

Тема 4. Операторы и исключения

Парадигмы программирования. C#

План

- Введение в операторы
- Использование оператора выбора и условного оператора
- Использование операторов цикла
- Использование операторов безусловного перехода
- Обработка простейших исключений
- Выброс исключений

Введение в операторы

Часть 1

Блоки операторов

- Используйте фигурные скобки для разделения кода на блоки

```
{  
    // code  
}
```

- Идентификаторы переменных вложенных блоков не должны совпадать

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

- Одноуровневые блоки могут содержать переменные с одинаковыми идентификаторами

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

Типы операторов

- Операторы выбора и условного перехода `if` и `switch`
- Операторы циклов `while`, `do`, `for` и `foreach`
- Операторы безусловного перехода `goto`, `break` и `continue`

Использование оператора выбора и условного оператора

Часть 2

Условный оператор

- Синтаксис

```
if ( Boolean-expression )  
    true-embedded-statement  
else  
    false-embedded-statement
```

- Допустимый тип аргумента – только bool

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

Каскадный if

```
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 выбора switch

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

- Не забывайте использовать `break`
- Вариант `default` не является обязательным
- Дополнительная возможность: использование `goto case Suit.Clubs;` для перехода на соответствующий вариант выбора

Использование операторов цикла

Часть 3

Цикл с предусловием

- Выполнение или невыполнение встроенных операторов осуществляется на основании значения выражения Булевского типа
- Булевское выражение вычисляется до начала выполнения цикла
- Цикл выполняется до тех пор, пока Булевское выражение принимает значение True

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

Формат цикла while

```
Подготовка цикла, инициализация  
while ( Булевское выражение )  
{  
    Встроенные операторы  
    Обновление выражения  
}
```

Цикл с постусловием

- Выполнение или невыполнение встроенных операторов осуществляется на основании значения выражения Булевского типа
- Булевское выражение вычисляется после выполнения первой итерации цикла
- Цикл выполняется до тех пор, пока Булевское выражение принимает значение True

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

Формат цикла do

```
Подготовка цикла, инициализация  
do  
{  
    Встроенные операторы  
    Обновление выражения  
} while ( Булевское выражение );
```

Цикл for

- Разновидность цикла с предусловием
- Обновление Булевского выражения размещено в начале цикла, но выполняется после завершения итерации

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

- Область переменной цикла ограничена циклом

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

- Цикл for может использовать несколько переменных цикла

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

Формат цикла for

Подготовка цикла

```
for ( инициализация переменной  
    цикла ; условие выполнения  
    следующей итерации ; обновление  
    переменной цикла )
```

```
{
```

Встроенные операторы

```
}
```


Вечные циклы

- `while (true) {}`
- `do {} while (true)`
- `for(;;) {}`

Цикл foreach

- Устанавливает тип и название переменной цикла, соответствующий элементу коллекции
- Тип переменной цикла может быть определён автоматически с помощью ключевого слова `var`
- Выполняет встроенные операторы для каждого элемента коллекции
- Значение переменной цикла нельзя модифицировать вручную
- Размер коллекции нельзя изменять во время работы цикла

```
ArrayList numbers = new ArrayList( );  
for (int i = 0; i < 10; i++)  
{  
    numbers.Add(i);  
}  
foreach (int number in numbers)  
    Console.WriteLine(number);
```

Использование операторов безусловного перехода

Часть 4

Оператор goto

- Управление переходит к оператору с меткой
- Результат неграмотного использования – запутанный код «спагетти»

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

Операторы break и continue

- Оператор break завершает работу цикла
- Оператор continue завершает работу текущей итерации цикла

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

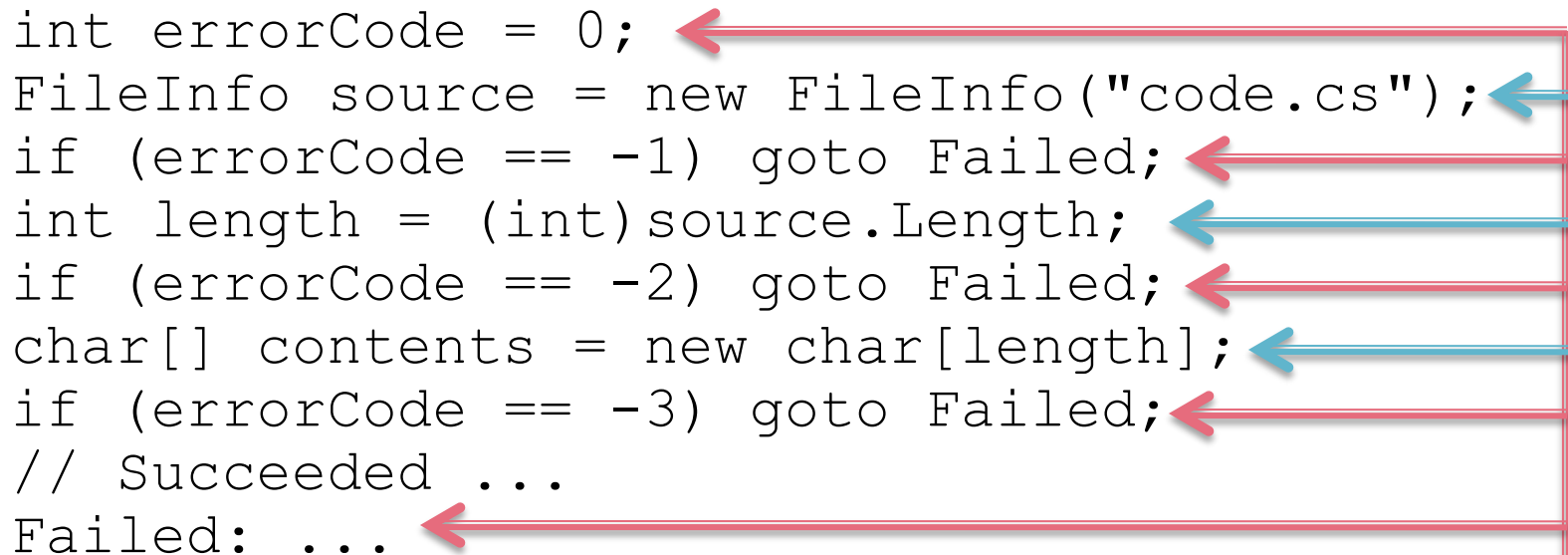
Обработка простейших исключений

Часть 5

Зачем нужны исключения?

- Традиционная процедурная обработка ошибок приводит к смешиванию кода основного алгоритма программы с кодом обработки ошибок

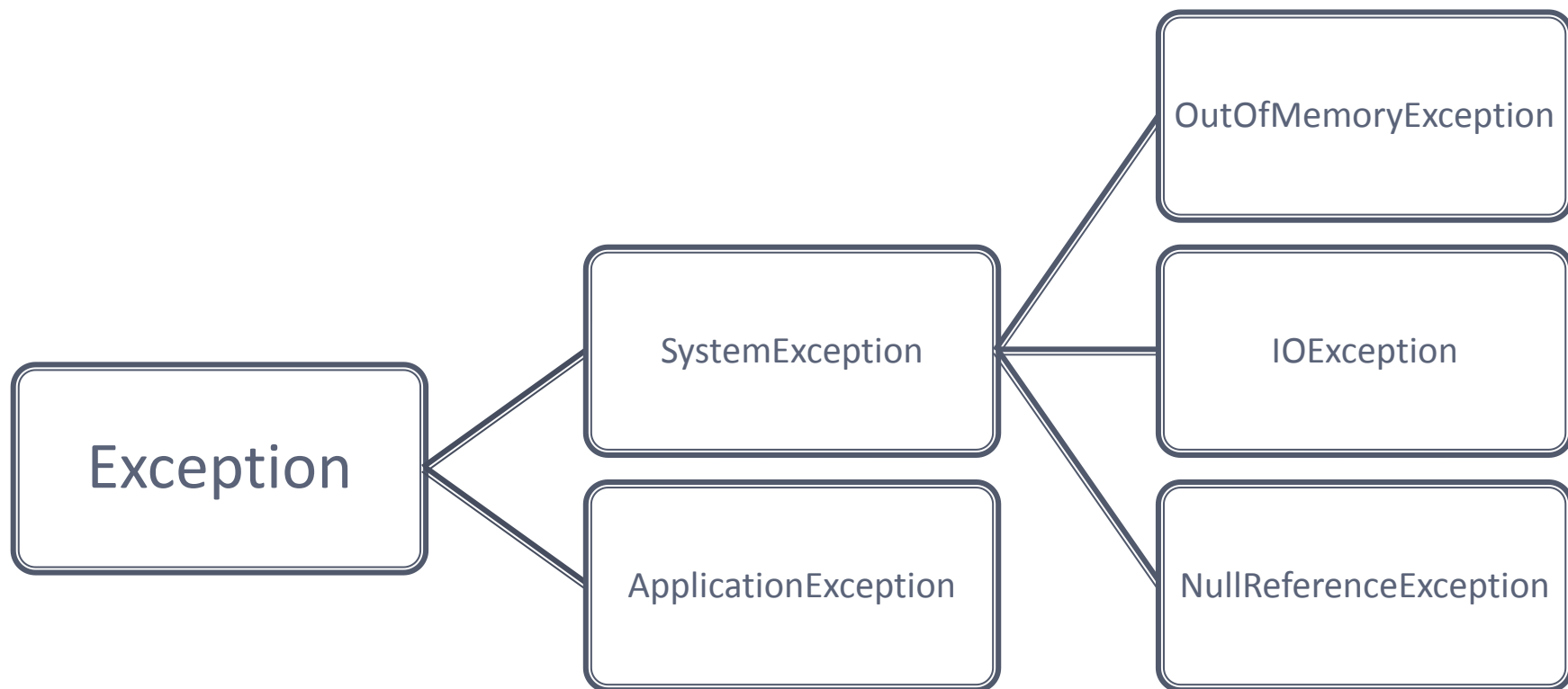
```
int errorCode = 0;
FileInfo source = new FileInfo("code.cs");
if (errorCode == -1) goto Failed;
int length = (int)source.Length;
if (errorCode == -2) goto Failed;
char[] contents = new char[length];
if (errorCode == -3) goto Failed;
// Succeeded ...
Failed: ...
```



Основной алгоритм

Обработка ошибок

Иерархия исключений



Использование блоков try catch

- Объектно-ориентированная обработка исключений
- Код основного алгоритма размещается в блоке `try`
- Код обработки исключений размещается в одном или нескольких блоках `catch`

```
try
{
    Console.Write ("Enter a number: ");
    int i = Int32.Parse(Console.ReadLine());
}
catch (OverflowException caught)
{
    Console.WriteLine(caught);
}
```

Использование нескольких блоков catch

- Каждый блок `catch` ловит исключение своего или производного класса
- Недопустимо размещать блок `catch` для класса исключения, производного от другого обрабатываемого класса исключения после обработчика базового исключения
- Исключения должны обрабатываться в порядке от наиболее специфичных к более общим
- Порядок блоков `catch` имеет значение
- Блок `try` может иметь не более одного блока `catch` без указания класса исключения, используемого для обработки всех оставшихся необработанных исключений

```
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) {...}
catch {...}
```

Выброс исключений

Часть 6

Оператор throw

- Выбрасывает исключение выбранного типа
- Добавляет к выбрасываемому исключению значимое сообщение
- Исключения можно выбрасывать из блоков catch

```
if (minute < 1 || minute >= 60)
{
    throw new InvalidTimeException(minute +
        " is not a valid minute");
    // This code is never reached
}
```

Блок finally

- Все операторы блока finally обязательно выполняются даже при наличии исключения
- Блоку try должен соответствовать либо один блок catch, либо один блок finally
- Совместное использование блоков catch и finally допустимо

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

Проверка арифметического переполнения

- По умолчанию арифметическое переполнение не проверяется
- Для безусловного включения проверки арифметического переполнения используется блок `checked`

```
checked
{
    int number = Int32.MaxValue;
    Console.WriteLine(++number);
}
```

- Обратный по действию блок – `unchecked`

```
unchecked
{
    int number = Int32.MaxValue;
    Console.WriteLine(++number);
}
```

- Альтернативный вариант включения и отключение проверки переполнения – использование опций компилятора

Рекомендации по использованию исключений

- Выбрасывание
 - Не используйте исключений для реализации основного алгоритма (сценария) работы
 - Никогда не выбрасывайте исключения класса `Exception`
 - Используйте наиболее подходящие исключения исходя из типа исключительной ситуации
 - Включайте строку описания в выбрасываемое исключение
- Отлавливание
 - Распределяйте блоки `catch` в порядке от наиболее специфичных исключений к более общим
 - Не давайте исключениям возможность выпасть из метода `Main`
 - Предусмотрите механизм обработки необработанных исключений

Заключение

- Спасибо за внимание!