Лекция 3: Выражения и исключения

Обзор

- Введение в управляющие операторы
- Операторы выбора
- Итерационные операторы
- Операторы перехода
- Обработка исключений
- Генерирование исключений

Обзор управляющих операторов

- Объединение операторов в блоки
- Типы управляющих операторов

Объединение операторов в блоки

- В качестве ограничителей используйте фигурные скобки

```
{
   // code
}
```

- Родительский и дочерний блоки не могут иметь переменные с одинаковыми именами

```
{
   int i;
   ...
   {
      int i;
      ...
   }
}
```

- Блоки одного уровня могут иметь переменные с одинаковыми именами

```
{
   int i;
   ...
}
...
{
   int i;
   ...
}
```

Типы управляющих операторов

Операторы выбора
if, switch

Итерационные операторы
while, do, for и foreach

Операторы перехода
goto, break и continue

Операторы выбора

- Оператор if
- Конструкция if-else-if и вложенные if-инструкции
- Оператор switch

Условный оператор if

- **Синтаксис:**

```
if ( условие )
  инструкция
else
  инструкция
```

- **Нет автоматического преобразования типа int в bool:**

```
int x;
...
if (x) ...      // Must be if (x != 0) in C#
if (x = 0) ... // Must be if (x == 0) in C#
```

Конструкция if-else-if и вложенные if-инструкции

```
String color;
enum Suit { Clubs, Hearts, Diamonds, Spades }
Suit trumps = Suit.Hearts;
if (trumps == Suit.Clubs)
    color = "Black";
else if (trumps == Suit.Hearts)
    color = "Red";
else if (trumps == Suit.Diamonds)
    color = "Red";
else
    color = "Black";
```

Оператор switch

- Для обеспечения многонаправленного ветвления используйте инструкцию switch
- Для запрета «провальной» передачи управления вниз используйте инструкцию break

```
switch (trumps) {
case Suit.Clubs :
      color = "Black"; break;
case Suit.Spades :
    color = "Black"; break;
case Suit.Hearts :
case Suit.Diamonds :
    color = "Red"; break;
default:
    color = "ERROR"; break;
}
```

Контрольный вопрос

Скомпилируется ли данный фрагмент кода успешно?

```
private int GetID (string inputText)
{
    if (inputText != "")
        return 1;
    else if (inputText == "")
        return 0;
}
```

- **Ответ: нет**
- **Пояснение: кажется, что, все варианты учтены, но нужно, чтобы был либо безусловный return в конце метода, либо во вложенном if ветка else. В приведённом коде получается, что не все ветки учтены (хотя они и учтены логически)**

Контрольный вопрос

Дана переменная `float num;`

Как можно проинициализировать num *, чтобы код*

```
if (num == num)
    Console.WriteLine("Equal");
else
    Console.WriteLine("Not equal");
```

Выдал результат

Not equal

- **Пояснение:**
- **В результате действия например, оператора 0 / 0.0F получается результат, который не является числом (NAN - not a number), поэтому выражение (num == num) нельзя рассматривать с логической точки зрения**

Итерационные операторы

- Цикл while
- Цикл do-while
- Цикл for
- Цикл foreach

Цикл while

- Выполняет инструкции на основе логического условия
- Проверяет выполнение логического условия в начале цикла
- Выполняет инструкции до тех пор, пока условное выражение возвращает значение ИСТИНА

```
int i = 0;
while (i < 10) {
    Console.WriteLine(i);
    i++;
}
```

```
0 1 2 3 4 5 6 7 8 9
```

Цикл do-while

- Выполняет инструкции на основе логического условия
- Проверяет выполнение логического условия в конце цикла
- Выполняет инструкции до тех пор, пока условное выражение возвращает значение ИСТИНА

```
int i = 0;
do {
    Console.WriteLine(i);
    i++;
} while (i < 10);
```

0 1 2 3 4 5 6 7 8 9

Цикл for

- **Синтаксис обновления обновляющей переменной помещается в начале цикла**

```
for (int i = 0; i < 10; i++) {
    Console.WriteLine(i);
}
```

0 1 2 3 4 5 6 7 8 9

- **Область видимости переменной, объявленной внутри цикла for ограничивается этим циклом**

```
for (int i = 0; i < 10; i++)
    Console.WriteLine(i);
Console.WriteLine(i); // Error: i is no longer in scope
```

- **В цикле for может использоваться несколько управляющих переменных**

```
for (int i = 0, j = 0; ... ; i++, j++)
```

Цикл foreach

- **Задайте тип и имя итерационной переменной**
- **Выполняет инструкции для каждого элемента класса коллекции**

```
ArrayList numbers = new ArrayList( );
for (int i = 0; i < 10; i++ ) {
    numbers.Add(i);
}

foreach (int number in numbers) {
    Console.WriteLine(number);
}
```

0 1 2 3 4 5 6 7 8 9

Вложенные циклы

- **Вложенными могут быть циклы любых типов: while, do while, for.**
- **Каждый внутренний цикл должен быть полностью вложен во все внешние циклы. "Пересечения" циклов не допускаются**

```
static void Main()
{
  for (int i = 1; i <= 4; ++i, Console.WriteLine())
  for (int j=1; j<=5; ++j)
  Console.Write(" " + 2);
}
```

Операторы перехода

- Оператор goto
- Операторы break и continue

Оператор goto

- Выполнение управления программой передается инструкции, указанной с помощью метки
- Может привести к созданию «спагетти-кода»

```
if (number % 2 == 0) goto Even;
Console.WriteLine("odd");
goto End;
Even:
Console.WriteLine("even");
End:;
```

Операторы break и continue

- **Оператор break организовывает «досрочный» выход из цикла**
- **Оператор continue организовывает «досрочный» выход из итерации**

```
int i = 0;
while (true) {
    Console.WriteLine(i);
    i++;
    if (i < 10)
        continue;
    else
        break;
}
```

```
int n = 100;
for (int i = 1; i <= n; i++)
  {
if (i % 2 == 0) continue;
 Console.WriteLine(i);
  }
```

Лабораторная работа 3.1: Использование выражений

Дополнительно:

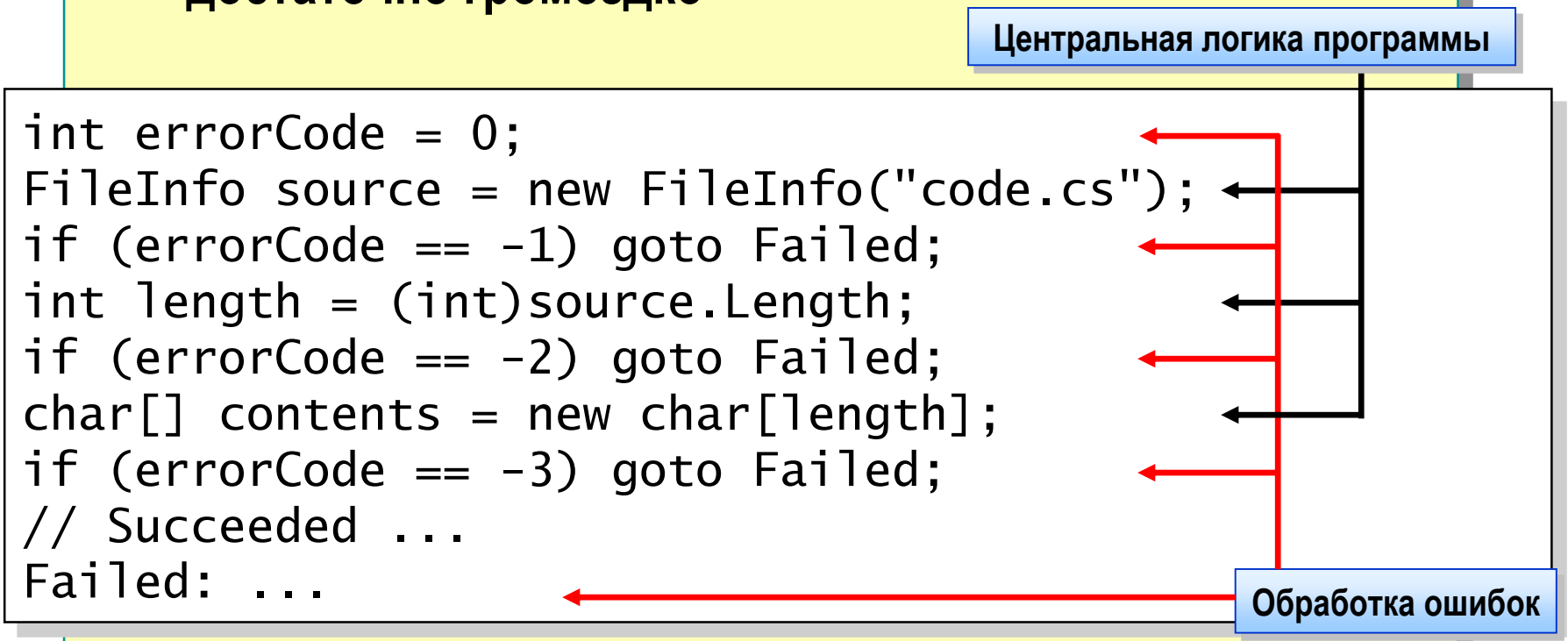
- **Вывести на экран: таблицу перевода 5, 10, 15, …, 100 долларов США в рубли по текущему курсу (значение курса вводится с клавиатуры);**
  - *Замечание*. Решите каждую задачу тремя способами – используя операторы цикла *while, do while* и *for*.
- **Написать программу, которая вводит с клавиатуры целое число и выдает на экран дисплея сумму цифр этого числа.**

Обработка исключений

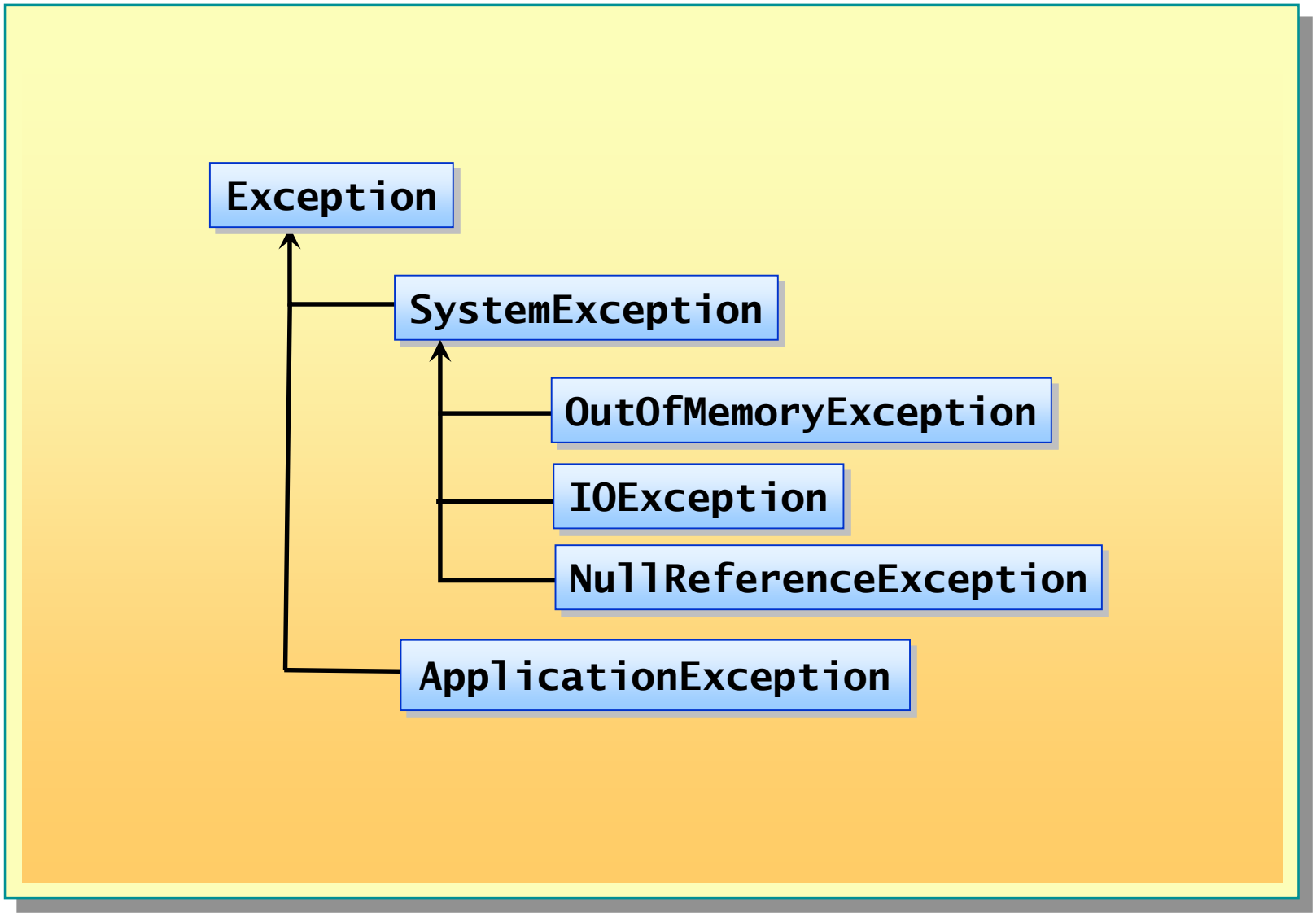
- Зачем нужны исключения?
- Класс Exception
- Использование try- и catch-блоков
- Использование нескольких catch-инструкций

Зачем нужны исключения?

- **Процедурная обработка ошибок выглядит достаточно громоздко**



Класс Exception



Возможности типов исключений

- **Понятное для пользователя сообщение с описанием ошибки.**
  - При происхождении исключения среда выполнения предоставляет текстовое сообщение, которое информирует пользователя о характере ошибки и предлагает способы ее устранения.
    - ➢ это текстовое сообщение содержится в свойстве **Message** объекта исключения
- **Состояние стека вызова при возникновении исключения.**
  - В свойстве **StackTrace** содержится трассировка стека, которую можно использовать для обнаружения места происхождения ошибки в коде.

Использование try- и catch-блоков

- **Объектно-ориентированный подход к обработке ошибок**
  - Программные инструкции, подлежащие проверке на наличие ошибок, помещают в try-блок
  - Исключения обрабатываются в отдельном catch-блоке

```
try {
      Console.WriteLine("Enter a number");
      int i = int.Parse(Console.ReadLine());
}
catch (OverflowException caught)
{
      Console.WriteLine(caught);
}
```

Программная логика

Обработка ошибок

Использование нескольких catch-инструкций

- Каждый catch-блок перехватывает один тип исключений
- Можно организовать перехват всех исключений
- Нельзя перехватывать тип исключений, который является производным от типа исключений, перехваченного в предыдущем catch-блоке

```
try
{
      Console.WriteLine("Enter first number");
      int i = int.Parse(Console.ReadLine());
      Console.WriteLine("Enter second number");
      int j = int.Parse(Console.ReadLine());
      int k = i / j;
}
catch (OverflowException caught) {…}
catch (DivideByZeroException caught) {…}
```

Обработчики стандартных исключений

Имя	Описание
ArithmeticException	Ошибка в арифметических операциях или преобразованиях
ArrayTypeMismatchException	Попытка сохранения в массиве элемента несовместимого типа
DivideByZeroException	Попытка деления на ноль
FormatException	Попытка передать в метод аргумент неверного формата
IndexOutOfRangeException	Индекс массива выходит за границу диапазона
InvalidCastException	Ошибка преобразования типа
OutOfMemoryException	Недостаточно памяти для нового объекта
OverflowException	Переполнение при выполнении арифметических операций
StackOverflowException	Переполнение стека

Вложенность обработчиков исключений

- **Один try-блок можно вложить в другой**
  - исключение, сгенерированное во внутреннем try-блоке и не перехваченное catch-инструкцией, которая связана с этим try-блоком, передается во внешний try-блок
  - внешний try-блок можно использовать для перехвата самых серьезных ошибок, позволяя внутренним try-блокам обрабатывать менее опасные.

Порядок проверки

Порядок следования в списке catch-блоков важен.

- **Вначале проверяются обработчики в порядке следования их за try-блоком, и первый потенциальный захватчик становится активным, захватывая исключение и выполняя его обработку.**
- **Первыми идут наиболее специализированные обработчики, далее по мере возрастания универсальности.**
  - Например, вначале должен идти обработчик исключения DivideByZeroException, а уже за ним - ArithmeticException.
- **Универсальный обработчик, если он есть, должен стоять последним.**

Генерирование исключений

- Инструкция throw
- Использование блока finally
- Переполнения в арифметических вычислениях

Инструкция throw

- **Выбрасывает соответствующее исключение**
- **Задавайте в исключениях осмысленные сообщения**

```
throw выражение ;
```

```
if (minute < 1 || minute >= 60) {
  throw new InvalidTimeException(minute +
                                 " is not a valid minute");
  // !! Not reached !!
}
```

Блок finally

- finally-блок выполняет освобождение ресурсов, занятых try-блоком
- Код в блоке finally исполняется всегда

```
Monitor.Enter(x);
try {
    ...
}
finally {
    Monitor.Exit(x);
}
```

Проверка на арифметическое переполнение

- **По умолчанию проверка на арифметическое переполнение не производится**
  - Инструкция checked включает проверку на арифметическое переполнение

```
checked {
    int number = int.MaxValue;
    Console.WriteLine(++number);
}
```

OverflowException

Выбрасывается объект Exception. WriteLine *не* выполняется.

```
unchecked {
    int number = int.MaxValue;
    Console.WriteLine(++number);
}
```

MaxValue + 1 отрицательно?

```
-2147483648
```

Формы операторов

- **Проверяет конкретное выражение и называется операторной checked-формой**

```
checked ((тип-выражения) expr)
```

- **Проверяет блок инструкций**

```
checked
{
 // Инструкции, подлежащие проверке
}
```

- **Игнорирует переполнение для заданного выражения**

```
unchecked ((тип-выражения) expr)
```

- **Игнорирует переполнение, которое возможно в блоке инструкций**

```
unchecked
{ // Инструкции, для которых переполнение игнорируется }
```

Контрольный вопрос

Что выведет на экран данный код?

```
class Program
{    static void Main(string[] args)
    {
        Console.WriteLine(Foo("Original string"));
    }
    public static string Foo(string str)
    {
        try
        {
            return str;
        }
        finally
        {
            Console.WriteLine("In finally");
            str = "Somewhere in finally";
        }    } }
```

Ответ: In finally
Original string

Лабораторная работа 3.2: Использование исключений

Дополнительно:

- **Постройте таблицу значений функции *y=f(x)* для x[a, b] с шагом h.**
  - варианты: $y = \ln(x-1)$ $y = \frac{1}{x^2 - 1}$
  - если в некоторой точке x функция не определена, то выведите на экран сообщение об этом.
  - проведите обработку возможных исключений.