Notatki do 3. tygodnia kursu Dotneta

Maciek Mielczarek

16 sierpnia 2020

1 Wstęp. Stan aplikacji po 2. tygodniu

U mnie jak i u przynajmniej kilkorga innych kursantów aplikacja ma przynajmniej 1 plik który ma kilkaset linii. Prawdopodobnie można to uznać za wyznacznik bałaganu, potrzeby refactoru i powtarzającego się kodu. Aplikacje jednak w jakimś stopniu działają.

2 Konstruktory

Stworzenie jakiegokolwiek konstruktora powoduje nie tworzenie konstruktora domyślnego.

Słowo this jest do wołania innych konstruktorów tej samej klasy. Wpisujemy je po dwukropku, jak wywołanie funkcji z parametrami, między listą parametrów a ciałem konstruktora. W tym miejscu możemy zamiast this użyć base, by wywołać konkretny konstruktor klasy bazowej.

Można zainicjalizować pola zaraz za wywołaniem konstruktora domyślnego, w nawiasach klamrowych.

3 Przeciążanie metod

W różnych wariantach metody o danej nazwie zwracamy ten sam typ.

Pojawiła się wzmianka o podpowiedziach IDE na temat przeciążonych metod. To całkiem przydatna funkcja, szczególnie gdy nie jesteśmy pewni jakich dokładnie danych potrzebuje dana funkcja, najczęściej napisana przez kogoś innego (np. twórców bibliotek standardowych).

4 Dziedziczenie

O dziedziczenie często pyta się na rozmowach rekrutacyjnych.

Przy dziedziczeniu klasa pochodna musi mieć konstruktory takie jak w klasie bazowej. Inne metody też mogą wołać metody klasy bazowej, w swoich ciałach poprzez base.Metoda().

W C# każda klasa może dziedziczyć tylko po 1 klasie.

W programowaniu sieciowym często spotyka się klasę bazową AuditableModel z właściwościami mówiącymi kto i kiedy ją stworzył i zmodyfikował. Po tej klasie dziedziczy wszystko co chcemy zapisywać do bazy danych.

5 Polimorfizm

Statyczny – przeciążanie funkcji i wybieranie wprost liczbą i typem argumentów która ma być wołana. Program wybiera funkcję na etapie kompilacji.

Dynamiczny – decyzja który kod zostanie wykonany jest odkładana do momentu wykonania kodu. Do nadpisywania metod z klas bazowych używamy modyfikatorów virtual – przy metodzie (w klasie bazowej) do nadpisania i override przy metodzie (w klasie potomnej) nadpisującej.

Jak to włożyć do swojego projektu? Zrobić klasę bazową z metodą wirtualną i 2 klasy dziedziczące po niej, nadpisujące tą metodę. Następnie trzymać obiekty typów potomnych w zmiennej (prawdopodobnie tablicy zmiennych) zadeklarowanej jako bazowa. Potem wołać tą wspólną metodę i użyć faktu, że dla obiektów każdej z klas potomnych będzie wołana jej wersja funkcji. Czyli jeśli mam w projekcie jakąś metodą wirtualną, to prawdopodobnie używam tego konceptu.

Aha, nie jestem pewien czy było to powiedziane wprost, ale obiekty typów pochodnych można trzymać w zmiennej typu bazowego.

new zamiast override mówi, że to jest nowa metoda, nie związana z metodą w klasie bazowej. To znaczy jeśli mamy obiekt typu pochodnego w zmiennej typu bazowego i wołamy tą "wspólną" metodę, wtedy, przy braku powiązania metody bazowej z pochodną, zostanie wywołana metoda z klasy bazowej.

6 Hermetyzacja

Niektóre pola (albo tylko ich settery) i metody są wewnętrznymi sprawami klasy bądź projektu, więc nie powinny być dostępne z zewnątrz. Hermetyzacja albo inaczej enkapsulacja polega właśnie na takim ustawieniu modyfikatorów dostępu, żeby sprawy wewnętrzne pozostawały wewnątrz.

7 Klasy abstrakcyjne

Modyfikator abstract przy klasie mówi, że nie da się tworzyć obiektów danej klasy. Służą one do tego, żeby po nich dziedziczyć, a więc po to żeby trzymać tam pola, właściwości i metody wspólne dla klas potomnych.

Modyfikator abstract przy metodzie mówi, że tej metody nie da się używać, służy ona tylko do tego, żeby ją nadpisywać w klasach potomnych. Czyli chcemy, żeby wszystkie klasy potomne miały metodę o danej nazwie i sygnaturze (czyli typach przyjmowanych i zwracanych), ale każda z nich może być inna.

Metoda abstrakcyjna nie może mieć ciała.

Nie nadpisanie metody abstrakcyjnej w klasie pochodnej zostanie wychwycone przez kompilator jako błąd.

Modyfikator sealed przy klasie mówi, że nie można po niej dziedziczyć. Ta możliwość jest stosowana bardzo rzadko.

8 Interfejsy