Wyjątki i odporny program

Czas: 2 × 45 min | praca indywidualna | Wykład 1.11

Cel: Napiszesz kalkulator, który nie kończy się błędem przy żadnych danych, obsługując wyjątki wbudowane i własny wyjątek.

Po zajęciach potrafisz:

- obsługuje wyjątki blokami `try`, `catch` i `finally`
- zgłasza wyjątek instrukcją `throw`
- definiuje własną klasę wyjątku
- projektuje komunikaty błędów zrozumiałe dla użytkownika

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Kalkulator bez zabezpieczeń

Utwórz projekt `Kalkulator`. Użytkownik wpisuje działanie w jednej linii: `12 / 4`.

1. Napisz klasę `Kalkulator` z metodą `decimal Oblicz(string wyrazenie)`: podział na 3 części, `decimal.Parse`, `switch` po znaku.
2. Uruchom kolejno: `12 / 4`, `1 / 0`, `abc`, `2 ^ 3`, `12/4` (bez spacji). Zapisz, jakie wyjątki przerwały program.

2 Obsługa wyjątków

Dodaj pętlę i obsługę błędów w programie głównym.

1. Obejmij wywołanie `Oblicz` blokiem `try` i dodaj osobne `catch` dla `FormatException`, `DivideByZeroException` i `OverflowException`.
2. Każdy komunikat ma mówić, co zrobić, np. „Liczby wpisz z przecinkiem, np. 2,5 * 4.”.
3. Dodaj blok `finally`, który wypisuje linię oddzielającą — sprawdź, że wykonuje się zawsze.
4. Gdy wyrażenie nie ma trzech części, zgłoś `throw new FormatException(...)`.

3 Własny wyjątek

Nieznany znak działania sygnalizuj własnym wyjątkiem.

```
class NieznaneDzialanieException(string znak) : Exception($"Nieznane działanie: {znak}")
{
    public string Znak { get; } = znak;
}
```

1. Utwórz klasę `NieznaneDzialanieException` dziedziczącą po `Exception` z właściwością `Znak`.
2. Zgłoś ją w ramieniu `_` wyrażenia `switch`.
3. Dodaj `catch` i komunikat z listą dozwolonych znaków.

4 Testy złych danych

Sprawdź odporność programu.

1. Przygotuj co najmniej 10 przypadków testowych (poprawne, błędne, graniczne, np. `79228162514264337593543950335 + 1`).
2. Upewnij się, że program nigdy się nie zatrzymuje z błędem.
3. Sprawdź, co się stanie z `double` zamiast `decimal` przy dzieleniu przez zero.
4. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod programu
- ☐ lista wyjątków z wersji bez zabezpieczeń
- ☐ tabela 10 testów
- ☐ wyjaśnienie różnicy w dzieleniu przez zero dla `double` i `decimal`

Pytania kontrolne

1. Kiedy wykonuje się blok `finally`?
2. Dlaczego nie warto pisać jednego `catch (Exception)` dla wszystkich błędów?
3. Dlaczego `1.0 / 0` dla `double` nie zgłasza wyjątku?
4. Kiedy lepiej sprawdzić dane warunkiem (`TryParse`), a kiedy użyć wyjątku?

Kryteria oceny

Kryterium	Punkty
Metoda Oblicz i analiza wersji bez zabezpieczeń	4
Obsługa wyjątków z czytelnymi komunikatami	5
finally i throw	2
Własny wyjątek	4
Tabela 10 testów i pytania	5
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).