ОГБПОУ «ТОМСКИЙ ТЕХНИКУМ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ»

Утверждаю

Заместитель директора по УМР

\_\_\_\_\_\_\_\_\_\_\_\_/Е.А. Родзик

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

Методические рекомендации по выполнению   
учебной-практической работы №4

учебной практики   
*УП 05. Программирование*

*Тема: «Конструкторы»*

г. Томск – 2020 г

РАССМОТРЕННО

на заседании ПЦК

«Информационные системы и программирование»

протокол №\_\_\_\_\_\_

от «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

Председатель ПЦК

\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Фунтиков М.Н.Рекомендации разработаны на основе Федерального государственного образовательного стандарта (далее – ФГОС) по специальностям среднего профессионального образования (далее СПО) 09.02.07 – «Информационные системы и программирование» и в соответствии с примерной основной образовательной программой.

Организация-разработчик:

ОГБОУ СПО «Томский техникум информационных технологий»

Разработчики:

\_\_\_\_\_\_\_\_\_\_\_\_\_/ Сидиков И.Д.

\_\_\_\_\_\_\_\_\_\_\_\_\_/ Владимировна А. В.

**СОДЕРЖАНИЕ**

[1. Общая характеристика учебной-практической работы №4 4](#_Toc22557890)

[2. Краткие теоретические сведения 5](#_Toc22557891)

[3. Задания для выполнения 25](#_Toc22557892)

[4. Индивидуальные задания 26](#_Toc22557893)

[5. Контрольные вопросы 27](#_Toc22557894)

[6. Рекомендованная литература 28](#_Toc22557895)

[ПРИЛОЖЕНИЕ А 29](#_Toc22557896)

[ПРИЛОЖЕНИЕ Б 30](#_Toc22557897)

# ОБЩАЯ ХАРАКТЕРИСТИКА УЧЕБНОЙ-ПРАКТИЧЕСКОЙ РАБОТЫ №4

**Тема:** «Методы».

**Цель работы:** получение первоначальных навыков по работе с конструкторами на языке программирования C#

**Проверяемые компетенции:**

ОК 1, ОК 2, ОК 4, ОК 9, ПК11.2.

**Инструкция по выполнению:**

1. Ознакомьтесь с теоретическим материалом, необходимым для выполнения практической работы.
2. Выполните предложенное практическое задание.
3. Оформите отчет по учебной практике (шаблон представлен в приложении)
4. Отправьте отчет на проверку.

**Время выполнения заданий:** 6 часов.

**Критерии оценки:**

|  |  |  |
| --- | --- | --- |
| **№** | **Критерий** | **Баллы** |
| 1 | Отчет соответствует предложенному шаблону | 1 |
| 2 | Задание выполнено правильно | 2 |
| 3 | В коде присутствует функции | 4 |
| 4 | Составлена блок схема к программе | 2 |
| 5 | Программа работает правильно без сбоев | 2 |
| 6 | Студент ответил на контрольные вопросы по лабораторной работе | 2 |
| 7 | Оформление отчета соответствует требованиям (шрифт, поля, отступы, интервалы, оформление рисунков, автоматическое оглавление) | 1 |
| 8 | Своевременность выполнения задания | 1 |
| Итого | | 15 |

**Перевод в пятибалльную систему оценивания:**

«отлично» - 13 - 15 баллов

«хорошо» - 10 -12 баллов

«удовлетворительно» - 7-9

«неудовлетворительно» < 7 баллов

# КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

## Введение конструкторы

Каждый раз, когда создается класс или структура, вызывается конструктор. Класс или структура может иметь несколько конструкторов, принимающих различные аргументы. Конструкторы позволяют программисту задавать значения по умолчанию, ограничивать число установок и писать код, который является гибким и удобным для чтения.

### Конструкторы без параметров

Если не предоставить конструктор для класса, C# создаст конструктор по умолчанию, который создает экземпляр объекта и задает переменным-членам значения по умолчанию, как показано в статье Значения по умолчанию типов C#. Если не предоставить конструктор для структуры, C# будет использовать неявный конструктор без параметров, чтобы автоматически инициализировать каждое поле значением по умолчанию.

### Синтаксис конструктора

Конструктор — это метод, имя которого совпадает с именем его типа. Его сигнатура метода содержит только имя метода и список параметров. Она не содержит возвращаемый тип. В приведенном ниже примере демонстрируется конструктор для класса с именем Person.

|  |
| --- |
| С# |
| public class Person  {  private string last;  private string first;    public Person(string lastName, string firstName)  {  last = lastName;  first = firstName;  }    // Remaining implementation of Person class.  } |

Если конструктор поддерживает реализацию в виде оператора, можно использовать определение тела выражения. В следующем примере определяется класс Location, конструктор которого имеет один строковый параметр name. Определение тела выражения присваивает аргумент полю locationName.

|  |
| --- |
| С# |
| public class Location  {  private string locationName;    public Location(string name) => Name = name;  public string Name  {  get => locationName;  set => locationName = value;  }  } |

### Статические конструкторы

В приведенных выше примерах показаны конструкторы экземпляров, которые создают новый объект. В классе или структуре также может быть статический конструктор, который инициализирует статические члены типа. Статические конструкторы не имеют параметров. Если вы не предоставили статический конструктор для инициализации статических полей, компилятор C# инициализирует статические поля значениями по умолчанию, как показано в статье Значения по умолчанию типов C#.

В следующем примере статический конструктор используется для инициализации статического поля.

|  |
| --- |
| С# |
| public class Adult : Person  {  private static int minimumAge;    public Adult(string lastName, string firstName) : base(lastName, firstName)  { }  static Adult()  {  minimumAge = 18;  }  // Remaining implementation of Adult class.  } |

Можно также определить статический конструктор с помощью определения тела выражения, как показано в следующем примере.

|  |
| --- |
| С# |
| public class Child : Person  {  private static int maximumAge;    public Child(string lastName, string firstName) : base(lastName, firstName)  { }  static Child() => maximumAge = 18;  // Remaining implementation of Child class.  } |

## Использование конструкторов

Каждый раз, когда создается класс или структура, вызывается конструктор. Конструкторы имеют имя, совпадающее с именем класса или структуры, и обычно инициализируют члены данных нового объекта.

В следующем примере класс с именем Taxi определяется с помощью простого конструктора. Затем оператор new создает экземпляр этого класса. Конструктор Taxi вызывается оператором new сразу после того, как новому объекту будет выделена память.

|  |
| --- |
| С# |
| public class Taxi  {  public bool IsInitialized;  public Taxi()  {  IsInitialized = true;  }  }  class TestTaxi  {  static void Main()  {  Taxi t = new Taxi();  Console.WriteLine(t.IsInitialized);  }  } |

Конструктор, который не принимает никаких параметров, называется конструктором без параметров. Конструкторы без параметров вызываются всякий раз, когда создается экземпляр объекта с помощью оператора new, а аргументы в new не передаются.

Если класс не является статическим, компилятор C# выделяет классам без конструкторов открытый конструктор без параметров, позволяющий создавать экземпляры классов.

Создание экземпляров класса можно запретить, сделав конструктор закрытым, следующим образом:

|  |
| --- |
| С# |
| class NLog  {  // Private Constructor:  private NLog() { }  public static double e = Math.E; //2.71828...  } |

Конструкторы для типов struct похожи на конструкторы классов, однако structs не может содержать явный конструктор без параметров, так как он предоставляется компилятором автоматически. Этот конструктор инициализирует каждое поле в struct со значением по умолчанию. При этом конструктор без параметров вызывается только в том случае, если экземпляр struct создается с помощью переменной new. Например, в этом коде конструктор без параметров используется для Int32 — это обеспечивает инициализацию целого числа:

|  |
| --- |
| С# |
| int i = new int();  Console.WriteLine(i); |

В то же время следующий код вызывает ошибку компилятора, поскольку не использует new, и потому, что использует не инициализированный объект:

|  |
| --- |
| С# |
| int i;  Console.WriteLine(i); |

Кроме того, объекты на основе structs (включая все встроенные числовые типы) можно инициализировать или назначить, а затем использовать, как в следующем примере:

|  |
| --- |
| С# |
| int a = 44; // Initialize the value type...  int b;  b = 33; // Or assign it before using it.  Console.WriteLine("{0}, {1}", a, b); |

В связи с этим вызывать конструктор без параметров для типа значения необязательно.

Оба класса и structs могут определять конструкторы, принимающие параметры. Конструкторы, принимающие параметры, необходимо вызывать с помощью оператора new или base. Классы и structs могут определять также несколько конструкторов; для определения конструктора без параметров ни один их них не требуется. Пример:

|  |
| --- |
| С# |
| public class Employee  {  public int Salary;  public Employee(int annualSalary)  {  Salary = annualSalary;  }  public Employee(int weeklySalary, int numberOfWeeks)  {  Salary = weeklySalary \* numberOfWeeks;  }  } |

Этот класс можно создать, воспользовавшись одним из следующих операторов:

|  |
| --- |
| С# |
| Employee e1 = new Employee(30000);  Employee e2 = new Employee(500, 52); |

Конструктор может использовать ключевое слово base для вызова конструктора базового класса. Пример:

|  |
| --- |
| С# |
| public class Manager : Employee  {  public Manager(int annualSalary)  : base(annualSalary)  {  //Add further instructions here.  }  } |

В этом примере конструктор базового класса вызывается перед выполнением соответствующего ему блока. Ключевое слово base можно использовать как с параметрами, так и без них. Любые параметры для конструктора можно использовать как параметры для base или как часть выражения. Дополнительные сведения см. в разделе base.

В производном классе, если конструктор базового класса не вызывается явным образом с помощью ключевого слова base, конструктор без параметров, если он существует, вызывается неявно. Это означает, что следующие объявления конструкторов действуют одинаково:

|  |
| --- |
| С# |
| public Manager(int initialData)  {  //Add further instructions here.  } |

|  |
| --- |
| С# |
| public Manager(int initialData)  : base()  {  //Add further instructions here.  } |

Если базовый класс не предлагает конструктор без параметров, производный класс должен явно вызвать конструктор базового класса с помощью base.

Конструктор может вызывать другой конструктор в том же объекте с помощью ключевого слова this. Как и base, this можно использовать с параметрами или без, а все параметры в конструкторе доступны как параметры this или как часть выражения. Например, второй конструктор в предыдущем примере можно переписать, используя this:

|  |
| --- |
| С# |
| public Employee(int weeklySalary, int numberOfWeeks)  : this(weeklySalary \* numberOfWeeks)  {  } |

Применение ключевого слова this в приведенном выше примере привело к вызову конструктора:

|  |
| --- |
| С# |
| public Employee(int annualSalary)  {  Salary = annualSalary;  } |

Конструкторы могут иметь пометку public, private, protected, internal, protected internal или private protected. Эти модификаторы доступа определяют, каким образом пользователи класса смогут создавать класс.

Конструктор можно объявить статическим, используя ключевое слово static. Статические конструкторы вызываются автоматически непосредственно перед доступом к статическим полям и обычно используются для инициализации членов статического класса.

## Конструкторы экземпляров

Конструкторы экземпляров используются для создания и инициализации переменных члена экземпляра, если создание объекта class осуществляется с помощью выражения new. Для инициализации класса static или статических переменных в нестатическом классе определяется статический конструктор.

В следующем примере показан конструктор экземпляра.

|  |
| --- |
| С# |
| class Coords  {  public int x, y;  // constructor  public Coords()  {  x = 0;  y = 0;  }  } |

|  |
| --- |
| Примечание!  Для ясности этот класс содержит открытые поля. Открытые поля не рекомендуется использовать на практике, поскольку в этом случае любой метод в любом месте программы получает неограниченный и неконтролируемый доступ к внутренней работе объекта. Члены данных обычно должны быть закрытыми, а доступ к ним должен осуществляться только посредством методов и свойства класса. |

Этот конструктор экземпляра вызывается каждый раз при создании объекта на базе класса Coords. Такой конструктор без аргументов называется конструктором без параметров. Зачастую такие конструкторы используются для предоставления дополнительных конструкторов.

Например, можно добавить конструктор в класс Coords, позволяющий указывать начальные значения для членов данных:

|  |
| --- |
| С# |
| // A constructor with two arguments.  public Coords(int x, int y)  {  this.x = x;  this.y = y;  } |

Это позволяет создавать объекты Coords с начальными значениями по умолчанию или с другими начальными значениями:

|  |
| --- |
| С# |
| var p1 = new Coords();  var p2 = new Coords(5, 3); |

Если класс не имеет конструктора, автоматически создается конструктор без параметров и для инициализации полей объекта используются значения по умолчанию. Например, int инициализируется значением 0.

Следовательно, поскольку конструктор без параметров класса Coords инициализирует все члены данных с нулевыми значениями, его можно удалить. При этом работа класса не изменится. Полный пример использования нескольких конструкторов см. в данном разделе в примере 1; пример автоматически созданного конструктора см. в примере 2.

Конструкторы экземпляров также можно использовать для вызова конструкторов экземпляров базового класса. Конструктор класса может вызвать конструктор базового класса с помощью инициализатора:

|  |
| --- |
| С# |
| class Circle : Shape  {  public Circle(double radius)  : base(radius, 0)  {  }  } |

В этом примере класс Circle передает значения радиуса и высоты конструктору, предоставленному классом Shape, для которого класс Circle является производным. Полный текст кода с использованием классов Shape и Circle см. в данном разделе в примере 3.

### Пример 1

В следующем примере демонстрируется класс с двумя конструкторами, один из которых не имеет аргументов, а второй имеет два аргумента.

|  |
| --- |
| С# |
| class Coords  {  public int x, y;  // Default constructor.  public Coords()  {  x = 0;  y = 0;  }  // A constructor with two arguments.  public Coords(int x, int y)  {  this.x = x;  this.y = y;  }  // Override the ToString method.  public override string ToString()  {  return $"({x},{y})";  }  }  class MainClass  {  static void Main()  {  var p1 = new Coords();  var p2 = new Coords(5, 3);  // Display the results using the overriden ToString method.  Console.WriteLine($"Coords #1 at {p1}");  Console.WriteLine($"Coords #2 at {p2}");  Console.ReadKey();  }  }  /\* Output:  Coords #1 at (0,0)  Coords #2 at (5,3)  \*/ |

### Пример 2

В этом примере класс Person не имеет конструкторов, поэтому автоматически предоставляется конструктор без параметров, а все поля инициализируются значениями по умолчанию.

|  |
| --- |
| С# |
| public class Person  {  public int age;  public string name;  }  class TestPerson  {  static void Main()  {  var person = new Person();  Console.WriteLine("Name: {person.name}, Age: {person.age}");  // Keep the console window open in debug mode.  Console.WriteLine("Press any key to exit.");  Console.ReadKey();  }  }  // Output: Name: , Age: 0 |

Обратите внимание, что значение по умолчанию age равно 0, а значение по умолчанию name равно null.

### Пример 3

В следующем примере демонстрируется использование инициализатора базового класса. Класс Circle является производным от основного класса Shape, а класс Cylinder является производным от класса Circle. Конструктор каждого производного класса использует инициализатор своего базового класса.

|  |
| --- |
| С# |
| abstract class Shape  {  public const double pi = Math.PI;  protected double x, y;  public Shape(double x, double y)  {  this.x = x;  this.y = y;  }  public abstract double Area();  }  class Circle : Shape  {  public Circle(double radius)  : base(radius, 0)  {  }  public override double Area()  {  return pi \* x \* x;  }  }  class Cylinder : Circle  {  public Cylinder(double radius, double height)  : base(radius)  {  y = height;  }  public override double Area()  {  return (2 \* base.Area()) + (2 \* pi \* x \* y);  }  }  class TestShapes  {  static void Main()  {  double radius = 2.5;  double height = 3.0;  Circle ring = new Circle(radius);  Cylinder tube = new Cylinder(radius, height);  Console.WriteLine("Area of the circle = {0:F2}", ring.Area());  Console.WriteLine("Area of the cylinder = {0:F2}", tube.Area());  // Keep the console window open in debug mode.  Console.WriteLine("Press any key to exit.");  Console.ReadKey();  }  }  /\* Output:  Area of the circle = 19.63  Area of the cylinder = 86.39  \*/ |

## Закрытые конструкторы

Закрытый конструктор — это особый конструктор экземпляров. Обычно он используется в классах, содержащих только статические элементы. Если в классе один или несколько закрытых конструкторов и ни одного открытого конструктора, то прочие классы (за исключением вложенных классов) не смогут создавать экземпляры этого класса. Пример:

|  |
| --- |
| С# |
| class NLog  {  // Private Constructor:  private NLog() { }  public static double e = Math.E; //2.71828...  } |

Объявление пустого конструктора запрещает автоматическое создание конструктора без параметров. Обратите внимание, что если не использовать с конструктором модификатор доступа, то по умолчанию он все равно будет закрытым. Однако обычно используется модификатор private, чтобы явно обозначить невозможность создания экземпляров этого класса.

Закрытые конструкторы используются, чтобы не допустить создания экземпляров класса при отсутствии полей или методов экземпляра, например для класса Math, или когда осуществляется вызов метода для получения экземпляра класса. Если все методы в классе являются статическими, имеет смысл сделать статическим весь класс.

### Пример

Ниже приведен пример класса с закрытым конструктором.

|  |
| --- |
| С# |
| public class Counter  {  private Counter() { }  public static int currentCount;  public static int IncrementCount()  {  return ++currentCount;  }  }  class TestCounter  {  static void Main()  {  // If you uncomment the following statement, it will generate  // an error because the constructor is inaccessible:  // Counter aCounter = new Counter(); // Error  Counter.currentCount = 100;  Counter.IncrementCount();  Console.WriteLine("New count: {0}", Counter.currentCount);  // Keep the console window open in debug mode.  Console.WriteLine("Press any key to exit.");  Console.ReadKey();  }  }  // Output: New count: 101 |

Обратите внимание, что если в примере раскомментировать следующий оператор, возникнет ошибка, так как конструктор недоступен из-за уровня защиты:

|  |
| --- |
| С# |
| // Counter aCounter = new Counter(); // Error |

## Статические конструкторы

Статический конструктор используется для инициализации любых статических данных или для выполнения определенного действия, которое требуется выполнить только один раз. Он вызывается автоматически перед созданием первого экземпляра или ссылкой на какие-либо статические члены.

|  |
| --- |
| С# |
| сlass SimpleClass  {  // Static variable that must be initialized at run time.  static readonly long baseline;  // Static constructor is called at most one time, before any  // instance constructor is invoked or member is accessed.  static SimpleClass()  {  baseline = DateTime.Now.Ticks;  }  } |

### Примечания

Статические конструкторы обладают следующими свойствами.

* Статический конструктор не принимает модификаторы доступа и не имеет параметров.
* Класс или структура могут иметь только один статический конструктор.
* Статические конструкторы не могут быть унаследованы или перегружены.
* Статический конструктор нельзя вызывать напрямую. Он предназначен только для вызова из общеязыковой среды выполнения (CLR). Он запускается автоматически.
* Пользователь не управляет временем, в течение которого статический конструктор выполняется в программе.
* Статический конструктор вызывается автоматически для инициализации класса перед созданием первого экземпляра типа или ссылкой на какие-либо статические члены. Статический конструктор выполняется раньше, чем конструктор экземпляра. Статический конструктор типа вызывается при вызове статического метода, назначенного делегату или событию, а не при его назначении. Если в классе со статическим конструктором существуют переменные инициализаторы статических полей, они поочередно выполняются в том порядке, в котором включены в объявление класса, и сразу после этого выполняется статический конструктор.
* Если вы не предоставили статический конструктор для инициализации статических полей, все статические поля инициализируются значениями по умолчанию, которые указаны в разделе Значения по умолчанию типов C#.
* Если статический конструктор инициирует исключение, среда выполнения не вызывает его во второй раз, и тип остается неинициализированным на время существования домена приложения, в котором выполняется программа. Чаще всего исключение TypeInitializationException вызывается, когда статический конструктор не может создать экземпляр типа или в статическом конструкторе создается необработанное исключение. Для устранения неполадок с неявными статическими конструкторами, которые не определены в исходном коде, может потребоваться проверка кода на промежуточном языке (IL).
* Наличие статического конструктора не позволяет добавлять атрибут типа BeforeFieldInit. Это ограничивает возможности оптимизации во время выполнения.
* Поле, объявленное как static readonly, может быть присвоено только при его объявлении или в статическом конструкторе. Если явный статический конструктор не является обязательным, инициализируйте статические поля в объявлении, а не через статический конструктор, чтобы сохранить больше возможностей для оптимизации в среде выполнения.

|  |
| --- |
| Примечание!  Наличие явного статического конструктора следует документировать, несмотря на отсутствие прямого доступа к нему, поскольку это важно для устранения неполадок с исключениями при инициализации. |

### Использование

* Типичным использованием статических конструкторов является случай, когда класс использует файл журнала и конструктор применяется для добавления записей в этот файл.
* Статические конструкторы также полезны при создании классов-оболочек для неуправляемого кода, когда конструктор может вызвать метод LoadLibrary.
* Статические конструкторы очень удобны для применения в среде выполнения проверок параметра типа, которые невозможно проверить с помощью ограничений во время компиляции (ограничения параметров типа).

### Пример

В этом примере класс Bus имеет статический конструктор. При создании первого экземпляра класса Bus (bus1) для инициализации класса вызывается статический конструктор. В выходных данных этого примера можно увидеть, что статический конструктор выполняется только один раз, несмотря на то, что создается два экземпляра класса Bus. Кроме того, этот конструктор вызывается до выполнения конструктора экземпляра.

|  |
| --- |
| С# |
| public class Bus  {  // Static variable used by all Bus instances.  // Represents the time the first bus of the day starts its route.  protected static readonly DateTime globalStartTime;  // Property for the number of each bus.  protected int RouteNumber { get; set; }  // Static constructor to initialize the static variable.  // It is invoked before the first instance constructor is run.  static Bus()  {  globalStartTime = DateTime.Now;  // The following statement produces the first line of output,  // and the line occurs only once.  Console.WriteLine("Static constructor sets global start time to {0}",  globalStartTime.ToLongTimeString());  }  // Instance constructor.  public Bus(int routeNum)  {  RouteNumber = routeNum;  Console.WriteLine("Bus #{0} is created.", RouteNumber);  }  // Instance method.  public void Drive()  {  TimeSpan elapsedTime = DateTime.Now - globalStartTime;  // For demonstration purposes we treat milliseconds as minutes to simulate  // actual bus times. Do not do this in your actual bus schedule program!  Console.WriteLine("{0} is starting its route {1:N2} minutes after global start time {2}.",  this.RouteNumber,  elapsedTime.Milliseconds,  globalStartTime.ToShortTimeString());  }  }  class TestBus  {  static void Main()  {  // The creation of this instance activates the static constructor.  Bus bus1 = new Bus(71);  // Create a second bus.  Bus bus2 = new Bus(72);  // Send bus1 on its way.  bus1.Drive();  // Wait for bus2 to warm up.  System.Threading.Thread.Sleep(25);  // Send bus2 on its way.  bus2.Drive();  // Keep the console window open in debug mode.  System.Console.WriteLine("Press any key to exit.");  System.Console.ReadKey();  }  }  /\* Sample output:  Static constructor sets global start time to 3:57:08 PM.  Bus #71 is created.  Bus #72 is created.  71 is starting its route 6.00 minutes after global start time 3:57 PM.  72 is starting its route 31.00 minutes after global start time 3:57 PM.  \*/ |

## Создание конструктора копий

В C# не предусмотрен конструктор копии для объектов, однако его можно написать самостоятельно.

В следующем примере класс Person определяет конструктор копии, который использует экземпляр Personв качестве аргумента. Значения свойств аргумента присваиваются свойствам нового экземпляра Person. Код содержит дополнительный конструктор копии, который отправляет свойства Name и Age экземпляра, который необходимо скопировать конструктору экземпляра класса.

|  |
| --- |
| С# |
| class Person  {  // Copy constructor.  public Person(Person previousPerson)  {  Name = previousPerson.Name;  Age = previousPerson.Age;  }  //// Alternate copy constructor calls the instance constructor.  //public Person(Person previousPerson)  // : this(previousPerson.Name, previousPerson.Age)  //{  //}  // Instance constructor.  public Person(string name, int age)  {  Name = name;  Age = age;  }  public int Age { get; set; }  public string Name { get; set; }    public string Details()  {  return Name + " is " + Age.ToString();  }  }  class TestPerson  {  static void Main()  {  // Create a Person object by using the instance constructor.  Person person1 = new Person("George", 40);  // Create another Person object, copying person1.  Person person2 = new Person(person1);  // Change each person's age.  person1.Age = 39;  person2.Age = 41;  // Change person2's name.  person2.Name = "Charles";  // Show details to verify that the name and age fields are distinct.  Console.WriteLine(person1.Details());  Console.WriteLine(person2.Details());  // Keep the console window open in debug mode.  Console.WriteLine("Press any key to exit.");  Console.ReadKey();  }  }  // Output:  // George is 39  // Charles is 41 |

# ЗАДАНИЯ ДЛЯ ВЫПОЛНЕНИЯ

1. Решите задание используя конструкторы
2. Спроектируйте блок схему к программе
3. Отобразить в отчете проделанные шаги
4. Ответить на контрольные вопросы
5. Написать вывод о проделанной работе

# ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

|  |
| --- |
| Вариант 1 – Напишите класс Ball, который должен иметь следующие две закрытые переменные-члены со значениями по умолчанию:  m\_color (Red);  m\_radius (20.0).  В Ball должны быть следующие конструкторы:  для установления значения только для m\_color;  для установления значения только для m\_radius;  для установления значений и для m\_radius, и для m\_color;  для установления значений, когда значения не предоставлены вообще.  Не используйте параметры по умолчанию для конструкторов. Напишите ещё одну функцию для вывода цвета (m\_color) и радиуса (m\_radius) шара (объекта класса Ball). |
| Вариант 2 – Объявить класс, содержащий одномерный целочисленный массив размерностью 10, индексатор для работы с этим массивом, а также конструктор по умолчанию и конструктор с одним аргументом. Конструктор по умолчанию присваивает элементам массива значение равное индексу элемента. Конструктор с аргументами присваивает элементам массива значение равное сумме аргумента с индексом элемента. Создать объекты этого класса с использованием всех конструкторов. Аргументы для конструктора вводятся с клавиатуры. Распечатать поля (массивы) объектов. |
| Вариант 3 – Необходимо создать класс с именем time, содержащий три поля типа int, предназначенные для хранения часов, минут и секунд. Один из конструкторов должен инициализировать поля нулевыми значениями, а другой - заданным набором значений. Создать метод класса, который будет выводить значения полей на экран в формате 11:40:40. |
| Вариант 4 - Объявите класс Man, создающий тип – человека. Элементы класса – фамилия, имя, отчество, возраст, адрес человека. Предусмотрите конструкторы класса: - по умолчанию; - получающий параметры; - получающий параметр –ссылку на класс Man. Напишите функции – методы класса для: - ввода – вывода данных о человеке; - определения, является ли человек избирателем, если учитывать только его возраст; - определения совпадения фамилии человека с некоторой заданной. Напишите программу, в которой создайте массив объектов типа Man, заполните его в диалоговом режиме, выведите сначала исходный массив, а затем всех избирателей с заданной фамилией. Примечание: Массив объектов класса может быть создан как статически, так и динамически. Объекты класса, которые будет содержать массив, обязательно должны иметь конструкторы по умолчанию. При создании массива объектов класса для каждого объекта массива будет вызван конструктор по умолчанию. |
| Вариант 5 – Представьте, что разрабатываете игру. Создайте конструктор Unit(), создающий объекты с координатами x, y. При помощи прототипного наследования расширьте Unit(), создав новый конструктор Fighter() с свойством power. |
| Вариант 6 - Напишите класс, конструктор которого принимает два массива: массив значений и массив весов значений. Класс должен содержать метод, который будет возвращать элемент из первого массива случайным образом, с учётом его веса. Пример: Дан массив [1, 2, 3], и массив весов [1, 2, 10]. В среднем, значение **«1»** должно возвращаться в 2 раза реже, чем значение **«2»** и в десять раз реже, чем значение **«3»**. |
| Вариант 7 - Санаторий «Бешеные псы» им. Тарантино принимает собак всех пород и мастей на двухнедельный отдых. Регистрация открыта до 02.11.2021 года. Заполните одну из трех анкет: указать имя, рост и цвет питомца. Создайте класс Dog конструктор для регистрации. |
| Вариант 8 - Если нам нужна окружность для вычисления его длины, нам нужен её радиус, если для аналитической геометрии — радиус и центр, ну а если мы хотим вывести её на экран, то желательно ещё и её цвет прописать. Клоним к тому, что нам нужно создать класс Circle и собрать для него три разных конструктора. |
| Вариант 9 - Давайте создадим класс «прямоугольник» с полями «верхняя координата», «левая координата», высота, ширина. А затем определим для него как можно больше конструкторов. |
| Вариант 10 - Люди — разные, и у каждого человека различные важные особенности. Скажем, этот — новорожденный, и у него есть только имя и мама, а этот — музыкант, и для него важны уровень исполнительского мастерства и его группа. Давайте напишем класс Human с 6 полями и придумаем для него 10 различных конструкторов. Каждый конструктор должен иметь смысл. |

# КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как устроена работа Статических конструкторов?
2. Как устроена работа Закрытые конструкторы?
3. Что такое конструкторы?

# РЕКОМЕНДОВАННАЯ ЛИТЕРАТУРА

1. Официальная документация Microsoft C# — Текст: электронный // Microsoft [сайт]. — URL: https://docs.microsoft.com/ (дата обращения: 12.03.2020).
2. Сообщество IT-специалистов — Текст: электронный // Habr [сайт]. — URL: https://habr.com/ (дата обращения: 12.03.2020).
3. Сайт о программирование / — Текст: электронный // Metanit [сайт]. — https://metanit.com/ (дата обращения: 12.03.2020).
4. С# Tutotial — Текст: электронный // Tutotial [сайт]. — URL: https://csharp.net-tutorials.com/ (дата обращения: 13.03.2020).

# ПРИЛОЖЕНИЕ А

**Требования к отчету**

Общие требования:

1. Шрифт – Times New Roman, 14 пт.
2. Интервалы: междустрочный – 1,5 строки, интервал до и после абзаца – 0 пт.
3. Отступ первой строки – 1,25
4. Рисунки и подписи к ним выравниваются по центру.

Требования к структуре отчета:

1. Титульный лист
2. Содержание
3. Основная часть:
   1. Задачи
   2. Код
   3. Блок-схемы
   4. Результат работы
4. Выводы по работе
5. Ответы на контрольные вопросы

# ПРИЛОЖЕНИЕ Б

ОГБПОУ «ТОМСКИЙ ТЕХНИКУМ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ»

Отчет по учебной-практической работе №4

учебной практике   
*УП 05. Программирование*

*Тема: «Конструкторы»*

Выполнил:

студент \_\_\_\_\_\_\_ группы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил:  
преподаватель

Сидиков И.Д.

г. Томск – 2020 г