Лабораторная работа №8.

Эта лабораторная продолжает задания из 7й лабораторной. Т.е. перед выполнением задания Вам нужно скопировать код из 7й лабораторной работы.

Решения должны находиться в файле с названием, соответствующем его номеру внутри папки с названием вашей лиги. Решение номеров необходимо писать внутри класса с соответствующим номером (Task1 – Task5). Автотестами будет сравниваться состояние Ваших объектов с выходными данными.

Все поля должны быть приватными! Все поля необходимо инициализировать в конструкторе! В свойствах и методах перед обращением к ссылочным полям проверять их на null. Вам нужно решить задания в точности так, как указано в условии. Для выполнения задания вы можете использовать дополнительные приватные поля, свойства или методы.

Если все тесты (автоматические и ручные) пройдены успешно, Вы можете отправить лабораторную на заключительную проверку на GitHub. Более подробная инструкция описана в задании на Moodle.

Если Ваша работа принята, вы можете прорешать номера из других лиг в качестве подготовки к контрольной. Не обязательно все, а только те, которые не похожи на то, что было в Вашей лиге. Рекомендуется так сделать для тех, кто не имеет опыта в программировании и был определен в белую лигу, потому что для успешного освоения курса Вам нужно научиться решать задачи уровня зеленой и частично синей лиги.

В работе разрешены методы следующих классов: *Console, Math, Random, Array, Linq*.

Задания белой лиги.

Task1. Переделать структуру **Participant** в класс.

Создать статические поля норматива, количество активных и дисквалифицированных участников **_standard, _jumpers, _disqualified** соответственно.

Создать статические свойства для чтения новых полей **Jumpers** и **Disqualified** соответственно.

Создать статический конструктор, в котором установить значения статическим полям (нормативу установить значение 5).

Создать метод **public static void Disqualify(ref Participant[] participants)**, который будет дисквалифицировать (удалять из массива) участников, не достигших хотя бы в одном прыжке заданного норматива.

Task2. Переделать структуру **Participant** в класс.

Создать неизменяемое статическое поле для норматива.

Создать свойство для определения статуса участника **IsPassed** – прошел он норматив или нет.

Создать статический конструктор, в котором установить нормативу значение **3**.

Создать ещё один конструктор, который принимает 2 строковых поля для имени и фамилии и 2 вещественных поля для значений прыжков участника.

Создать метод **public static Participant[] GetPassed(Participant[] participants)**, который будет возвращать массив участников, прошедших норматив по результатом лучшей попытки.

Task3. Переделать структуру **Student** в класс. Создать от него класс–наследник **Undergraduate**.

Изменить уровни доступа полей, отвечающих за массив оценок и количество пропусков на **protected**.

Добавить защищенный конструктор в класс **Student**, который принимает объект студента и копирует его значения в соответствующие поля.

Создать в классе **Undergraduate** 2 конструктора: один принимает имя и фамилию и передает их базовому конструктору, второй принимает объект студента и передает его базовому конструктору.

Создать в классе **Undergraduate** метод **public void WorkOff(int mark)**, который позволяет получить новую оценку по предмету вместо одного пропуска. Если пропусков нет, то заменить имеющуюся оценку 2 на новую оценку. Избегать дублирования! Используйте уже имеющиеся метод **Lesson**.

Создать метод **Print()** для вывода информации о необходимых полях класса.

Task4. Создать класс **Human**. Переделать структуру **Participant** в класс, наследующийся от **Human**.

Перенести свойства **Name** и **Surname**, а также связанные с ними поля в класс **Human**.

Создать в классе **Human** конструктор, который принимает 2 строковых значения и заполняет ими поля имени и фамилии.

Создать виртуальный метод **Print()** для вывода информации о необходимых полях класса.

Создать в классе **Participant** статическое поле, подсчитывающее общее количество участников.

Создать в классе **Participant** статическое свойство **Count** только для чтения поля общего количества участников.

В статическом конструкторе инициализировать общее количество участников нулем.

Переопределить метод **Print()** для вывода информации о необходимых полях класса.

Task5. Переделать структуру **Team** в абстрактный класс. Создать от него 2 класса–наследника **ManTeam** и **WomanTeam**.

В классе **Team** сделать метод **PlayMatch** виртуальным.

В классе **ManTeam** создать поле, хранящее в себе ссылку на команду–дерби и свойство для чтения этого поля **Derby**.

В классе **ManTeam** создать конструктор, который принимает строковое значение для названия команды и значение типа **ManTeam** со значением по умолчанию **null**, для поля команды–дерби. Строковое значение для названия передавать базовому конструктору.

В классе **ManTeam** создать метод **public void PlayMatch(int goals, int misses, ManTeam team = null)**, который увеличивает количество забитых мячей в матче на 1, если матч происходит с командой–дерби.

В классе **WomanTeam** создать поле, хранящее в себе массив штрафов, полученных в матчах.

В классе **WomanTeam** создать свойства для чтения этого поля **Penalties** и свойство, выводящее общее количество штрафов, набранных командой **TotalPenalties**.

В классе **WomanTeam** создать конструктор, который принимает строковое значение для названия команды. Строковое значение для названия передавать базовому конструктору. В конструкторе инициализировать массив штрафов.

Переопределить базовый метод **PlayMatch**, добавив к нему проверку: если пропущенных голов больше, чем забитых, добавить новое значение в массив штрафов, равное разнице между пропущенными и забитыми голами.

Задания зеленой лиги.

Task1. Переделать структуру **Participant** в абстрактный класс. Поле норматива сделать нестатическим, изменяемым и повысить уровень доступа до защищенного.

Создать в классе **Participant** метод **public static Participant[] GetTrainerParticipants(Participant[] participants, Type participantType, string trainer)**, который возвращает всех участников выбранного тренера в выбранном забеге.

Создать от класса **Participant** 2 класса–наследника участников забега **Participant100M** и **Participant500M**.

Переопределить в конструкторах классов **Participant100M** и **Participant500M** норматив на 12 и 90 соответственно.

Task2. Создать класс **Human**. Переделать структуру **Student** в класс, наследующийся от **Human**.

Перенести свойства **Name** и **Surname**, а также связанные с ними поля в класс **Human**.

Создать в классе **Human** конструктор, который принимает 2 строковых значения и заполняет ими поля имени и фамилии.

Создать метод **Print()** для вывода информации о необходимых полях класса.

В классе **Student** добавить статическое поле, подсчитывающее общее количество отличников. Поле увеличивать, когда студент получит оценки по всем экзаменам (не ниже “4”).

В классе **Student** добавить статическое свойство **ExcellentAmount** только для чтения поля общего количества отличников.

Task3. Переделать структуру **Student** в класс.

Создать в классе **Student** уникальное поле, которое будет хранить номер студенческого билета (целое число).

Создать в классе **Student** свойство для чтения номера студ. билета **ID**.

Статический конструктор в классе **Student** должен устанавливать поле для номера первого студенческого билета равным 1.

Создать в классе **Student** метод **public void Restore()**, который меняет статус студента с “отчисленного” на “не отчисленного”.

Создать класс **Commission**.

Создать в классе **Commission** метод **public static void Sort(Student[] students)**, который сортирует студентов по номеру их студ. билета.

Создать в классе **Commission** метод **public static Student[] Expel(ref Student[] students)**, который возвращает массив исключенных студентов. При этом в переданном массиве должны остаться только действующие студенты.

Создать в классе **Commission** метод **public static void Restore(ref Student[] students, Student restored)**, который позволяет добавить студента в массив на место в соответствии с номером его студ. билета. Избегать дублирования записей! Также нельзя восстановить студента, которого изначально в списке не было (нового).

Task4. Создать абстрактный класс **Discipline** с обязательными приватными полями названия и массива участников.

Создать в классе свойства для чтения приватных полей **Name** и **Participants** соответственно.

В защищенном конструкторе (protected) передавать название дисциплины и сохранять его. Также инициализировать массив участников.

Создать методы **public void Add**, которые добавляют одного и несколько новых участников в список.

Создать метод **public void Sort()**, который сортирует участников данной дисциплины, используя статический метод структуры **Participant**.

Создать абстрактный метод **public abstract void Retry(int index)**, который позволяет выбранному участнику соревнований прыгнуть ещё раз (с определенным условием).

Создать метод **Print()** для вывода информации о необходимых полях класса.

Создать от **Discipline** два класса–наследника: **LongJump** и **HighJump**. В каждом из классов переопределить базовый конструктор и с его помощью передавать название дисциплины базовому классу как "Long jump" и "High jump" соответственно.

В классе **LongJump** переопределить метод **Retry** так, чтобы для выбранного из списка участника сохранялся результат лучшего прыжка как 1й результат и добавлялись две свободные попытки.

В классе **HighJump** переопределить метод **Retry** так, чтобы для выбранного из списка участника сбрасывался результат его последнего прыжка (с возможностью повторить попытку).

Task5. Переделать структуру **Group** в класс.

Свойство **AverageMark** сделать виртуальным.

Создать от **Group** два класса-наследника: **EliteGroup** и **SpecialGroup**.

Переопределить свойство **AverageMark** в классе **EliteGroup** следующим образом: каждая оценка имеет свой “вес”. Для 5 – это 1, для 4 – это 1.5, для 3 – это 2, для 2 – это 2.5. Свойство должно возвращать средний балл участников с учетом весовых коэффициентов. Если оценок у участника нет – его не учитывать.

Заменить свойство **AverageMark** в классе **SpecialGroup** следующим образом: каждая оценка имеет свой “вес”. Для 5 – это 1, для 4 – это 0.75, для 3 – это 0.5, для 2 – это 0.25. Свойство должно возвращать средний балл участников с учетом весовых коэффициентов. Если оценок у участника нет – его не учитывать.

Задания синей лиги.

Task1. Переделать структуру **Response** в класс. Перенести в него поле, отвечающее за имя (наименование) и связанное с ним свойство. Аналогично перенести поле (и связанное с ним свойство), отвечающее за количество голосов и изменить у него уровень доступа на **protected**.

Создать в классе **Response** метод **CountVotes** виртуальным, изменив его так, чтобы он подсчитывал ответы с одинаковыми наименованиями. Метод **Print** сделать виртуальным.

Конструктор класса **Response** сократить, чтобы он принимал одно строковое поле для наименования.

Создать от класса **Response** класс–наследник **HumanResponse**.

Конструктор класса **HumanResponse** должен принимать 2 строковых поля: имя и фамилию. Имя он должен передавать базовому классу.

Переопределить метод **CountVotes** в классе **HumanResponse**, реализовав его прежнюю логику. Использовать преобразование типов.

Переопределить метод **Print()** в классе **HumanResponse** для вывода информации о всех необходимых полях.

Task2. Создать абстрактный класс **WaterJump** с обязательными полями для названия турнира, призового фонда и массива участников.

Создать свойства для чтения приватных полей **Name, Bank** и **Participants**.

Создать абстрактное свойство **Prize**, которое возвращает суммы для участников, занявших призовые места (вещественные числа) в порядке занятых мест.

Защищенный конструктор (protected) должен принимать 2 поля: название турнира и размер призового фонда (целое число). Также он должен инициализировать массив участников.

Создать методы **public void Add**, которые добавляют одного и несколько новых участников в список.

Создать классы–наследники **WaterJump3m** и **WaterJump5m**.

В классе **WaterJump3m** переопределить свойство **Prize**, чтобы 50% от фонда выделяется участнику, занявшему первое место, 30% – второе место, 20% – третье место. Если участников менее 3-х, ничего не выдавать.

В классе **WaterJump5m** переопределить свойство **Prize**, чтобы по N% уходило всем участникам, оказавшимся выше середины таблицы (но не менее 3-х и не более 10), где N – это 20 / количество участников, оказавшихся выше середины таблицы. Также 40% от фонда выделяется участнику, занявшему первое место, 25% – второе место, 15% – третье место. Если участников менее 3-х, ничего не выдавать.

Task3. Переделать структуру **Participant** в класс. Уровень доступа поля, хранящего массив штрафов повысить до **protected**. Свойство **IsExpelled** сделать виртуальными, свойства **PenaltyTimes** и **TotalTime** переименовать в **Penalties** и **Total** соответственно. Метод **PlayMatch** сделать виртуальным.

Создать классы–наследники **BasketballPlayer** и **HockeyPlayer**.

В классе **BasketballPlayer** переопределить необходимые свойства и метод таким образом, чтобы игрок исключался из кандидатов в сборную в следующих случаях: он сыграл более 10% матчей с 5-ю фолами или у него суммарное количество фолов вдвое больше, чем количество матчей.

В метод **PlayMatch** будут подаваться не минуты штрафов, а количество фолов за матч в диапазоне от 0 до 5 включительно.

В классе **HockeyPlayer** добавить статические поля **_players** и **_totalTime** для подсчета времени и количества всех хоккеистов соответственно. Переопределить необходимые свойства и метод таким

образом, чтобы игрок исключался из кандидатов в сборную в следующих случаях: хотя бы в одном матче у него было 10 минут штрафного времени или же его суммарное штрафное время больше, чем 10% от суммарного штрафного времени всех хоккеистов, разделенного на количество хоккеистов.

Task4. Переделать структуру **Team** в абстрактный класс.

Создать классы–наследники **ManTeam** и **WomanTeam**.

Переделать структуру **Group** в класс. Поле и связанное с ним свойство **Teams** переименовать в **ManTeams**. Создать аналогичные поле и свойство **WomanTeams**. В конструкторе инициализировать оба поля пустыми массивами из 12 команд.

Изменить методы **Add** так, чтобы они проверяли входящую команду, является ли она мужской или женской, и заполняли соответствующий массив команд новыми командами.

Изменить метод **Sort**, чтобы он сортировал оба массива по отдельности. Избегать дублирования кода!

Изменить метод **Merge**, чтобы он соединял с сохранением упорядоченности мужские и женские команды в соответствующие массивы финалистов отдельно. Избегать дублирования кода!

Task5. Переделать структуру **Sportsman** в класс. Переделать структуру **Team** в абстрактный класс.

Создать классы–наследники **ManTeam** и **WomanTeam**.

Создать в классе **Team** метод **protected abstract double GetTeamStrength()**, который рассчитывает “силу команды”.

Создать в классе **Team** метод **public static Team GetChampion(Team[] teams)**, который выводит команду–чемпиона по силе среди мужских и женских команд.

В классе **ManTeam** переопределить метод **GetTeamStrength**, чтобы он рассчитывал силу команды как 100 / среднее значение их мест.

В классе **WomanTeam** переопределить метод **GetTeamStrength**, чтобы он рассчитывал силу команды как 100 * сумму мест, умноженную на число участников в команде и деленную на произведение их мест.

Задания фиолетовой лиги.

Task1. Переделать структуру **Participant** в класс.

Создать класс **Judge** с полями для имени и массива оценок, которые этот судья любит ставить участникам.

Создать свойство **Name** для чтения поля имени.

Конструктор должен получать имя и массив оценок.

Создать метод **public int CreateMark()**, который при вызове возвращает очередную оценку из массива. Если он дошёл до конца массива, то начинает с начала.

Создать метод **Print()** в классе для вывода информации о необходимых полях.

Создать класс **Competition** с приватными полями судей и участников.

Создать свойства **Judges** и **Participants** для чтения полей судей и участников соответственно.

Конструктор должен получать массив судей и инициализировать массив участников.

Создать метод **public void Evaluate(Participant jumper)**, который передает участнику оценки за прыжок от всех судей.

Создать методы **Add**, которые получают одного или несколько участников и добавляют их в хранимый массив участников. При добавлении участнику сразу нужно заполнить оценки за один прыжок от всех судей.

Создать метод **public void Sort()**, который сортирует участников соревнования.

Избегать дублирования! При написании методов используйте уже написанные методы.

Task2. В структуре **Participant** изменить метод **public void Jump(int distance, int[] marks)** на: **public void Jump(int distance, int[] marks, int target),** где **target** – та дистанция, на которую нужно прыгнуть спортсмену, чтобы получить 60 дополнительных очков.

Создать абстрактный класс **SkiJumping** с полями для названия соревнования, норматива дистанции и списка участников.

Создать свойства **Name, Standard** и **Participants** для чтения приватных полей.

Конструктор должен получать название соревнования и норматив дистанции, а также инициализировать массив участников.

Создать методы **Add**, которые получают одного или несколько участников и добавляют их в хранимый массив участников.

Создать метод **public void Jump(int distance, int[] marks)**, который вызывает у первого ещё не прыгнувшего спортсмена метод **Jump** и передает ему полученные данные. Расчет дополнительных баллов должен зависеть от норматива соревнования.

Создать метод **Print()** в классе для вывода информации о необходимых полях.

Создать классы–наследники от **SkiJumping**: **JuniorSkiJumping** и **ProSkiJumping**.

Переопределить конструктор по умолчанию в классе **JuniorSkiJumping**, чтобы он передавал в базовый класс название “100m” и дистанцию 100м.

Переопределить конструктор по умолчанию в классе **ProSkiJumping**, чтобы он передавал в базовый класс название “150m” и дистанцию 150м.

Task3. Создать абстрактный класс **Skating** с полями для списка участников и массива настроений судей (вещественные значения) с уровнем доступа **protected**.

Создать свойство **Participants** для чтения поля участников и свойство **Moods** для чтения настроения судей.

Конструктор должен получать массив настроений судей, заполнять данные для каждого судьи (их 7) и сразу менять данные массива, вызывая метод **ModificateMood**.

Создать метод **protected abstract void ModificateMood()**, который будет изменять настроение судей в разных соревнованиях.

Создать метод **public void Evaluate(double[] marks)**, который будет вызывать у первого ещё не выступившего спортсмена метод **Evaluate** и передавать ему полученные оценки, умноженные на настроение соответствующего судьи.

Создать методы **Add**, которые получают одного или несколько участников и добавляют их в хранимый массив участников.

Создать классы–наследники от **Skating**: **FigureSkating** и **IceSkating**.

Их конструкторы получают массив настроений и передают его базовому классу.

Переопределить метод **ModificateMood** в классе **FigureSkating** следующим образом: к настроению добавляется порядковый номер судьи, деленный на 10.

Переопределить метод **ModificateMood** в классе **IceSkating** следующим образом: настроение судьи увеличивается на i%, где i – порядковый номер судьи.

Task4. Переделать структуру **Sportsman** в класс.

Создать в классе **Sportsman** метод **public static void Sort(Sportsman[] array)**, который сортирует массив спортсменов по возрастанию времени забега.

Создать классы–наследники от **Sportsman**: **SkiMan** и **SkiWoman.**

В обоих классах–наследниках сделать по 2 конструктора: первый принимает имя и фамилию, а второй – имя, фамилию и время забега. Во втором конструкторе вызывать метод **Run** для выставления времени.

Переделать структуру **Group** в класс.

Создать в классе **Group** метод **public void Split(out Sportsman[] men, out Sportsman[] women)**, который разделяет массив спортсменов группы на массивы лыжников и лыжниц.

Создать в классе **Group** метод **public void Shuffle()**, который сортирует спортсменов группы по времени и выстраивает в порядке: лыжник, лыжница, лыжник, лыжница и т д. начиная от самого быстрого спортсмена или спортсменки.

Task5. Создать класс **Report** с полем для массива исследований и уникальным полем для нумерации будущих исследований.

Создать в классе **Report** свойство **Researches** для чтения поля исследований.

Статический конструктор должен инициализировать нумерацию исследований с единицы.

Конструктор должен инициализировать пустой массив исследований.

Создать в классе **Report** метод **public Research MakeResearch()**, который начинает новое исследование и называет его в следующем виде: No_X_MM/YY, где X – порядковый номер исследования, MM – месяц создания отчета (брать из DateTime) YY – последние 2 цифры года создания отчета (брать из DateTime). Исследование добавлять в список и возвращать на выходе из метода.

Создать в классе **Report** метод **public (string, double)[] GetGeneralReport(int question)**, который возвращает массив кортежей данных, собранный среди всех исследований по выбранному вопросу, в виде: ответ + % этих ответов от общего количества (непустых) ответов.