# ЛАБА6

1. Исключение генерируется в программе, когда возникает некоторая ошибка или необычная ситуация, которая нарушает нормальный поток выполнения программы. Генерация исключения происходит при помощи оператора throw, который указывает на тип исключения, которое нужно создать и выбросить.

2. Методика обработки исключений включает в себя использование блоков try-catch-finally. Код, который может сгенерировать исключение, помещается в блок try. Затем следует один или несколько блоков catch, каждый из которых обрабатывает определенный тип исключения. Если исключение выбрасывается в блоке try, то управление передается в соответствующий блок catch, который обрабатывает исключение. Блок finally используется для выполнения кода, который должен быть выполнен независимо от того, было ли выброшено исключение или нет.

3. Ключевое слово для обозначения блока кода, в котором можно генерировать исключение, - throw.

4. В языке C# используются ключевые слова try, catch, finally и throw для обработки и генерации исключений. Механизм обработки исключений состоит в следующем: код, который может вызвать исключение, помещается в блок try. Если исключение выбрасывается в блоке try, то управление передается в соответствующий блок catch, который обрабатывает исключение. Блок finally используется для выполнения кода, который должен быть выполнен независимо от того, было ли выброшено исключение или нет. Оператор throw используется для генерации и выброса исключения.

5. Если в программе нет предложения catch, способного обработать исключение, то исключение будет передано на верхний уровень вызова (на уровень, где был вызван метод, содержащий блок try-catch). Если исключение не обработано на этом уровне вызова, оно может привести к завершению программы и выводу сообщения об ошибке.

6. Фильтры исключений представляют возможность добавить дополнительные условия для обработки исключений в блоке catch. Фильтр исключения указывается с помощью ключевого слова when, за которым следует условие. Если условие в фильтре истинно, то блок catch выполняется, иначе исключение передается на следующий блок catch или на верхний уровень вызова. Пример:

try

{

// Код, который может вызвать исключение

}

catch (Exception ex) when (ex.Message.Contains("important"))

{

// Обработка исключения, если сообщение содержит "important"

}

catch (Exception ex)

{

// Обработка остальных исключений

}

7. Да, исключения могут быть вложенными. В блоке catch одного исключения может быть выброшено другое исключение, которое будет обработано во внешнем блоке catch или передано на верхний уровень вызова.

8. Для отлова любого возможного исключения в C# используется блок catch с параметром типа Exception. Пример:

try

{

// Код, который может вызвать исключение

}

catch (Exception ex)

{

// Обработка исключения

}

9. При размещении обработчиков исключений следует руководствоваться принципом "ловить исключения там, где вы можете предпринять действия для их обработки или восстановления. Обработчики исключений следует размещать на тех участках кода, где они имеют смысл и могут быть эффективно обработаны. Например, если вы знаете, что определенная операция может вызвать исключение, вы можете разместить соответствующий блок try-catch вокруг этой операции, чтобы обработать исключение и принять необходимые меры.

10. В результате выполнения данного фрагмента кода будет выведено на консоль сообщение "IndexOutOfRangeException". При попытке присвоить значение "anything" элементу массива str[4] произойдет выход за границы массива, и будет сгенерировано исключение типа IndexOutOfRangeException. Блок catch с параметром IndexOutOfRangeException будет соответствовать этому исключению и выведет соответствующее сообщение на консоль.

11. Для повторной генерации того же самого исключения в блоке catch можно использовать оператор throw без аргументов. Например:

try

{

// Код, который может вызвать исключение

}

catch (Exception ex)

{

// Обработка исключения

throw; // Повторная генерация исключения

}

12. В классе Exception содержится ряд методов, которые можно использовать для работы с исключениями. Некоторые из них:

- GetMessage(): Возвращает сообщение, связанное с исключением.

- GetType(): Возвращает тип исключения.

- ToString(): Возвращает строковое представление исключения.

- GetStackTrace(): Возвращает строку, содержащую стек вызовов, связанных с исключением.

Эти методы могут быть использованы для получения информации об исключении и его обработке, например, для вывода сообщений об ошибке или логирования. Методы вызываются на экземпляре класса Exception, который представляет собой конкретное исключение.

# ЛАБА7

1. Обобщение (generic) в программировании - это механизм, который позволяет создавать обобщенные типы данных и методы, которые могут работать с различными типами данных. Обобщение позволяет параметризовать типы данных, чтобы они могли быть использованы безопасно и эффективно с различными типами, не требуя дублирования кода.

2. 5

3. Ограничения на параметр обобщения можно наложить с помощью ключевого слова where. Например, можно наложить ограничение, чтобы параметр был классом, интерфейсом, структурой или имел конструктор по умолчанию.

4. Для наложения нескольких ограничений на параметр обобщения можно использовать запятую для разделения ограничений. Например: where T : class, IDisposable.

5. Существуют следующие ограничения на типы данных обобщений:

- where T : class - параметр должен быть ссылочным типом (классом или интерфейсом).

- where T : struct - параметр должен быть значимым типом (структурой).

- where T : new() - параметр должен иметь конструктор без параметров.

- where T : базовый\_класс - параметр должен быть производным от указанного базового класса.

- where T : интерфейс - параметр должен реализовывать указанный интерфейс.

- where T : U - параметр должен быть или наследником от класса U, или реализовывать интерфейс U.

6. Ограничение на тип задано в следующем фрагменте листинга: `class Test<T> where T : A { }`. Здесь говорится, что параметр T должен быть или классом A, или производным от класса A.

7. Ограничение на тип задано в следующем фрагменте листинга: `class Test<T> where T : class { }`. Здесь говорится, что параметр T должен быть ссылочным типом (классом).

8. Ограничение на тип задано в следующем фрагменте листинга: `class Test<T> where T : struct { }`. Здесь говорится, что параметр T должен быть значимым типом (структурой).

9. Примеры, когда обобщенный класс может действовать как базовый или производный класс, могут быть следующими:

- Обобщенный класс может быть использован в качестве базового класса для производных классов, которые могут специализировать его параметры типов или добавить дополнительные функции.

|  |
| --- |
| public abstract class Shape<T>  {  public abstract T CalculateArea();  }  public class Circle : Shape<double>  {  public double Radius { get; set; }  public override double CalculateArea()  {  return Math.PI \* Radius \* Radius;  }  }  public class Square : Shape<int>  {  public int SideLength { get; set; }  public override int CalculateArea()  {  return SideLength \* SideLength;  }  } |

- Обобщенный класс может быть производным от другого обобщенного класса с более специализированными параметрами типов.

|  |
| --- |
| class Person<T>  {      public T Id { get;}      public Person(T id)      {          Id = id;      }  }  class IntPerson<T> : Person<int>  {      public T Code { get; set; }      public IntPerson(int id, T code) : base(id)      {          Code = code;      }  } |

10. Оператор default в обобщениях используется для получения значения по умолчанию для параметра типа T. Он возвращает значение, которое соответствует типу T: null для ссылочных типов, 0 для числовых типов и т.д. Оператор default может быть полезен, когда требуется инициализировать переменную типа T с некоторым значением по умолчанию внутри обобщенного класса или метода.

11. Статические переменные в обобщенных классах общие для всех экземпляров класса с одним и тем же параметром типом. Они относятся к самому классу, а не к конкретным экземплярам. Их можно использовать для хранения общих данных или состояний между различными экземплярами обобщенного класса. Для объявления статической переменной в обобщенном классе используется ключевое слово static перед типом переменной.

12. Пример обобщенного интерфейса:

interface IRepository<T>

{

void Add(T item);

void Remove(T item);

T Get(int id);

}

В данном примере `IRepository<T>` - это обобщенный интерфейс, который определяет методы для добавления, удаления и получения элемента типа T.

13. Основное отличие между обобщенными классами и обобщенными структурами в языке программирования C# заключается в том, что классы являются ссылочными типами, а структуры - значимыми типами. Обобщенные классы могут наследоваться от других классов, реализовывать интерфейсы и иметь дополнительные возможности, такие как наследование и полиморфизм. В то же время обобщенные структуры ограничены в своих возможностях и не могут наследоваться от других структур или классов, а также не могут быть использованы в качестве базовых классов.

14. В языке программирования C# для работы с файлами можно использовать классы из пространства имен `System.IO`. Некоторые из наиболее часто используемых классов для работы с файлами включают:

- `File` - предоставляет статические методы для создания, копирования, удаления, перемещения и других операций с файлами.

- `Directory` - предоставляет статические методы для создания, перемещения и управления каталогами.

- `Path` - предоставляет статические методы для работы с путями к файлам, включая операции получения имени файла, имени каталога, расширения файла и т.д.

- `StreamReader` и `StreamWriter` - классы для чтения и записи текстовых данных из/в файлы.

- `BinaryReader` и `BinaryWriter` - классы для чтения и записи бинарных данных из/в файлы.

Пример использования класса `File` для чтения содержимого файла:

string path = "example.txt";

string content = File.ReadAllText(path);

Console.WriteLine(content);

В этом примере содержимое файла с именем "example.txt" читается и выводится на консоль.