using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

namespace ExamSystem

{

    public enum ExamMode { Starting, Queued, Finished }

    public class Subject

    {

        public string Code { get; set; }

        public string Name { get; set; }

        public Subject(string code, string name)

        {

            Code = code;

            Name = name;

        }

        public override string ToString() => $"{Code} - {Name}";

    }

    public class Answer

    {

        public string Text { get; set; }

        public bool IsCorrect { get; set; }

        public Answer(string text, bool isCorrect = false)

        {

            Text = text;

            IsCorrect = isCorrect;

        }

        public override string ToString() => Text;

    }

    public class AnswerList : List<Answer> { }

    public abstract class Question : ICloneable, IComparable<Question>

    {

        public string Header { get; set; }

        public string Body { get; set; }

        public int Marks { get; set; }

        public AnswerList Answers { get; set; } = new();

        protected Question(string header, string body, int marks)

        {

            Header = header;

            Body = body;

            Marks = marks;

        }

        public abstract void Show();

        public object Clone() => MemberwiseClone();

        public int CompareTo(Question other) => Marks.CompareTo(other.Marks);

        public override string ToString() => $"{Header}: {Body} ({Marks} marks)";

        public override bool Equals(object obj) =>

            obj is Question q && Header == q.Header && Body == q.Body && Marks == q.Marks;

        public override int GetHashCode() => HashCode.Combine(Header, Body, Marks);

    }

    public class TrueFalseQuestion : Question

    {

        public TrueFalseQuestion(string header, string body, int marks)

            : *base*(header, body, marks) { }

        public override void Show()

        {

            Console.WriteLine(ToString());

            Console.WriteLine("  a) True\n  b) False");

        }

    }

    public class ChooseOneQuestion : Question

    {

        public ChooseOneQuestion(string header, string body, int marks)

            : *base*(header, body, marks) { }

        public override void Show()

        {

            Console.WriteLine(ToString());

            for (int i = 0; i < Answers.Count; i++)

                Console.WriteLine($"  {i + 1}) {Answers[i]}");

        }

    }

    public class ChooseAllQuestion : Question

    {

        public ChooseAllQuestion(string header, string body, int marks)

            : *base*(header, body, marks) { }

        public override void Show()

        {

            Console.WriteLine(ToString() + " (choose all that apply)");

            for (int i = 0; i < Answers.Count; i++)

                Console.WriteLine($"  {i + 1}) {Answers[i]}");

        }

    }

    public class QuestionList : List<Question>

    {

        private readonly string \_logFile;

        public QuestionList(string logFile)

        {

            \_logFile = logFile;

        }

        public new void Add(Question q)

        {

*base*.Add(q);

            using var writer = new StreamWriter(\_logFile, append: true);

            writer.WriteLine($"{DateTime.Now}: {q}");

        }

    }

    public abstract class Exam : ICloneable, IComparable<Exam>

    {

        public Subject Subject { get; set; }

        public TimeSpan Time { get; set; }

        public ExamMode Mode { get; private set; }

        public Dictionary<Question, Answer> QuestionAnswer { get; set; } = new();

        public event Action<string> ExamStarting;

        protected Exam(Subject subject, TimeSpan time)

        {

            Subject = subject;

            Time = time;

        }

        public abstract void ShowExam();

        public void Start()

        {

            Mode = ExamMode.Starting;

            ExamStarting?.Invoke($"Exam for {Subject.Name} is starting now!");

        }

        public void Queue() => Mode = ExamMode.Queued;

        public void Finish() => Mode = ExamMode.Finished;

        public object Clone() => MemberwiseClone();

        public int CompareTo(Exam other) => Time.CompareTo(other.Time);

        public override string ToString() => $"{Subject} | Duration: {Time}";

        public override bool Equals(object obj) =>

            obj is Exam e && Subject.Equals(e.Subject) && Time.Equals(e.Time);

        public override int GetHashCode() => HashCode.Combine(Subject, Time);

    }

    public class PracticeExam : Exam

    {

        public PracticeExam(Subject subject, TimeSpan time) : *base*(subject, time) { }

        public override void ShowExam()

        {

            Console.WriteLine($"Practice Exam - {Subject}");

            foreach (var kvp in QuestionAnswer)

            {

                kvp.Key.Show();

                Console.WriteLine($"Correct Answer: {kvp.Value}\n");

            }

        }

    }

    public class FinalExam : Exam

    {

        public FinalExam(Subject subject, TimeSpan time) : *base*(subject, time) { }

        public override void ShowExam()

        {

            Console.WriteLine($"Final Exam - {Subject}");

            foreach (var kvp in QuestionAnswer)

            {

                kvp.Key.Show();

                Console.WriteLine();

            }

        }

    }

    class Program

    {

        static void Main()

        {

            var subject = new Subject("CS101", "Object Oriented Programming");

            var practiceExam = new PracticeExam(subject, TimeSpan.FromMinutes(60));

            var finalExam = new FinalExam(subject, TimeSpan.FromMinutes(60));

            practiceExam.ExamStarting += msg => Console.WriteLine("[Notification] " + msg);

            finalExam.ExamStarting += msg => Console.WriteLine("[Notification] " + msg);

            var q1 = new TrueFalseQuestion("Q1", "C# is object oriented?", 5);

            q1.Answers.Add(new Answer("True", true));

            q1.Answers.Add(new Answer("False"));

            practiceExam.QuestionAnswer.Add(q1, q1.Answers.First(a => a.IsCorrect));

            finalExam.QuestionAnswer.Add((Question)q1.Clone(), q1.Answers.First(a => a.IsCorrect));

            Console.Write("Select Exam Type (1-Practice, 2-Final): ");

            var choice = Console.ReadLine();

            if (choice == "1")

            {

                practiceExam.Start();

                practiceExam.ShowExam();

                practiceExam.Finish();

            }

            else

            {

                finalExam.Start();

                finalExam.ShowExam();

                finalExam.Finish();

            }

        }

    }

}