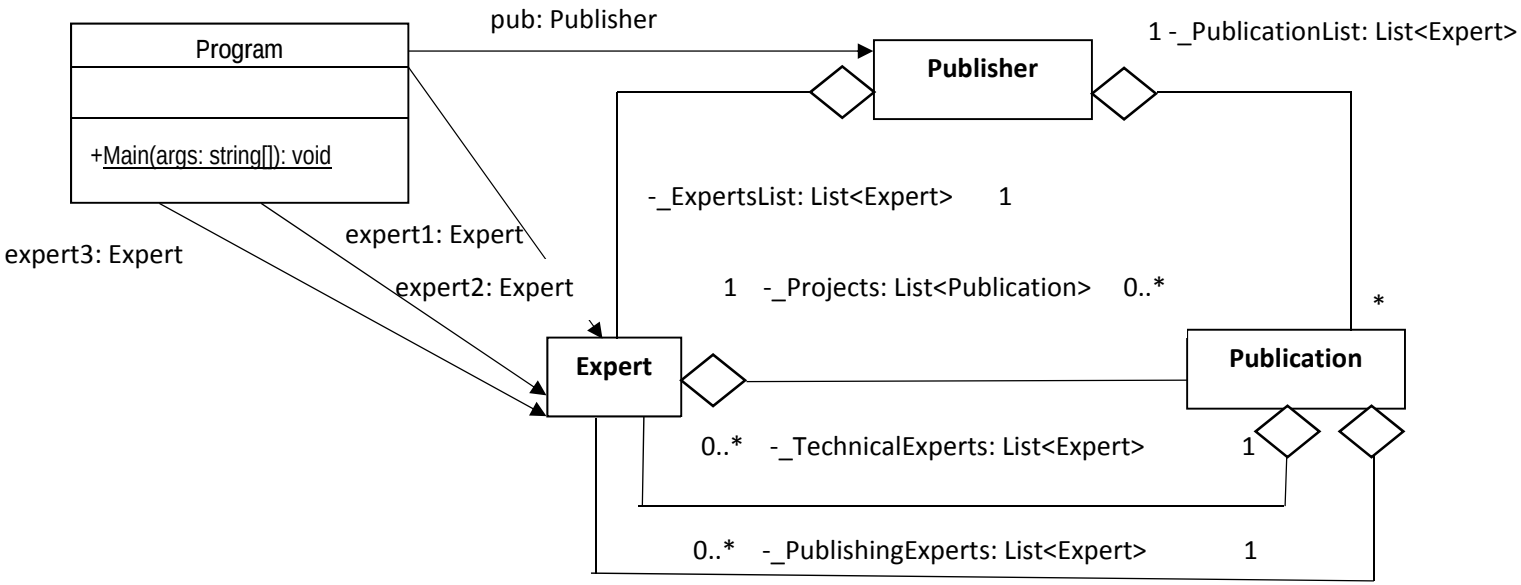
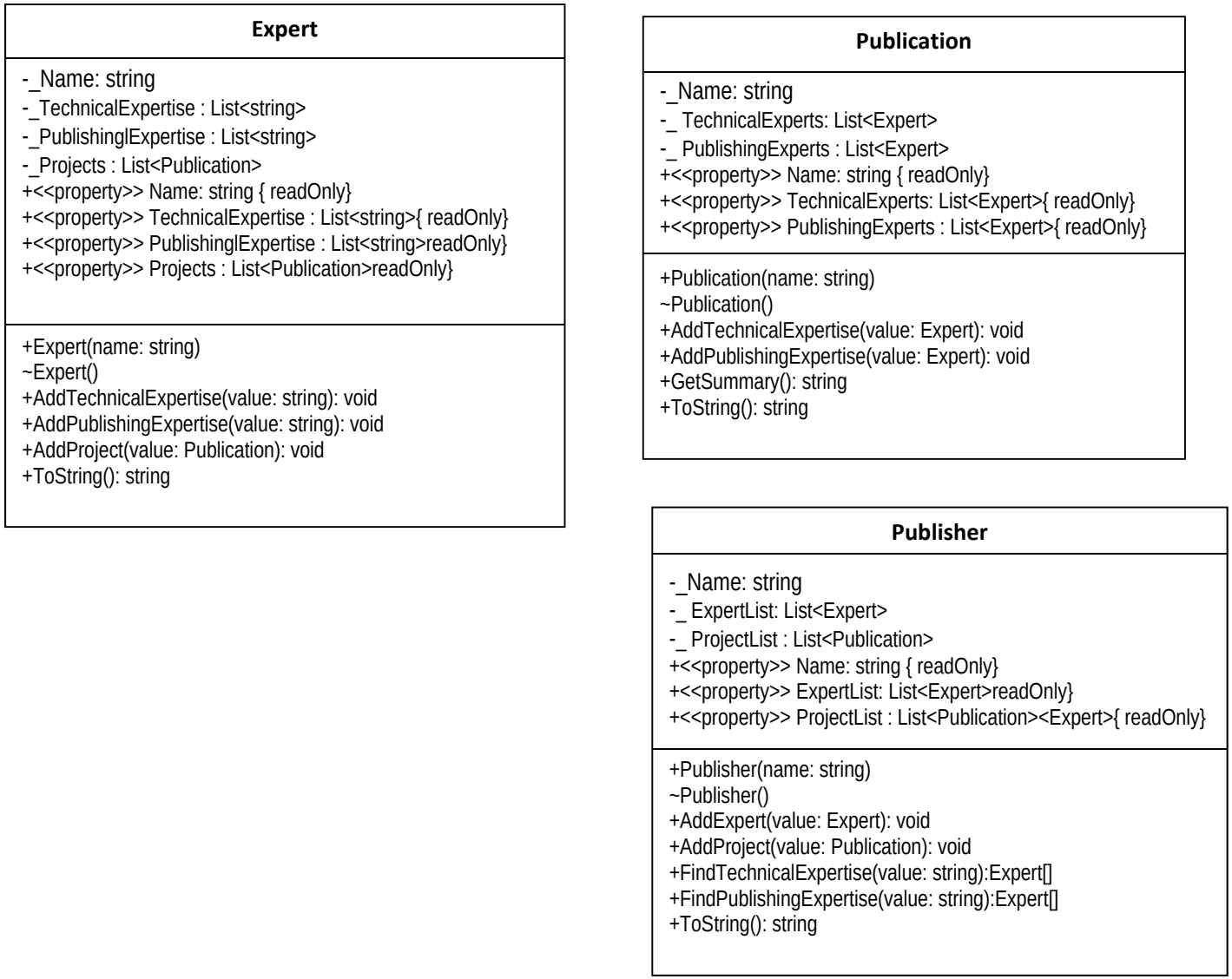

Program
+Main(args: string[]): void
pub: Publisher
Publisher
1 -_PublicationList: List<Expert>
-_ExpertsList: List<Expert> 1
1 -_Projects: List<Publication> 0..*
*
expert1: Expert
expert2: Expert
expert3: Expert
Expert
Publication
0..* -_TechnicalExperts: List<Expert> 1
0..* -_PublishingExperts: List<Expert> 1


Expert
-_Name: string
-_TechnicalExpertise : List<string>
-_PublishinglExpertise : List<string>
-_Projects : List<Publication>
+<<property>> Name: string { readOnly}
+<<property>> TechnicalExpertise : List<string>{ readOnly}
+<<property>> PublishinglExpertise : List<string>readOnly}
+<<property>> Projects : List<Publication>readOnly}
+Expert(name: string)
~Expert()
+AddTechnicalExpertise(value: string): void
+AddPublishingExpertise(value: string): void
+AddProject(value: Publication): void
+ToString(): string
Publication
-_Name: string
-_ TechnicalExperts: List<Expert>
-_ PublishingExperts : List<Expert>
+<<property>> Name: string { readOnly}
+<<property>> TechnicalExperts: List<Expert>{ readOnly}
+<<property>> PublishingExperts : List<Expert>{ readOnly}
+Publication(name: string)
~Publication()
+AddTechnicalExpertise(value: Expert): void
+AddPublishingExpertise(value: Expert): void
+GetSummary(): string
+ToString(): string
Publisher
-_Name: string
-_ ExpertList: List<Expert>
-_ ProjectList : List<Publication>
+<<property>> Name: string { readOnly}
+<<property>> ExpertList: List<Expert>readOnly}
+<<property>> ProjectList : List<Publication><Expert>{ readOnly}
+Publisher(name: string)
~Publisher()
+AddExpert(value: Expert): void
+AddProject(value: Publication): void
+FindTechnicalExpertise(value: string):Expert[]
+FindPublishingExpertise(value: string):Expert[]
+ToString(): string

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
namespace PublisherExperts
{
	class Expert
	{
		private string _Name;
		private List<string> _TechnicalExpertise = new List<string>();
		private List<string> _PublishingExpertise = new List<string>();
		private List<Publication> _Projects = new List<Publication>();

		public string Name
		{		get { return _Name; }		}
		public ReadOnlyCollection<string> TechnicalExpertise
		{
			get { return _TechnicalExpertise.AsReadOnly(); }
		}
		public ReadOnlyCollection<string> PublishingExpertise
		{
			get { return _PublishingExpertise.AsReadOnly(); }
		}
		public ReadOnlyCollection<Publication> Projects
		{
			get { return _Projects.AsReadOnly(); }
		}
		public Expert(string name)
		{
			_Name = name;
		}
		// optional clean up aggregation data members
		~Expert()
		{
			if (_Expertise.Count > 0) _Expertise.Clear();
			if (_TechnicalExpertise.Count > 0) _TechnicalExpertise.Clear();
			if (_Projects.Count > 0) _Projects.Clear();
		}
		public void AddTechnicalExpertise(string value)
		{
		       int i;
			for(i = 0; i < _TechnicalExpertise.Count; i++)
			   if(_TechnicalExpertise.ToUpper() == value.ToUpper()) break;
			if(i == _TechnicalExpertse.Count) _TechnicalExpertise.Add(value);
			else Console.WriteLine("{0} already recorded", value);
		}
		public void AddPublishingExpertise(string value)
		{
			int i;
			for(i = 0; i < _ PublishingExpertise.Count; i++)
			   if(_PublishingExpertise.ToUpper() == value.ToUpper()) break;
			if(i == _ PublishingExpertise.Count) _ PublishingExpertise.Add(value);
			else Console.WriteLine("{0} already recorded", value);
		}
```

```
                public void AddProject(Publication value)
                {
                        int i;
                        for(i = 0; i < _ Projects.Count; i++)
                           if(_Projects.Name.ToUpper() == value.NameToUpper()) break;
                        if(i == _ Projects.Count) _ Projects.Add(value);
                        else Console.WriteLine("{0} already recorded", value);
                }
                public override string ToString()
                {
                        return _Name;
                }
        }
}

using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Text;



namespace PublisherExperts
{
   class Publication
   {
      private string _Name;
      private List<Expert> _TechnicalExperts = new List<Expert>();
      private List<Expert> _PublishingExperts = new List<Expert>();
      public string Name
      {
         get { return _Name; }
         set { _Name = value; }
      }
      public ReadOnlyCollection<Expert> TechnicalExperts
      {
         get { return _TechnicalExperts.AsReadOnly(); }
      }
      public ReadOnlyCollection<Expert> PublishingExperts
      {
         get { return _PublishingExperts.AsReadOnly(); }
      }
      public Publication(string name)
      {
         _Name = name;
      }
     // optional clean up aggregation data members
     ~ Publication()
     {
        if (_TechnicalExperts.Count > 0) _ TechnicalExperts.Clear();
        if (_PublishingExperts.Count > 0) _ PublishingExperts.Clear();
     }
```

```
        public void AddTechnicalExpert(Expert value)
        {
            int i;
            for(i = 0; i < _ TechnicalExperts.Count; i++)
                if(_TechnicalExperts.Name.ToUpper() == value.NameToUpper()) break;
            if(i == _ TechnicalExperts.Count) _ TechnicalExperts.Add(value);
            else Console.WriteLine("{0} already recorded", value);
        }
        public void AddPublishingExpert(Expert value)
        {
            int i;
            for(i = 0; i < _ PublishingExperts.Count; i++)
                if(_PublishingExperts.Name.ToUpper() == value.NameToUpper()) break;
            if(i == _ PublishingExperts.Count) _ PublishingExperts.Add(value);
            else Console.WriteLine("{0} already recorded", value);

        }
        public string GetSummary()
        {
            StringBuilder sb = new StringBuilder();

            sb.AppendLine(_Name);
            sb.AppendLine("\tTechnical experts:");
            foreach(Expert e in _TechnicalExperts)
                sb.AppendLine(string.Format("\t\t{0}", e));
            sb.AppendLine("\tPublishing experts:");
            foreach (Expert e in _PublishingExperts)
                sb.AppendLine(string.Format("\t\t{0}", e));

            return sb.ToString();
        }
        public override string ToString()
        {
             return _Name;
        }
    }
}


using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;


namespace PublisherExperts
{
    class Publisher
    {
        private string _Name;
        List<Expert> _ExpertList = new List<Expert>();
        List<Publication> _ProjectList = new List<Publication>();

        public string Name
        {
            get { return _Name; }       set { _Name = value; }
        }
```

```
        public ReadOnlyCollection<Expert> ExpertList
        {
            get { return _ExpertList.AsReadOnly(); }
        }
        public ReadOnlyCollection<Publication> ProjectList
        {
            get { return _ProjectList.AsReadOnly(); }
        }
        public Publisher(string name)
        {
            _Name = name;
        }
        ~ Publisher()
        {
                    if (_ExpertList.Count > 0) _ ExpertList.Clear();
                    if (_ProjectList.Count > 0) _ ProjectList.Clear();
        }
        public void AddExpert(Expert value)
        {
            int i;
            for(i = 0; i < _ ExpertList.Count; i++)
                if(_ExpertList.Name.ToUpper() == value.NameToUpper()) break;
            if(i == ExpertList.Count) _ ExpertList.Add(value);
            else Console.WriteLine("{0} already recorded", value);
        }
        public void AddProject(Publication value)
        {
            int i;
            for(i = 0; i < _ ProjectList.Count; i++)
                if(_ProjectList.Name.ToUpper() == value.NameToUpper()) break;
            if(i == ProjectList.Count) _ ProjectList.Add(value);
            else Console.WriteLine("{0} already recorded", value);
        }
        public Expert[] FindTechnicalExpertise(string value)
        {
            List<Expert> list = new List<Expert>();
            foreach (Expert e in _ExpertList)
                if (e.TechnicalExpertise.Contains(value))    list.Add(e);
            return list.ToArray();
        }
        public Expert[] FindPublishingExpertise(string value)
        {
            List<Expert> list = new List<Expert>();
            foreach (Expert e in _ExpertList)
                if (e.PublishingExpertise.Contains(value))   list.Add(e);
            return list.ToArray();
        }
        public override string ToString()
        {
            return _Name;
        }
    }
}
```

```
using System;
namespace PublisherExperts
{
   class Program
   {
      static void Main(string[] args)
      {
         Publisher pub = new Publisher("TechniPress");

         Expert expert1 = new Expert("Expert 1");
         expert1.AddTechnicalExpertise("programming");
         expert1.AddTechnicalExpertise("operating systems");
         expert1.AddTechnicalExpertise("web site design");
         expert1.AddPublishingExpertise("editor");
         expert1.AddPublishingExpertise("proofreader");
         pub.AddExpert(expert1);

         Expert expert2 = new Expert("Expert 2");
         expert2.AddTechnicalExpertise("programming");
         expert2.AddTechnicalExpertise("machine learning");
         expert2.AddTechnicalExpertise("bioinformatics");
         expert2.AddPublishingExpertise("technical reviewer");
         expert2.AddPublishingExpertise("proofreader");
         pub.AddExpert(expert2);

         Expert expert3 = new Expert("Expert 3");
         expert3.AddTechnicalExpertise("web site design");
         expert3.AddTechnicalExpertise("operating systems");
         expert3.AddTechnicalExpertise("machine learning");
         expert3.AddPublishingExpertise("editor");
         expert3.AddPublishingExpertise("proofreader");
         pub.AddExpert(expert3);

         foreach (Expert e in pub.FindTechnicalExpertise("programming"))     Console.WriteLine("Expert in programming: {0}", e);

         foreach (Expert e in pub.FindTechnicalExpertise("operating systems"))
            Console.WriteLine("Expert in operating systems: {0}", e);
         foreach (Expert e in pub.FindTechnicalExpertise("web site design"))
            Console.WriteLine("Expert in web site design: {0}", e);
         foreach (Expert e in pub.FindTechnicalExpertise("machine learning"))
            Console.WriteLine("Expert in machine learning: {0}", e);
         foreach (Expert e in pub.FindTechnicalExpertise("bioinformatics"))  Console.WriteLine("Expert in bioinformatics: {0}", e);
         foreach (Expert e in pub.FindPublishingExpertise("editor"))     Console.WriteLine("Editor: {0}", e);
         foreach (Expert e in pub.FindPublishingExpertise("technical reviewer"))   Console.WriteLine("Technical reviewer: {0}", e);
         foreach (Expert e in pub.FindPublishingExpertise("proofreader"))   Console.WriteLine("Proofreader: {0}", e);

         Publication book = new Publication("Programming Made Easy");
         book.AddTechnicalExpert(expert1);
         book.AddPublishingExpert(expert2);
         book.AddPublishingExpert(expert3);
         Console.Write(book.GetSummary());
      }
   }
}
```

Task 8.2

algorithmic decomposition

The process of breaking a problem down into its consituent steps, i.e., a program is seen as a sequence of steps which are broken down into smaller steps, each of which is further broken into smaller steps;

candidate objects

The list of possible objects, identified in the problem domain, that could be used to construct an object-oriented application;

imperative programming languages

Also known as procedural programming languages, are languages used for developing applications by exploiting algorithmic decomposition;

object behaviour

Those parts of an object that define how it acts upon other objects and reacts to other objects in an object-oriented application (behaviour is mostly represented as methods defined in a class in a C# program);

object identity

Those parts of an object that allow two identical objects to be distinguished from one another, e.g., two fridges sitting next to each other that are the same brand, model, production lot, etc. are identical, yet they have separate identity (identity is mostly represented by variable names/references in a C# program);

object state

Those parts of an object that represent information that is stored in the object and/or define the operations that are possible (by defining the object's state), e.g., how much money is inserted in a vending machine, whether or not enough money is inserted to vend a particular product, etc. (state is mostly represented by instance variables and properties in a C# program);

object-oriented analysis

The process of analysing/examining a problem to determine the requirements for an application by identifying the candidate objects in the system, what information they might store, what operations might be possible on those objects, and how they relate to each other;

object-oriented decomposition

The process of breaking a problem down into its consituent parts (objects) as part of designing an object-oriented application;

object-oriented design

The process of converting the requirements for an application identified through the analysis phase into a blueprint (design) for an object-oriented application;