МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образование «Полоцкий государственный университет имени Евфросинии Полоцкой»

Факультет информационных технологий

Кафедра технологий программирования

**Лабораторная работа №2**

по дисциплине: «Программирование на платформе .NET»

на тему: «Типы данных, определяемые пользователем. Наследование. Обработка исключений в C#»

Вариант №10

|  |  |
| --- | --- |
| Выполнил | Студент группы 21-ИТ-1  Чиникайло А.П. |
|  |  |
| Проверил | Преподаватель  Макарычева В.А. |

Полоцк, 2024

**ЦЕЛЬ РАБОТЫ**

Познакомиться с пользовательскими типами данных в языке С#: структура и перечисление. Ознакомиться со структурой класса, его созданием и использованием, описанием членов класса: полей, свойств, инициализации объектов класса с помощью конструкторов. Изучить механизм создания иерархий классов в C# и применение интерфейсов при наследовании. Изучить механизм генерации и обработки исключений.

**ЗАДАНИЕ**

Выполнить задания на двумерный массив:

1. Для заданной структуры данных разработать абстрактный класс и класс-наследник. В классе реализовать несколько конструкторов.
2. Создать методы, работающие с полями класса. Часть из них должны быть виртуальными. Добавить методы-свойства и индексаторы.
3. Разработать интерфейсные классы, добавляющие некоторые методы в класс-потомок. Изучить причины возникновения коллизий имен при наследовании и способы их устранения.
4. Разработать классы исключительных ситуаций и применить их для обработки, возникающих исключений.

**ХОД РАБОТЫ**

Листинг 1 – Ship.cs

public abstract class Ship

{

protected string name;

protected float displacement;

protected ShipType shipType;

protected List<CabinCategory> cabinCategories;

//метод для вывода информации на экран

public virtual void ShowInfo()

{

Console.WriteLine($"name: {name}");

Console.WriteLine($"displacement: {displacement}");

Console.WriteLine($"shipType: {shipType}");

Console.WriteLine($"cabinCategories:");

foreach( var category in cabinCategories )

{

Console.WriteLine(category);

}

}

//индексатор

public CabinCategory this[int i]

{

get

{

if (i >= 0 && i < cabinCategories.Count) return (cabinCategories[i]);

else return (cabinCategories[0]);

}

set

{

if (i >= 0 && i < cabinCategories.Count)//проверка, что бы значение не выходило за предел массива

{

cabinCategories[i] = value;

}

else

{

cabinCategories.Add(value);

}

}

}

// свойство поля name

public string Name

{

get => this.name;

set

{

this.name = value;

}

}

// свойство поля Name

public float displacement

{

get => this.displacement;

set

{

this.displacement = value;

}

}

// сеттер поля shipType

public void setShipType(ShipType shipType)

{

this.shipType = shipType;

}

public ShipType getShipType()

{

return this.shipType;

}

// сеттер списка CabinCategory

public void setCabinCategories(List<CabinCategory> cabinCategories)

{

this.cabinCategories = cabinCategories;

}

public List<CabinCategory> getCabinCategories()

{

return this.cabinCategories;

}

public void AddCabinCategory(CabinCategory cabinCategory)

{

this.cabinCategories.Add(cabinCategory);

}

//переопределённый метод

public override string ToString()

{

string categories = string.Empty;

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

{

categories += cabinCategories[i].ToString();

if (i != cabinCategories.Count - 1)

categories += ", ";

}

return "Name: " + name +

"; Displacement: " + displacement +

"; ShipType: " + shipType.ToString() +

"; CabinCategories: " + categories;

}

}

Абстрактный класс ship, который содержит 4 поля: name, displacement, shipType, cabinCategories. Виртуальный метод ShowInfo. Индексатор, который возвращает enum CabinCategory.

Есть 2 свойства для полей name и displacement. А так же геттеры и сеттеры для полей shipType, cabinCategories. В данном классе переопределён метод ToString().

Листинг 2 – PassengerShip.cs

//класс наследник

public class PassengerShip : Ship, ITurnLeft, ITurnRight

{

//конструктор без параметров

public PassengerShip()

{

}

public PassengerShip(string name, ShipType shipType)

{

this.name = name;

this.shipType = shipType;

}

//конструктор без параметров

public PassengerShip(string name, float displacement, ShipType shipType, List<CabinCategory> cabinCategories)

{

this.name = name;

this.displacement = displacement;

this.shipType = shipType;

this.cabinCategories = new List<CabinCategory>(cabinCategories);

}

//реализация первого интерфейса

void ITurnRight.Turn(int degree)

{

Console.WriteLine($"Turn Right {degree} degrees");

}

//реализация второго интерфейса

void ITurnLeft.Turn(int degree)

{

Console.WriteLine($"Turn Left at corner {degree} degrees");

}

//создание методов для вызова первого реализованного интерфейса

public void TurnLeft(int degree)

{

((ITurnLeft)this).Turn(degree);

}

//создание методов для вызова первого реализованного интерфейса

public void TurnRight(int degree)

{

((ITurnRight)this).Turn(degree);

try

{

if (degree == 0)

throw new MyException("why do you need to turn to 0 degrees");

}

catch (Exception ex)

{

Console.WriteLine(ex);

}

finally

{

Console.WriteLine("but in principle it is not a mistake");

}

}

}

Данный класс является классом-наследником класса Ship. Существует 3 конструктора: 1. Конструктор без параметров, 2. конструктор с 2 параметрами: имя корабля, и тип корабля.

3. конструктор со всеми параметрами, а именно имя, тип корабля, водоизмещение, типы комнат.

Реализовано 2 интерфейса с одинаковыми методами. Данные методы сделаны private. Так же созданы 2 метода, для вызова методов, которые реализовали от интерфейсов.

Листинг 3 – ITurnLeft.cs

public interface ITurnLeft

{

void Turn(int degree);

}

Листинг 4 – ITurnRight.cs

public interface ITurnRight

{

public void Turn(int degree);

}

Листинг 5 – MyException.cs

// собственный класс Exception

public class MyException : Exception

{

public MyException(string? message) : base(message)

{

}

}

Кастомный класс Exception, который наследуется от класса Exception.

Листинг 6 – CabinCategory.cs

public enum CabinCategory

{

InsideCabin,

OutsideCabinWithWindow,

OutsideCabinWithBalcony,

Luxury

}

Перечисление категорий комнат

Листинг 7 – ShipType.cs

public enum ShipType

{

CargoShip,

PassengerShip,

IndustrialShip,

}

Перечисление категорий кораблей

Листинг 8 – Program.cs

class Program

{

private static List<Ship> ships = new List<Ship>();

static void Main(string[] args)

{

while (true)

{

PrintMenu();// вывод меню

int choose;

beforeStart:

// проверка значения

if (int.TryParse(Console.ReadLine(), out int i) && i <= 2)

{

choose = i;

}

else

{

Console.WriteLine("you entered incorrect value or less than.\n try again");

goto beforeStart;// возвращаемся к повторному вводу значения

}

switch (choose)

{

case 0:

Ship ship = createShip(); // создание корабля

ships.Add(ship);

break;

case 1:

PrintShips();// вывод всех кораблей

break;

case 2:

// проверка значения

beforePrintShip:

if (int.TryParse(Console.ReadLine(), out i) && i >= 0)

{

if (ships.Count == 0)

{

Console.WriteLine("list is empty");

break;

}

if (i < ships.Count)

{

PrintShip(i);

}

}

else

{

Console.WriteLine("you entered incorrect value or less than.\n try again");

goto beforePrintShip; // возвращаемся к повторному вводу значения

}

break;

case -1:

return;

default:

break;

}

Console.ReadKey();

Console.Clear();

}

}

// функция вывода корабля по индексу

private static void PrintShip(int index)

{

Console.WriteLine(ships[index]);

}

private static void PrintMenu()

{

Console.WriteLine("0 - add ship");

Console.WriteLine("1 - print ships");

Console.WriteLine("2 - print ship by index");

Console.WriteLine("-1 - exit");

}

// функция вывода всех кораблей

private static void PrintShips()

{

if( ships.Count == 0)

{

Console.WriteLine("List is empty");

return;

}

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

{

Console.WriteLine(i + ": "+ ships[i]);

}

//foreach (Ship ship in ships)

//{

// Console.WriteLine(ship);

//}

}

// функция создание корабля

private static Ship createShip()

{

Console.WriteLine("Enter name: ");

string name = Console.ReadLine();

beforeDisplacement:

Console.WriteLine("Enter displacement: ");

float displacement = 0;

if (float.TryParse(Console.ReadLine(), out float f))

{

displacement = f;

}

else

{

Console.WriteLine("you entered incorrect value.\ntry again");

goto beforeDisplacement; // возвращаемся к повторному вводу значения

}

// проверка значения

beforeShipType:

Console.WriteLine("choose Ship Type:");

Console.WriteLine("0 - CargoShip");

Console.WriteLine("1 - PassengerShip");

Console.WriteLine("2 - IndustrialShip");

//int chooseShipType = int.Parse(Console.ReadLine());

ShipType shipType = 0;

if (int.TryParse(Console.ReadLine(), out int i) && i <= 2)

{

shipType = (ShipType)i;

}

else

{

Console.WriteLine("you entered incorrect value or less than.\n try again");

goto beforeShipType; // возвращаемся к повторному вводу значения

}

List<CabinCategory> cabinCategories = new List<CabinCategory>();

Console.WriteLine("Enter cabin categories");

Console.WriteLine("0 - InsideCabin");

Console.WriteLine("1 - OutsideCabinWithWindow");

Console.WriteLine("2 - OutsideCabinWithBalcony");

Console.WriteLine("3 - Luxury");

Console.WriteLine("-1 - cancel");

while (true)

{

Console.WriteLine("Enter number: ");

// проверка значения

if (int.TryParse(Console.ReadLine(), out i) && i <= 3)

{

if (i == -1)

break;

cabinCategories.Add((CabinCategory)i);

}

else

{

Console.WriteLine("you entered incorrect value or less than.\n try again");

goto beforeShipType; // возвращаемся к повторному вводу значения

}

}

Ship ship = new PassengerShip(name, displacement, shipType, cabinCategories);// создание экземпляра класса корабля

return ship;

}}

Функция main, с которой начинается ход программы, содержит switch, для разделения на 3 сектора, первый: добавления корабля в список, так же создан метод для создании экземпляра корабля. Второй для вывода информации о корабле по индексу. Используется метод, описанный ниже для вывода информации о корабле по индексу. Третий вывод информации о всех кораблях. Добавлен метод для вывода меню.

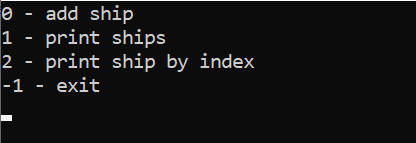


Рисунок 1 – выбор действия

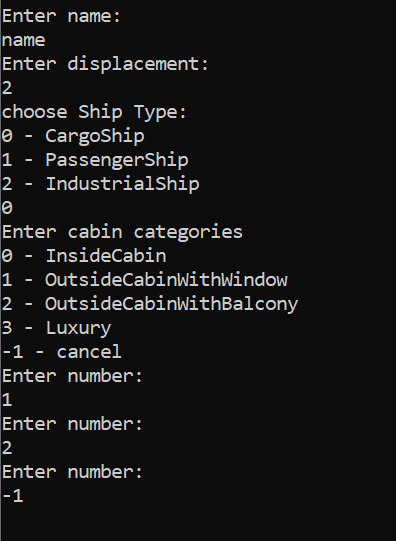


Рисунок 2 – добавление корабля

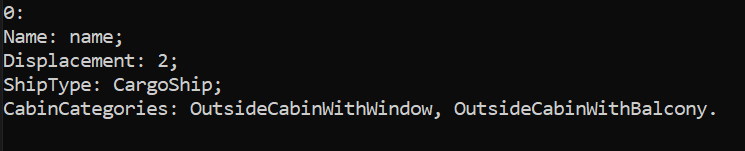


Рисунок 3 – вывод информации о корабле

**ОТВЕТЫ НА ВОПРОСЫ**

1. К какой группе типов переменных относятся перечисления и структуры?

Перечисления и структуры относятся к пользовательским типам данных в C#. Они являются значимыми типами, которые могут содержать данные и методы, но в отличие от классов, структуры являются значимыми типами, а перечисления представляют собой набор именованных констант.

2. Описание перечисления и его назначение?

Перечисление (enum) — это специальный тип данных, который позволяет задавать набор именованных целочисленных констант. Оно используется для улучшения читаемости кода и упрощения работы с фиксированными наборами значений, такими как дни недели, месяцы и т.д. Синтаксис объявления перечисления включает имя перечисления и список возможных значений.

3. Числовые значения констант перечисления по умолчанию?

Константы перечисления по умолчанию имеют целочисленные значения, начиная с 0 для первой константы и увеличиваясь на 1 для каждой последующей. Однако можно явно задать значения для констант.

4. Использование перечислений в программах.

Перечисления используются для создания переменных, которые могут принимать только заранее определенные значения. Это позволяет избежать ошибок, связанных с использованием некорректных значений. Например, можно использовать перечисления для определения статусов, режимов работы и т.д.

5. Описание структуры. Конструкторы.

Структура — это пользовательский тип данных, который может содержать поля, методы и конструкторы. Конструкторы в структурах могут быть определены для инициализации полей при создании экземпляра структуры. В отличие от классов, структуры не могут иметь неявных конструкторов без параметров.

6. Обращение к элементам структуры.

Элементы структуры (поля и методы) могут быть доступны через экземпляр структуры. Например, если структура называется MyStruct, а экземпляр — myStruct, то доступ к полям осуществляется через myStruct.FieldName.

7. Что такое класс? Для чего создаются классы?

Класс — это шаблон для создания объектов, который определяет поля, методы и свойства. Классы используются для организации и структурирования кода, инкапсуляции данных и поведения, а также для реализации наследования и полиморфизма.

8. Чем отличается класс от структуры?

Основные отличия между классами и структурами заключаются в следующем:

1. Классы являются ссылочными типами, а структуры — значимыми.
2. Классы могут наследоваться, структуры — нет.
3. Классы могут иметь неявные конструкторы без параметров, структуры — нет.

9. Модификаторы доступа к полям и методам класса.

Модификаторы доступа определяют уровень видимости полей и методов класса. Основные модификаторы:

1. public: доступен из любого места.
2. private: доступен только внутри класса.
3. protected: доступен в классе и его производных.
4. internal: доступен в пределах одной сборки.
5. protected internal: доступен в пределах одной сборки и в производных классах.

10. Модификаторы доступа к классам.

Модификаторы доступа к классам определяют, где класс может быть доступен. Основные модификаторы:

1. public: класс доступен из любого места.
2. private: класс доступен только внутри своего определения (может использоваться только в вложенных классах).
3. protected: класс доступен в своем классе и в производных классах.
4. internal: класс доступен только в пределах одной сборки.
5. protected internal: класс доступен в пределах одной сборки и в производных классах.

11. Что такое экземпляр класса? Как он создается в C#?

Экземпляр класса — это объект, созданный на основе определения класса. Он создается с помощью оператора new, который выделяет память для объекта и вызывает конструктор класса. Например:

MyClass myObject = new MyClass();

Здесь myObject является экземпляром класса MyClass.

12. Для чего в классе определяется конструктор? Сколько может быть конструкторов в классе? Когда вызывается конструктор?

Конструктор используется для инициализации новых объектов класса. В классе может быть несколько конструкторов (перегрузка конструкторов), каждый из которых может иметь разные параметры. Конструктор вызывается автоматически при создании экземпляра класса.

13. Как можно обратиться к полям и методам класса?

Для обращения к полям и методам класса используется экземпляр класса. Например, если у вас есть класс MyClass с полем myField и методом MyMethod, вы можете обратиться к ним так:

MyClass myObject = new MyClass();

myObject.myField = 10; // Доступ к полю

myObject.MyMethod(); // Вызов метода

14. Методы-свойства класса. Назначение и описание.

Методы-свойства (или свойства) позволяют управлять доступом к полям класса. Они обеспечивают механизм для чтения и записи значений полей, при этом могут включать логику валидации. Пример свойства:

public int MyProperty {

get { return myField; }

set { myField = value; }

}

15. Статические поля и методы класса. Назначение, описание и вызов статических методов.

Статические поля и методы принадлежат самому классу, а не его экземплярам. Они могут быть вызваны без создания экземпляра класса. Например:

public class MyClass {

public static int MyStaticField;

public static void MyStaticMethod() { }

}

MyClass.MyStaticField = 5; // Доступ к статическому полю

MyClass.MyStaticMethod(); // Вызов статического метода

16. Индексаторы. Назначение и описание.

Индексаторы позволяют объектам класса вести себя как массивы, предоставляя доступ к элементам по индексу. Они определяются с помощью ключевого слова this. Пример:

public class MyCollection {

private int[] array = new int[10];

public int this[int index] {

get { return array[index]; }

set { array[index] = value; }

}

}

17. Какое наследование применяется в С#? Что наследует потомок от класса-родителя?

В C# поддерживается однонаследование, то есть класс может наследовать только от одного родительского класса. Потомок наследует все публичные и защищенные члены родительского класса, включая поля, методы и свойства.

18. Изменение методов родителя в классе наследника.

Методы родителя могут быть переопределены в классе-наследнике с помощью ключевого слова override, если они были объявлены с модификатором virtual в родительском классе. Это позволяет изменять поведение унаследованных методов.

19. Конструкторы при наследовании.

При наследовании конструкторы родительского класса могут быть вызваны из конструктора класса-наследника с помощью ключевого слова base. Это позволяет инициализировать поля родительского класса. Например:

public class Parent {

public Parent(int value) { }

}

public class Child : Parent {

public Child(int value) : base(value) { }

}

20. Описание абстрактных методов и классов.

Абстрактные классы в C# служат в качестве базовых классов, которые не могут быть инстанцированы напрямую. Они могут содержать как абстрактные методы (без реализации), так и обычные методы. Абстрактные методы должны быть переопределены в производных классах. Это позволяет создавать общий интерфейс для группы связанных классов, обеспечивая при этом возможность реализации специфического поведения в каждом из них.

21. Вложенные классы.

Вложенные классы (или внутренние классы) — это классы, определенные внутри другого класса. Они могут быть полезны для организации кода и логического группирования классов, которые имеют смысл только в контексте родительского класса. Вложенные классы могут обращаться к членам родительского класса, включая его закрытые члены.

22. Интерфейсы. Назначение и описание.

Интерфейсы в C# определяют контракт, который классы могут реализовать. Они содержат объявления методов, свойств, событий и индексаторов, но не содержат их реализации. Интерфейсы позволяют создавать гибкие и расширяемые архитектуры, так как разные классы могут реализовать один и тот же интерфейс по-разному.

23. Наследование в интерфейсах.

Интерфейсы могут наследовать друг от друга, что позволяет создавать более сложные интерфейсы на основе более простых. Это позволяет избежать дублирования кода и обеспечивает возможность создания иерархий интерфейсов.

24. Реализация методов интерфейсов в классах.

Классы, которые реализуют интерфейсы, должны предоставлять реализацию всех методов, объявленных в интерфейсе. Это обеспечивает, что класс будет соответствовать контракту, определенному интерфейсом, и позволяет использовать объекты класса через интерфейс.

25. Коллизия имен в интерфейсах.

Если класс реализует несколько интерфейсов, которые содержат методы с одинаковыми именами, это может привести к конфликтам. В таких случаях класс должен явно указать, какой метод из какого интерфейса он реализует, используя явную реализацию интерфейса.

26. Исключения. Обработка исключений в С#.

Исключения в C# используются для обработки ошибок и других исключительных ситуаций, которые могут возникнуть во время выполнения программы. Обработка исключений осуществляется с помощью блоков try, catch и finally. Код, который может вызвать исключение, помещается в блок try, а обработка исключений — в блок catch.

27. Создание классов исключений и генерация исключения.

В C# можно создавать собственные классы исключений, наследуя от класса Exception. Это позволяет создавать специфические для приложения исключения, которые могут содержать дополнительную информацию. Исключения могут быть сгенерированы с помощью ключевого слова throw, что позволяет сигнализировать о возникновении ошибки в программе.

**ВЫВОД**

В результате выполнения лабораторной работы я познакомился с пользовательскими типами данных в языке С#, со структурой класса, его созданием и использованием, описанием членов класса. Изучил механизм создания иерархий классов в С# и применение интерфейсов при наследовании, а также механизм генерации и обработки исключений.