Языки программирования
Лекция 7

Обработка исключений

- Иногда при выполнении программы возникают ошибки, которые трудно предусмотреть или предвидеть, а иногда и вовсе невозможно.
- Такие ситуации называются **исключениями**.
- Язык C# предоставляет разработчикам возможности для обработки таких ситуаций.
- Для этого в C# предназначена конструкция **try...catch...finally**.

Конструкция try..catch..finally

- При использовании блока **try...catch..finally** вначале выполняются все инструкции в блоке **try**. Если в этом блоке не возникло исключений, то после его выполнения начинает выполняться блок **finally**. И затем конструкция try..catch..finally завершает свою работу.
- Если же в блоке try вдруг возникает исключение, то обычный порядок выполнения останавливается, и среда CLR начинает искать блок **catch**, который может обработать данное исключение. Если нужный блок catch найден, то он выполняется, и после его завершения выполняется блок finally.
- Если нужный блок catch не найден, то при возникновении исключения программа аварийно завершает свое выполнение.

Типы исключений. Класс Exception

Базовым для всех типов исключений является тип **Exception**. Этот тип определяет ряд свойств, с помощью которых можно получить информацию об исключении.

- **InnerException**: хранит информацию об исключении, которое послужило причиной текущего исключения
- **Message**: хранит сообщение об исключении, текст ошибки
- **Source**: хранит имя объекта или сборки, которое вызвало исключение
- **StackTrace**: возвращает строковое представление стека вызывов, которые привели к возникновению исключения
- **TargetSite**: возвращает метод, в котором и было вызвано исключение

Типы исключений. Класс Exception

Однако так как тип Exception является базовым типом для всех исключений, то выражение catch (Exception ex) будет обрабатывать все исключения, которые могут возникнуть.

Но также есть более специализированные типы исключений, которые предназначены для обработки каких-то определенных видов исключений:

- DivideByZeroException: представляет исключение, которое генерируется при делении на ноль
- ArgumentOutOfRangeException: генерируется, если значение аргумента находится вне диапазона допустимых значений
- ArgumentException: генерируется, если в метод для параметра передается некорректное значение
- IndexOutOfRangeException: генерируется, если индекс элемента массива или коллекции находится вне диапазона допустимых значений
- InvalidCastException: генерируется при попытке произвести недопустимые преобразования типов
- NullReferenceException: генерируется при попытке обращения к объекту, который равен null (то есть по сути неопределен)

И при необходимости мы можем разграничить обработку различных типов исключений, включив дополнительные блоки catch

Создание классов исключений

- Если нас не устраивают встроенные типы исключений, то мы можем создать свои типы. Базовым классом для всех исключений является класс Exception, соответственно для создания своих типов мы можем унаследовать данный класс.

Упаковка (boxing) и **распаковка (unboxing)**.

- Представляют собой механизм преобразования размерных типов данных языка C# из значимых в ссылочные и обратно через задействование свойств фундаментального базового класса *Object*.

Упаковка

- В языке C# упаковка работает как неявное преобразование экземпляра любого размерного типа в объект базового класса *Object*, которое происходит, когда компилятор наталкивается на значимый тип в том контексте, где ожидается появление ссылки.
- Это преобразование автоматически производится библиотекой CLR причём его выполнение не зависит от того, какой именно тип передан как входной — ссылочный или значимый.

Алгоритм упаковки

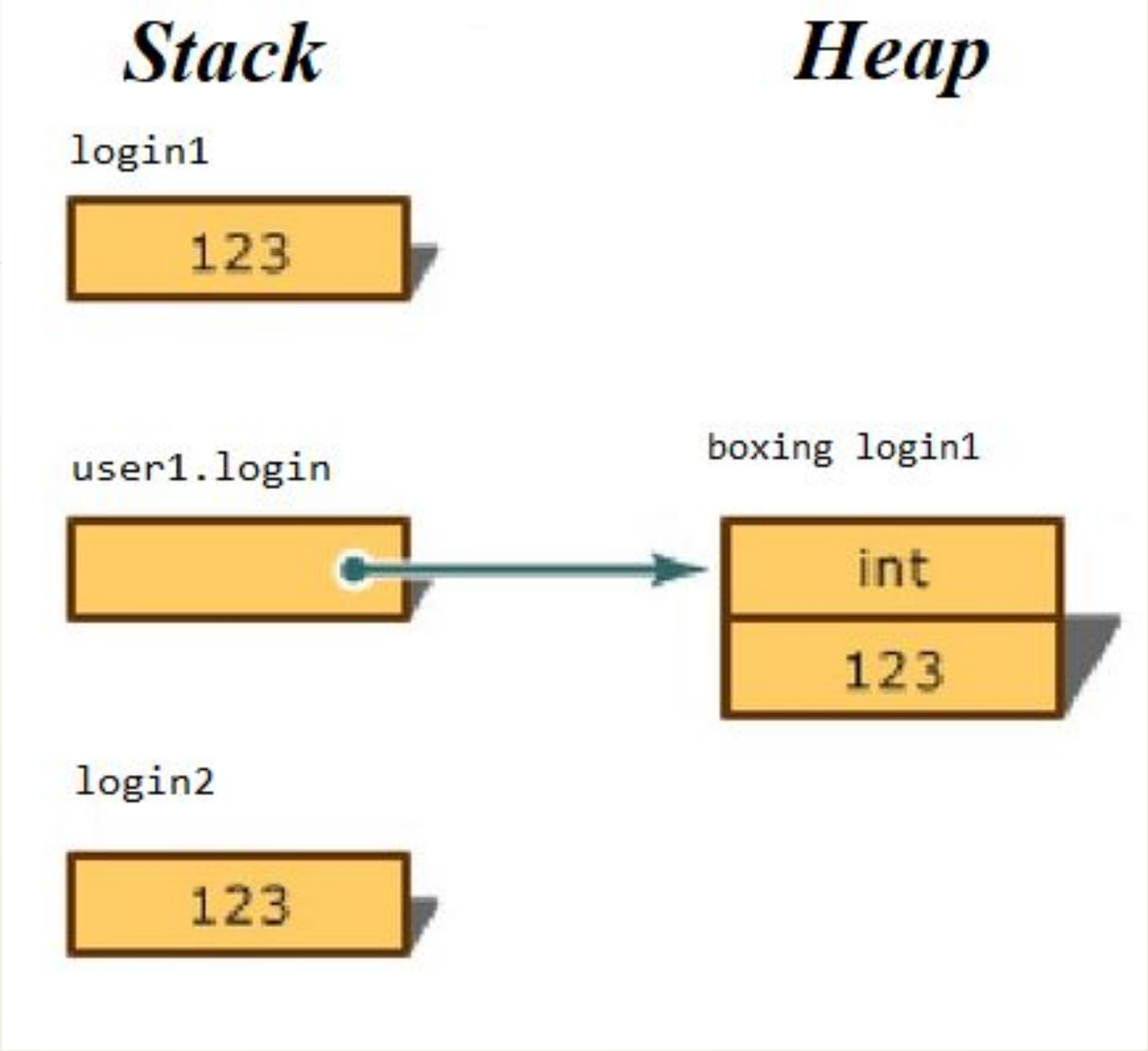
Трансформация значимого типа в ссылочный осуществляется автоматически, для этого выполняются следующие шаги:

- выделение необходимых ресурсов памяти для размещения нового объекта в «куче».
- поля значимого типа копируются в динамически выделенную память.
- возвращается адрес полученного объекта в виде ссылочного типа.

Распаковка

- Распаковка ссылочного типа в значимый подразумевает, что это должно быть выполнено явно. При этом, необходимо во-первых, сначала удостовериться, что тип упакованного объекта по ссылке соответствует исходному, а во-вторых, скопировать поля данных упакованного объекта в новую переменную данного типа. Как правило, проверку соответствия типов осуществляют с помощью механизма генерирования и обработки исключений, после чего копирование переносит внутренние данные (поля) объекта из «кучи» в стек выполняемого приложения, где хранятся его локальные переменные. Последовательность конкретных действий сводится к следующим шагам:
- если служебный указатель на упакованный значимый тип имеет значение *null*, то генерируется исключение *NullReferenceException*,
- если упакованный объект не соответствует требуемому типу, то выбрасывается исключение *InvalidCastException*.

Отмечается, что распаковка не является противоположностью упаковки в строгом смысле этого слова, она гораздо менее ресурсоёмка если состоит только в запросе указателя на исходный значимый тип.


Stack
Heap
login1
123
user1.login
boxing login1
int
123
login2
123