. Push
7. Pop
8. ShowElement pure virtual
9. ShowAll pure virtual
10. ClearAll pure virtual
11. DeleteStruct pure virtual
12. Etc…

In your design, you do not need to implement the functionalities rather than have an empty function for a future implementation.

Discuss designing such a system without using virtual function and what will be the drawback if applicable in such system.



//DataStructure.h

#pragma once

class DataStructure //all functions in one class so 1. All of functions can adhere to the given signature (design) = enforcing a standard 2. Can make an address array of other classes implementing this class

{

public: //always virtual functions should be public

virtual bool isFull();

virtual void Insert();

virtual void Delete();

virtual void Push();

virtual void Pop();

//pure virtual functions overridden in derived classes

//pure virtual functions = all classes are using

virtual bool isEmpty() const = 0;

virtual void ShowElement() const = 0; //for Stack and Queue this is negotiable

virtual void ShowAll() const = 0;

virtual void ClearAll() const = 0;

virtual void Initialise() = 0;

virtual void DeleteStruct() = 0;

};

//myArray.h

#pragma once

#include "DataStructure.h"

class myArray :

public DataStructure

{

public:

myArray();

~myArray();

virtual bool isFull();

virtual void Insert();

virtual void Delete();

//pure virtual functions overridden in derived classes

//pure virtual functions = all classes are using

virtual bool isEmpty() const = 0;

virtual void ShowElement() const = 0;

virtual void ShowAll() const = 0;

virtual void ClearAll() const = 0;

virtual void Initialise() = 0;

virtual void DeleteStruct() = 0;

};

//Stack.h

#pragma once

#include "DataStructure.h"

class Stack :

public DataStructure

{

public:

Stack();

~Stack();

virtual bool isFull();

virtual void Insert();

virtual void Delete();

virtual void Push();

virtual void Pop();

//pure virtual functions overridden in derived classes

//pure virtual functions = all classes are using

virtual bool isEmpty() const = 0;

virtual void ShowElement() const = 0;

virtual void ShowAll() const = 0;

virtual void ClearAll() const = 0;

virtual void Initialise() = 0;

virtual void DeleteStruct() = 0;

};

//Queue.h

#pragma once

#include "DataStructure.h"

class Queue :

public DataStructure

{

public:

Queue();

~Queue();

virtual bool isFull();

virtual void Insert();

virtual void Delete();

virtual void Push();

virtual void Pop();

//pure virtual functions overridden in derived classes

//pure virtual functions = all classes are using

virtual bool isEmpty() const = 0;

virtual void ShowElement() const = 0;

virtual void ShowAll() const = 0;

virtual void ClearAll() const = 0;

virtual void Initialise() = 0;

virtual void DeleteStruct() = 0;

};

//Singly Linked List.h

#pragma once

#include "DataStructure.h"

class SingleList :

public DataStructure

{

public:

SingleList();

~SingleList();

virtual void Insert();

virtual void Delete();

//pure virtual functions overridden in derived classes

//pure virtual functions = all classes are using

virtual bool isEmpty() const = 0;

virtual void ShowElement() const = 0;

virtual void ShowAll() const = 0;

virtual void ClearAll() const = 0;

virtual void Initialise() = 0;

virtual void DeleteStruct() = 0;

};

//Doubly Linked List.h

#pragma once

#include "DataStructure.h"

class DoubleList :

public DataStructure

{

public:

DoubleList();

~DoubleList();

virtual void Insert();

virtual void Delete();

//pure virtual functions overridden in derived classes

//pure virtual functions = all classes are using

virtual bool isEmpty() const = 0;

virtual void ShowElement() const = 0;

virtual void ShowAll() const = 0;

virtual void ClearAll() const = 0;

virtual void Initialise() = 0;

virtual void DeleteStruct() = 0;

};

//BinaryTree.h

#pragma once

#include "DataStructure.h"

class BinaryTree :

public DataStructure

{

public:

BinaryTree();

~BinaryTree();

virtual void Insert();

virtual void Delete();

//pure virtual functions overridden in derived classes

//pure virtual functions = all classes are using

virtual bool isEmpty() const = 0;

virtual void ShowElement() const = 0;

virtual void ShowAll() const = 0;

virtual void ClearAll() const = 0;

virtual void Initialise() = 0;

virtual void DeleteStruct() = 0;

};

//Hashing.h

#pragma once

#include "DataStructure.h"

class Hashing :

public DataStructure

{

public:

Hashing();

~Hashing();

virtual void Insert();

virtual void Delete();

//pure virtual functions overridden in derived classes

//pure virtual functions = all classes are using

virtual bool isEmpty() const = 0;

virtual void ShowElement() const = 0;

virtual void ShowAll() const = 0;

virtual void ClearAll() const = 0;

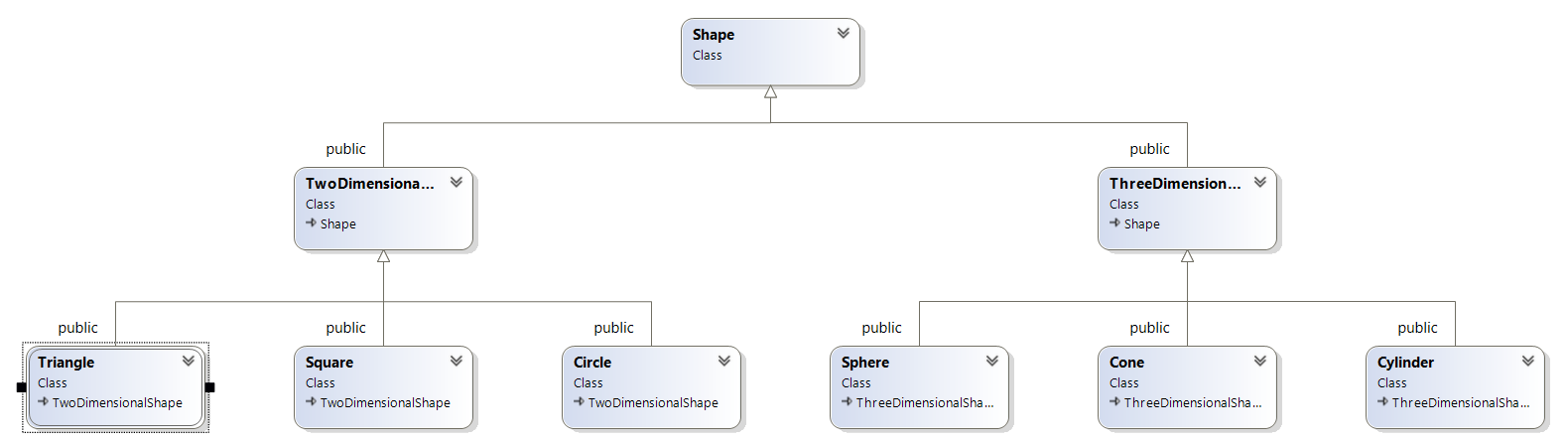
virtual void Initialise() = 0;

virtual void DeleteStruct() = 0;

};

# Exercise 2

Write down all the shapes you can think of--both two-dimensional and three-dimensional--and form those shapes into a shape hierarchy. Your hierarchy should have base class Shape from which class TwoDimensionalShape and class ThreeDimensionalShape are derived. Once you have developed the hierarchy, define each of the classes in the hierarchy. We will use this hierarchy in the coming exercises to process all shapes as objects of base-class Shape. This is a technique called polymorphism.



//Shape.h

#pragma once

class Shape

{

public:

//Virtual Functions

virtual double area() const { return 0.0; }

virtual double surfaceArea() const { return 0.0; }

virtual double volume() const { return 0.0; }

//Pure virtual functions overridden in derived classes

virtual void printShapeName() const = 0;

virtual void print() const = 0;

virtual void enlarge() const = 0;

virtual void transform() const = 0;

};

//TwoDimensionalShape.h

#pragma once

#include "Shape.h"

class TwoDimensionalShape :

public Shape

{

virtual double area() const;

virtual double side() const;

//Pure virtual functions overridden in derived classes

virtual void printShapeName() const;

virtual void print() const;

virtual void enlarge() const;

virtual void transform() const;

};

//circle.h

#pragma once

#include "TwoDimensionalShape.h"

class Circle :

public TwoDimensionalShape

{

public:

Circle();

~Circle();

//Virtual Functions

virtual double area() const;

virtual double side() const;

//Pure virtual functions overridden in derived classes

virtual void printShapeName() const;

virtual void print() const;

virtual void enlarge() const;

virtual void transform() const;

private:

int x, y;

};

//Square.h

#pragma once

#include "TwoDimensionalShape.h"

class Square :

public TwoDimensionalShape

{

public:

Square();

~Square();

//Virtual Functions

virtual double area() const;

virtual double side() const;

//Pure virtual functions overridden in derived classes

virtual void printShapeName() const;

virtual void print() const;

virtual void enlarge() const;

virtual void transform() const;

private :

int x, y;

};

//Triangle.h

#pragma once

#include "TwoDimensionalShape.h"

class Triangle :

public TwoDimensionalShape

{

public:

Triangle();

~Triangle();

//Virtual Functions

virtual double area() const;

virtual double side() const;

//Pure virtual functions overridden in derived classes

virtual void printShapeName() const;

virtual void print() const;

virtual void enlarge() const;

virtual void transform() const;

private:

int x, y;

};

//ThreeDimensionalShape.h

#pragma once

#include "Shape.h"

class ThreeDimensionalShape :

public Shape

{

virtual double height() const;

virtual void CrossSection() const;

virtual double surfaceArea() const;

virtual double volume() const;

virtual double height() const;

virtual void CrossSection() const;

//Pure virtual functions overridden in derived classes

virtual void printShapeName() const;

virtual void print() const;

virtual void enlarge() const;

virtual void transform() const;

};

//Cone.h

#pragma once

#include "ThreeDimensionalShape.h"

class Cone :

public ThreeDimensionalShape

{

public:

Cone();

~Cone();

//Virtual Functions

virtual double height() const;

virtual void CrossSection() const;

virtual double surfaceArea() const;

virtual double volume() const;

virtual double height() const;

virtual void CrossSection() const;

//Pure virtual functions overridden in derived classes

virtual void printShapeName() const;

virtual void print() const;

virtual void enlarge() const;

virtual void transform() const;

private:

int x, y, z;

};

//Cylinder.h

#pragma once

#include "ThreeDimensionalShape.h"

class Cylinder :

public ThreeDimensionalShape

{

public:

Cylinder();

~Cylinder();

//Virtual function

virtual double height() const;

virtual void CrossSection() const;

virtual double surfaceArea() const;

virtual double volume() const;

virtual double height() const;

virtual void CrossSection() const;

//Pure virtual functions overridden in derived classes

virtual void printShapeName() const;

virtual void print() const;

virtual void enlarge() const;

virtual void transform() const;

private:

int x, y, z;

};

//Sphere.h

#pragma once

#include "ThreeDimensionalShape.h"

class Sphere :

public ThreeDimensionalShape

{

public:

Sphere();

~Sphere();

//Virtual functions

virtual double height() const;

virtual void CrossSection() const;

virtual double surfaceArea() const;

virtual double volume() const;

virtual double height() const;

virtual void CrossSection() const;

//Pure virtual functions overridden in derived classes

virtual void printShapeName() const;

virtual void print() const;

virtual void enlarge() const;

virtual void transform() const;

private:

int x, y, z;

};