Исключения и модульное тестирование

Юрий Литвинов

3

Бросание исключений

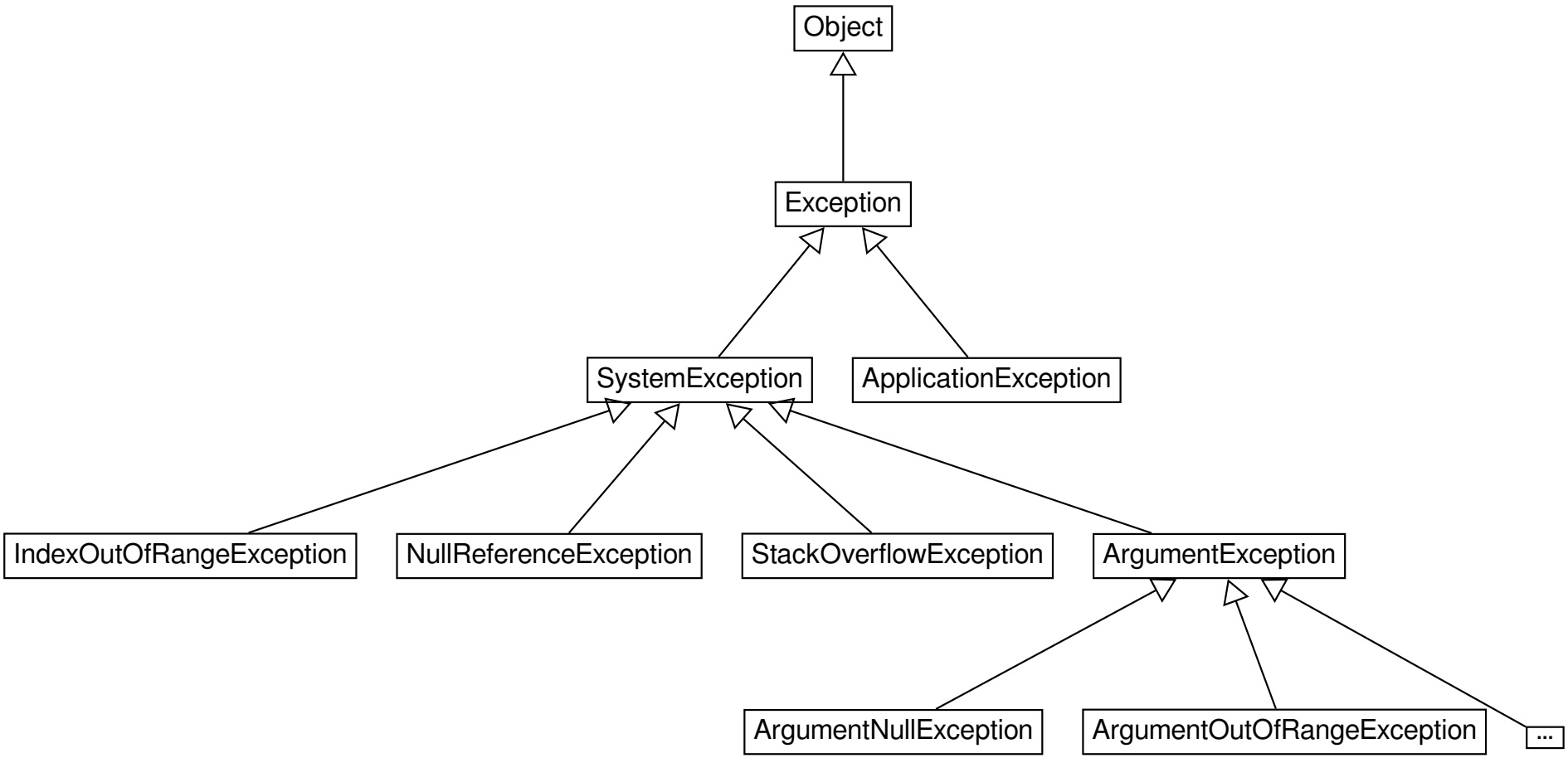
```
if (t == null)
{
    throw new NullReferenceException();
}

throw new NullReferenceException("Hello!");
```

Обработка исключений

```
try {
    // Код, который может бросать исключения
} catch (Type1 id1) {
    // Обработка исключений типа Type1
} catch (Type2 id2) {
    // Обработка исключений типа Type2
} catch (Type3 id3) {
    // Обработка исключений типа Type3
} finally {
    // Код, выполняющийся в любом случае
}
```

Иерархия классов-исключений



Свойства класса Exception

- Data
- HelpLink
- InnerException
- Message
- Source
- StackTrace

Пример (плохой)

```
try
{
    throw new Exception("Something is very wrong");
}
catch (Exception e)
{
    Console.WriteLine("Caught Exception");
    Console.WriteLine("e.Message: " + e.Message);
    Console.WriteLine("e.ToString(): " + e.ToString());
    Console.WriteLine("e.StackTrace:\n" + e.StackTrace);
}
```

Перебрасывание исключений

```
try
{
    throw new Exception("Something is wrong");
}
catch (Exception e)
{
    Console.WriteLine("Caught Exception");
    throw;
}
```

Или

```
throw new Exception("Outer exception", e);
```

Свои классы-исключения

```
public class MyException : ApplicationException
{
    public MyException()
    {
    }

    public MyException(string message)
        : base(message)
    {
    }
}
```

Идеологически правильное объявление исключения

```
[Serializable]
public class MyException : Exception
{
    public MyException() { }
    public MyException(string message) : base(message) { }
    public MyException(string message, Exception inner)
        : base(message, inner) { }
    protected MyException(
        System.Runtime.Serialization.SerializationInfo info,
        System.Runtime.Serialization.StreamingContext context)
            : base(info, context) { }
}
```

Исключения, best practices

- Не бросать исключения без нужды
  - В нормальном режиме работы исключения бросаться не должны вообще
- Свои исключения наследовать от System.Exception
- Документировать все свои исключения, бросаемые методом
- Не ловить Exception, SystemException
  - Исключения, указывающие на ошибку в коде (например, NullReferenceException) уж точно не ловить
- По возможности кидать библиотечные исключения или их наследников:
  - InvalidOperationException
  - ArgumentException
- Имя класса должно заканчиваться на “Exception”
- Код должен быть безопасным с точки зрения исключений
  - Не забывать про finally или using

Модульное тестирование

- Помогает искать ошибки
  - Особо эффективно, если налажен процесс Continuous Integration
- Облегчает изменение программы, рефакторинг
  - Но несколько замедляет процесс, тесты тоже требуют сопровождения
- Тесты — документация к коду
- Тесты помогают улучшить архитектуру, спагетти-код не оттестировать

Популярные библиотеки

- NUnit
  - Отдельный пакет
  - Интегрируется в IDE расширениями
- Microsoft Unit Test Framework
  - Работает прямо из коробки в Visual Studio, но требует некоторой возни, если нет
- XUnit, MbUnit?

Best practices

- Независимость тестов
  - Желательно, чтобы поломка одного куска функциональности ломала один тест
- Тесты должны работать быстро
  - И запускаться после каждой сборки
    - Continuous Integration!
- Тестов должно быть много
  - Следить за Code coverage, который многие инструменты умеют считать по тестовому прогону
- Каждый тест должен проверять конкретный тестовый сценарий
  - Никаких try-catch внутри теста
  - Атрибут ExpectedException для исключений
    - [ExpectedException(typeof(NullReferenceException))]
- Test-driven development

Mock-объекты

- Объекты-заглушки, симулирующие поведение реальных объектов и контролирующие обращения к своим методам
  - Как правило, такие объекты создаются с помощью библиотек
- Используются, когда реальные объекты использовать
  - Слишком долго
  - Слишком опасно
  - Слишком трудно
  - Для добавления детерминизма в тестовый сценарий
  - Пока реального объекта ещё нет
  - Для изоляции тестируемого объекта
- Для mock-объекта требуется, чтобы был интерфейс, который он мог бы реализовать, и какой-то механизм внедрения объекта
  - Паттерны “Dependency Injection”, “Стратегия”

Популярные библиотеки

- NSubstitute
- Moq
- Rhino Mocks
- ...

NSubstitute

Легковесная библиотека, очень удобно описывать поведение объектов:

```
var stackStub = Substitute.For<IStack>();
stackStub.IsEmpty().Returns(false);
stackStub.Pop().Returns(1);

Assert.AreEqual(1, stackStub.Pop());

stackStub.Received().Pop();
stackStub.DidNotReceive().Push(Arg.Any<int>());
```

Moq

Активно использует лямбда-функции и LINQ:

```
var stackMock = new Mock<IStack>();
stackMock.Setup(st => st.IsEmpty()).Returns(false);
stackMock.Setup(st => st.Pop()).Returns(1);

var stack = stackMock.Object;
stack.Push(1);
stack.Pop();

stackMock.Verify(st => st.IsEmpty(), Times.Never);
stackMock.Verify(st => st.Pop(), Times.Exactly(1));
stackMock.Verify(st => st.Push(It.IsAny<int>()), Times.Exactly(1));
```

Moq (2)

Или так (LINQ-магия):

```
var stack = Mock.Of<IStack>(
      st => st.IsEmpty() == false && st.Pop() == 1);

stack.Push(1);
stack.Pop();

Mock.Get(stack).Verify(st => st.Push(It.IsAny<int>()), Times.Exactly(1));
Mock.Get(stack).Verify(st => st.Pop(), Times.Exactly(1));
Mock.Get(stack).Verify(st => st.IsEmpty(), Times.Never);
```

Rhino Mocks

Пару слов про эту библиотеку

```
var stack = MockRepository.Mock<IStack>();
stack.Expect(st => st.Push(1));
stack.Expect(st => st.Pop()).Returns(() => 1);

stack.Push(1);
Assert.AreEqual(1, stack.Pop());

stack.VerifyAllExpectations();
```