


Programowanie w .NET – OOP2

Jan Polak
Poznań/13-03-2015
Wersja 1.0

- 
1. Zasięg statyczny
 2. Wyjątki
 3. Typy generyczne

Zasięg statyczny

- Wiąże byt z typem, a nie którymkolwiek jego obiektem („static” jest wspólny dla wszystkich obiektów)
- Modyfikator „static” może być użyty dla:
 - Klas
 - Składowych klasy:
 - Pola
 - Metody
 - Właściwości
 - Operatory
 - Zdarzenia
 - Konstruktory
- Modyfikator „static” nie może być użyty dla:
 - Indexerów
 - Destruktorów
 - W ramach innych typów niż klasy

Zasięg statyczny – dostęp

- Dostęp do statycznej składowej obiektu poza jego ciałem odbywa się za pomocą odwołania do typu
Przykład: `string.Format(...)` //wywołanie statycznej metody `format`
- W ramach ciała klasy przy dostępie do składowych statycznych:
 - Nie można odwoływać przez „this”
 - Nie trzeba używać przedrostka typu
- Wewnątrz metody statycznej nie można korzystać z obiektu „this”
- Wewnątrz metody statycznej można korzystać ze składowych niestatycznych:
 - Tylko dla przekazanego obiektu (np. jako parametr metody)
 - Modyfikatory widoczności zachowują działanie w przypadku korzystania ze składowych innych klas

Klasy statyczne

- ❑ Modyfikator „static” jest użyty przed słowem kluczowym „class”
- ❑ Mogą zawierać tylko składowe statyczne
- ❑ Nie można powołać ich obiektu
- ❑ Nie można po nich dziedziczyć
- ❑ Nie mogą zawierać niestatycznych konstruktorów
- ❑ Tylko jeden egzemplarz składowych statycznych istnieje w systemie
- ❑ Statyczny konstruktor:
 - Nie ma modyfikatorów dostępu ani parametrów
 - Jest wywoływany przy pierwszym dostępie do obiektu
 - Nie może być wywołany z kodu programu
 - Użytkownik nie ma wpływu na jego wykonanie
 - Jeżeli wyrzuci wyjątek, to typ nie może być zainicjalizowany

Przykład: StringHelper

- Klasa jest publiczna i statyczna
- Zawiera publiczną statyczną metodę o nazwie `IsValid`, która:
 - Zwraca typ logiczny (`bool`)
 - Jako parametr przyjmuje tekst (`string`)
 - Ma zaimplementowaną logikę:
 - Jeżeli tekst przekazany tekst nie istnieje lub zawiera tylko białe znaki (`string.IsNullOrEmpty`) zwracany jest `false`
 - Jeżeli długość przekazanego tekstu (właściwość `Length`) jest mniejsza od 3 to zwracany jest `false`
 - W innych przypadkach zwracana jest `true`

Wyjątki

- Zunifikowany mechanizm pozwalający na obsługę błędów
- CLR „powiadamia” aplikację o błędach poprzez wyrzucenie odpowiedniego wyjątku
- Możliwe przyczyny:
 - Błąd w kodzie programisty (np. dzielenie przez zero)
 - Błąd w zewnętrznej bibliotece (np. niezgodność wersji komponentu)
 - Niedostępne zasoby systemu operacyjnego (np. dysk pełen)
 - Nastąpiły nieoczekiwane warunki wykonania kodu programu (np. brak pamięci)
- Wyjątki z kodu zarządzanego (np. C#) i kodu niezarządzanego (np. C++) są traktowane na tych samych zasadach za pomocą mechanizmu SEH
- Mogą być przerzucane pomiędzy procesami i maszynami w sieci

Wyjątek – składowe

- Każdy wyjątek dziedziczony jest po klasie Exception i zawiera składowe:
 - Właściwość StackTrace – stos kolejnych wywołanych metod gdzie błąd nastąpił
 - Właściwość InnerException – inny wyjątek, którego obsłużenie spowodowało wyrzucenie głównego wyjątku
 - Właściwość Message – wiadomość o powodzenie błędu
- Wyjątek może być dziedziczony i rozszerzany przez innej składowe

Hierarchia wyjątków – przykłady

- `IndexOutOfRangeException` – przekroczenie rozmiaru tablicy
- `NullReferenceException` – odwołanie do wartości typu `null`
- `AccessViolationException` – odwołanie do złego obszaru pamięci (wskaźniki)
- `InvalidOperationException` – nieprawidłowa operacja wywołana
- `ArgumentNullException` – przekazany argument powinien mieć wartość
- `ArgumentOutOfRangeException` – odwołanie do argumentu spoza zakresu
- `ComException` – błąd wywołany operacjami na obiekcie typu COM
- `SEHException` – błąd wywołany operacjami na obiekcie niezarządzanym

„Nieobsługiwane” wyjątki:

- `OutOfMemoryException` – błąd wywołany brakiem pamięci
- `StackOverflowException` – błąd wywołany przez zbyt dużą liczbę wywołanych metod na stosie (np. przy rekurencji)

Wyrzucenie wyjątku

- Do wyrzucenia wyjątku służy instrukcja „throw”
- Można wrzucić wcześniej złapany wyjątek
 - Najlepiej uzupełniony dodatkowymi informacjami
 - Wyrzucenie niezmodyfikowanego wyjątku jest za pomocą instrukcji „throw;”
- Zazwyczaj jednocześnie tworzony jest nowy wyjątek i wyrzucany
Przykład: `throw new TextValidationException("Surname is invalid")`

Obsługa wyrzuconego wyjątku

- Do obsługi wyjątku służy blok instrukcji try – catch – finally:
 - Kod, który może wyrzucić błąd znajduje się w sekcji try
 - Obsługa wyjątku jest zawarta w sekcji catch:
 - Kaskadowo zgodnie z zasadami dziedziczenia od najbardziej szczegółowych po najbardziej ogólnych
 - Obsługa wszystkich wyjątków
 - Obsługa kodu, który zawsze zostanie wykonany niezależnie od wystąpienia wyjątku znajduje się w sekcji finally
- W przypadku obsługi błędu to w kodzie programu:
 - Kod wyrzucający wyjątek musi być umieszczony w sekcji „try”
 - Musi istnieć jedna lub więcej sekcji „catch” lub jedna sekcja „finally”
- Nieobsłużony błąd propaguje się – w skrajnych przypadkach do systemu operacyjnego, powodując zamknięcie programu
- Obsługę błędów można zagnieżdżać w sobie

Dobre praktyki podczas pracy z wyjątkami

- Zapobieganie wyjątkom przez sprawdzenie warunku (np. czy dzielnik nie jest zerem) przed wykonaniem kodu
- Odpowiednie wskazanie wyjątku w sekcji „catch”
- Wyrzucanie wyjątków wywiedzionych z odpowiednich klas (patrz slajd Hierarchia wyjątków)
- Komunikat w wyjątku powinien być jasno i poprawnie sformułowany
- Wyjątki powinny zawierać wszystkie konstruktory z klasy Exception
- Wyjątki powinny być umieszczane we współdzielonych bibliotekach

Typy generyczne (parametryzowane)

- Pozwalają na ponowne użycie typów i operacji
- Najczęściej używane w kolekcjach oraz metodach operujących na nich (np. Kolekcja<Miejsce>)
- Typ ma specjalne „miejsce”, w które zostanie wstawiony typ
 - Wstawianie odbywa się w trakcie pracy programu
 - Wstawiany typ musi dać się skonstruować – to nie może być kolejny „niepełny” typ generyczny
 - Po wstawieniu powstaje nowy typ (np. Kolekcja<int> to nie to samo co Kolekcja<Osoba>)
 - Powstały typ jest „zoptymalizowany” pod konkretne zastosowanie (typ użyty do wypełnienia pustego „miejsca”)
- Typy możliwe do użycia mogą posiada ograniczenia:
 - Jakiego rodzaju:
 - Proste (where Miejsce : struct)
 - Referencyjne (where Miejsce : class)
 - Po czym dziedziczą lub co implementują (where Miejsce : Exception)
 - Jaki mają konstruktor, np. bezparametrowy (where Miejsce : new())

Typy generyczne (parametryzowane) – implementacja

- Typ generyczny podaje się po nazwie klasy i w nawiasie ostrym
 - Opcjonalne ograniczenia następują po nawiasie ostrym (i ewentualnym dziedziczeniu lub implementacji wskazanych interfejsów)
 - Może istnieć wiele typów generycznych
 - W ramach ciała klasy/metody odwołanie do typu użytego do parametryzacji następuje jak do każdego innego, znanego typu
 - Obiekt typu użytego do parametryzacji dostarcza dostęp do składowych (public/internal)
 - Należy podać typ jaki ma być użyty do parametryzacji w momencie:
 - Tworzenia obiektu typu generycznego, np. `new Kolekcja<int>()`
 - Wywołania metody z typem generycznym np. `Sortuj<int>()`
 - Jeden typ generyczny może korzystać z typu użytego do parametryzacji innego typu
- Przykład `class SuperTyp<T> {
 private _zbior Kolekcja<T>;
 /*....*/
}`

Podsumowanie

Zostały omówione:

- Zasięg statyczny – klasa StringHelper
- Wyrzucanie wyjątków – klasy Person i BetterPerson
- Przesłonięcia wirtualnych metod – właściwości w ramach klas Person i BetterPerson
- Redefinicja niewirtualnych metod – GetUserFriendlyMessage
- Ponowne zastosowanie kodu („ponowna używalność”) i typy parametryzowane – klasa GenericDataValidationException