Notatki do 2. tygodnia kursu Dotneta

Maciek Mielczarek

11 sierpnia 2020

Spis treści

1 Wstęp **2**

2 Wybór projektu **2**

3 Start projektu, ekran startowy **3**

4 Gdzie wsadzić enuma i trochę struktury **4**

5 Typy referencyjne, stos i sterta **4**

6 Zadanie domowe z typów danych **4**

6.1 Zadanie 1. 5

6.2 Zadanie 2. 5

6.3 Zadanie 3. 6

6.4 Zadanie 4. 6

6.5 Zadanie 5. 6

7 Instrukcje warunkowe **6**

8 Operatory **7**

9 Zadania 2 **7**

10 Pętle i zadania 3 **8**

11 Instrukcje skoku **8**

12 Tablice i Listy **8**

13 Enum **9**

14 Klasy **9**

15 Parametry Metod **9**

16 Pola i właściwości **10**

17 Zakresy widoczności **10**

18 Częste błędy - podsumowanie **10**

19 Robienie projektu **10**

1 Wstęp

Na ten tydzień zaplanowane są podstawy języka C#. Najprawdopodobniej będę porównywał wprowadzane elementy do tego co znam z C++ albo Javy.

2 Wybór projektu

Nie wiem co wybrać, więc rzucę kostką. Na początek właściwe losowanie przy użyciu fizycznych kości k6 (6 ścian z numerami od 1 do 6). Są różne możliwości wylosowania liczby od 1 do 25 przy pomocy standardowych kostek. Zdecydowałem się na następujący wariant:

1. Z k6 robię k5 przez rzucenie ponownie w przypadku wyniku 6. Dzięki temu wszystkie powtórki rzutów załatwiam od razu.
2. 2 razy rzucam k5 i odejmuję od wyniku 1, aby dostać cyfrę dziesiątek (a właściwie piątek) i cyfrę jedności liczby w systemie piątkowym.
3. Przeliczam tą liczbę do systemu dziesiętnego (pierwsza cyfra razy 5 plus druga cyfra) i dodaję 1, żeby dostać liczbę z przedziału od 1 do 25.

Pierwszy rzut: 5. Drugi rzut: 3. To odpowiada liczbie 23, czyli grze w kółko i krzyżyk. To prawdopodobnie najprostszy temat, ale ponieważ istnieje wiele oczywistych wariantów tej gry, to będę mógł sprawdzić czy zaplanowałem i napisałem kod w taki sposób, żeby aplikację dało się rozwijać.

Skoro już jestem przy temacie rzucania kostką, to sprawdzę jak się losuje liczby w C# i odtworzę powyższą sytuację w kodzie. Pewnie wiele rzeczy jest nie na swoim miejscu lub zrobionych dziwnie, ale kod jest przetestowany i działa. Można go znaleźć tutaj.

3 Start projektu, ekran startowy

Projekt wybrany, czas stworzyć dla niego repozytorium na gicie i zacząć go robić. Do nauki lub przypomnienia sobie gita polecam to miejsce.

Na początku utworzyłem przy użyciu przeglądarki repozytorium do notatek i kodu z całego kursu. Następnie sklonowałem je i przeniosłem do stworzonego w ten sposób folderu już utworzone pliki. Następnie użyłem w tym folderze komendy "dotnet new gitignore" (ktoś chyba o tym wspomniał w okolicach tego kursu) i dodałem do nowego pliku kilka linii odpowiadających plikom pośrednim LaTeX-a. Potem kilka komend żeby wrzucić wszystko na Githuba, w razie błędów git podpowiadał co jest nie tak. To na Linuksie.

Na Windowsie klonowałem z poziomu Visual Studia. Zalogowałem się przy tym do Githuba z poziomu VS, dzięki czemu nie muszę wpisywać loginu i hasła przy każdym commicie. Następnie stworzyłem nowe Rozwiązanie (Solution) w pożądanym miejscu. Gdy już miałem otwarte w VS Rozwiązanie w folderze śledzonym przez Gita, to wszystko co Gitowe znalazłem w VS po środku prawej strony, po kliknięciu w napis "Team Explorer" obok napisu "Solution Explorer".

W pierwszej wersji programu (lekcja 2.) jest tylko ekran startowy z którego można wyjść. Zadbałem o to, żeby były tam już jakieś zmienne i stałe. O, jednak nie muszę ręcznie zamieniać stringów na inty, tak jak to robiłem w zabawie z kostkami powyżej. Tak mi się wydawało, że gdzieś w C# powinno być coś takiego jak Int32.Parse, czy tam TryParse, ale zanim to znalazłem, napisałem już swoją wersję (tylko bez obsługi błędów).

4 Gdzie wsadzić enuma i trochę struktury

W tym projekcie na ekranie głównym nie widzę zapotrzebowania na enumy (kolejny temat), więc potrzebuję już teraz dodać kilka klas.

Typ gracza komputerowego (łatwy/trudny) będzie enumem, więc pododaję klasy odpowiedzialne za typy graczy. Przy definiowaniu zależności między klasami jest parę słów kluczowych, których nie wiem czy dobrze używam, ale sprawdzę to przy okazji jakiegoś tematu o klasach. W VS łatwo tworzy się nowe klasy i interfejsy (kliknięcie na folder w Eksploratorze Rozwiązania -> dodaj nową pustą klasę/interfejs).

Niemal równie łatwo zapomnieć zmienić aktywną gałąź przy zaczynaniu nowego tematu.

5 Typy referencyjne, stos i sterta

Wiedziałem co to są typy referencyjne, ale nie wiedziałem co to stos i sterta. Rzeczy do sprawdzenia za chwilę:

- jak długi string mogę stworzyć
- czy zarówno \r\n jak i \n działają tak samo na Windowsie i Linuksie

Przy próbie stworzenia stringa długości 2^{30} program wywraca się z braku pamięci. W obu przypadkach \r\n i \n przechodzą na początek nowej linii, a samo \r wraca na początek bieżącej linii i kolejny napis nadpisuje początek tej linii. Czas na zadania.

6 Zadanie domowe z typów danych

Przy okazji tych zadań trochę sobie eksperymentuję. Zauważyłem, że gdy zaczynam nazwy folderów od cyfry, to automatycznie generowana przez dotneta przestrzeń nazw ma na początku znak "_". Może nie powinienem tego robić?

Przypomnienie stylu (do sprawdzenia w zadaniu domowym) z lekcji 1.9:

Wielkości liter:

- stałe wielkimi literami, słowa oddzielone znakiem "_",

- pliki, klasy, przestrzenie nazw i metody(funkcje) z wielkiej litery,
- zmienne lokalne i elementy prywatne programu z małej litery,
- słowa zaczynające się w środku nazwy zawsze z wielkiej litery,
- cała reszta małymi literami.

Nie ma tu nic o zaczynaniu nazwy katalogu od cyfry.

Inne:

- Początek nawiasu klamrowego od nowej linii.
- Max. jedna nowa linia między metodami/instrukcjami.
- Zawsze pisać modyfikator dostępu klas metod i pól (nie dotyczy zmiennych w metodach; tego pewnie w paru miejscach zapomniałem).
- Jedna klasa na plik.
- Brak typu zmiennej w nazwie, ale interfejsy zaczynają się od I.

6.1 Zadanie 1.

Zamiana stringa na zbiór cyfr w moim wykonaniu wygląda dość topornie – w pętli zajmuję się po kolei każdą cyfrą zamiast zawołać jakąś funkcję, która zrobiłaby to w 1 linijce. Zamiana chara na enuma też wygląda na dłuższą niż potrzeba.

6.2 Zadanie 2.

W zadaniu drugim chyba chodziło po prostu o 3 chary, ale nie byłem tego całkiem pewny, więc postanowiłem sprawdzić czy w C# używa się wskaźników.

Wygląda na to, że wszystkie użycia wskaźników trzeba oznaczyć jako kod niebezpieczny. Także raczej nie jest to szeroko stosowany element języka.

W każdym razie, udało mi się stworzyć 3 zmienne "a", wszystkie typu char (potem zapisać sobie ich adresy), a następnie wypisać dokładnie te zmienne (bez żadnego kopiowania wartości) w kolejności odwrotnej do kolejności tworzenia. Pomocne były tu nawiasy klamrowe, użyte do schowania jednych zmiennych przed drugimi.

Przypuszczam, że takie dodatkowe nawiasy czasem się stosuje do ograniczenia czasu życia lokalnych zmiennych.

6.3 Zadanie 3.

Tym razem postanowiłem sprawdzić jak przekazuje się do programu argumenty wywołania.

Przy wołaniu z linii komend po prostu wpisuje się je na końcu komendy wywołującej program, za dwoma średnikami. Znak ten oddziela argumenty dla dotneta od argumentów dla programu.

W VS opcja ta jest dostępna we właściwościach projektu (Solution Explorer -> nazwa projektu -> Alt + Enter -> Debug). Okienko to zamyka się w górnym lewym rogu.

6.4 Zadanie 4.

W tych zadaniach zauważałem czasem działające typy podstawowe pisane zarówno z wielkiej jak i z małej litery. O co chodzi?

Z tego co znalazłem, te pisane z małej to aliasy (inne nazwy) tych pisanych z wielkiej i z wyjątkiem typu enuma można ich używać zamiennie.

6.5 Zadanie 5.

Tutaj próbowałem powrzucać trochę kodu parsującego różne zmienne do jednej pętli, ale nie wychodziło mi używanie typu jako zmiennej. Poza tym, kod dla różnych typów zmiennych nie był taki sam bo parsowanie stringa do tablicy cyfr robi się inaczej niż do jednej liczby, a string w ogóle nie ma metody Parse (niezbyt zaskakujące, parsowanie stringa do stringa nie miałoby sensu).

Udało się tylko wypisać dane w pętli (żeby nie powtarzać kodu). Pomogła tu możliwość wsadzenia różnych typów zmiennych do jednej tablicy, jako zmiennych uniwersalnego typu object (znanym też jako Object).

7 Instrukcje warunkowe

Chyba wszystko tu działa jak w C++, więc ten materiał mam już przyswojony.

Może warto później sprawdzić czy są "krótkie spięcia" w && i ||. I czy jest w C# coś takiego jak Javovy try–catch z zasobami.

Ify w naturalny sposób pojawią się w aplikacji. Warto upewnić się że przy-

najmniej raz będzie wybór między więcej niż dwoma opcjami, żeby pojawił się jakiś switch. Mam już 3 opcje w pierwszym menu, więc ta sprawa jest już załatwiona.

O, czegoś takiego jak "przełącznik wyrażeniowy" chyba nie widziałem jeszcze w żadnym języku. Prawdopodobnie konstrukcja ta będzie miała jakieś naturalne zastosowanie w aplikacji, wystarczy tylko pamiętać o jej istnieniu. Do przećwiczenia jeśli nie pojawi się w najbliższych zadaniach.

8 Operatory

Tu bez niespodzianek. "Krótkie spięcia" istnieją, czyli nie będę potrzebował tego sprawdzać.

Jeśli są w C# operatory dla klas, to temat pewnie powróci przy klasach.

Przy używaniu operatorów niematematycznych, takich jak kropka, nawias kwadratowy czy rzutowanie naturalne jest tworzenie nadmiaru nawiasów. Jak zobaczę jakiś nadmiar nawiasów, to sprawdzę czy VS proponuje posprzątanie tego bałaganu. Właściwie, to czy w C# to też są operatory? Jestem prawie pewien że tak, (bo co innego miałoby to być) ale dopóki nie tworzę operatorów do własnych klas (o ile język to przewiduje), ta kwestia nie będzie istotna.

9 Zadania 2

Zdecydowałem się umieścić rozwiązania w metodach i sprawdzać je w pętli, żeby nie odpalać aplikacji od nowa za każdym sprawdzeniem.

VS podpowiada, że ?: w interpolowanym napisie trzeba wziąć w nawias. To chyba z powodu innych metod interpolacji, ale nie jestem pewien.

W temacie podpowiedzi VS-a, to czasem mogę je akceptować enterem albo tabem, a czasem tylko tabem. Co chwila próbuję zaakceptować dokończenie kodu enterem i zamiast tego przechodzę do nowej linii.

Jakoś udało się uniknąć zagłębiania w temat "co było przed kalendarzem Gregoriańskim".

Zostawiłem też chwilowo czytanie kilku zmiennych w jednej linii. Natomiast w zadaniu 11 udało się w końcu znaleźć zastosowanie dla tego nowego switcha.

Zauważyłem, że zakończenie nawiasu klamrowego w VS automatycznie na-

prawia wcięcia w tym nawiasie. I dodaje dodatkowe spacje dookoła operatorów po zakończeniu linii średnikiem. Warto mieć te fakty w pamięci, żeby nie robić czegoś, co za chwilę zrobi za nas VS.

10 Pętle i zadania 3

Pętle działają tak jak je pamiętam z C++ (ostatnio jest tam nawet pętla foreach, tyle że z nieco inną składnią). Prawdopodobnie warto natomiast przyswoić sobie skrót klawiaturowy "zakomentuj/odkomentuj linię lub linie kodu" (Ctrl+k+c/Ctrl+k+u). Aha, instrukcja break do wychodzenia z pętli też działa.

Dla szybszego sprawdzania wstawiłem do funkcji Main pętlę wołającą rozwiązania z możliwością wyboru numeru zadania i zakończeniem dopiero po wpisaniu 0 zamiast tego numeru. Dobrze byłoby sprawdzić jak w tym temacie próbowała sobie upraszczać życie reszta grupy.

11 Instrukcje skoku

Domyśliłem się że break i continue działają w C# i już używałem ich w zadaniach. Na wspomnienie goto w tym momencie kursu byłem nieco zaskoczony... przez te parę sekund które minęło do wspomnienia że się jej nie używa.

12 Tablice i Listy

Chyba wszystko działa albo tak jak pamiętam z innych języków albo jak już się domyśliłem i sprawdziłem przy robieniu zadań.

Jednak nie wszystko. Nie zauważyłem wcześniej klasy Array i jej przydatnych metod. Przy najbliższym użyciu którejś z metod tej klasy zapewne przejrzę listę dostępnych tam funkcji (Wyświetlą się po wpisaniu Array.). Bardzo przydatną własnością IDE jest w tym miejscu wyświetlanie krótkiego opisu każdej metody. Po wybraniu konkretnej metody można jeszcze poruszać się strzałkami w górę i w dół, aby poznać różne warianty tej funkcji.

Zapomniałem wcześniej wspomnieć, że wpisanie w IDE pierwszych liter metody/klasy/zmiennej itp. spowoduje domyślenie się przez IDE tego elementu.

Pisząc o IDE mam w tym rozdziale na myśli zarówno VS jaki i Monodevelopa.

Aha, czyli tablic chcę używać tylko gdy mam określoną liczbę obiektów i wolę poświęcić dodatkowe możliwości metod klasy List na rzecz szybkości działania programu.

13 Enum

Typ czegoś = enum. Fakt że enumy domyślnie konwertują się do stringów może się przydać.

Do sprawdzenia może kiedyś: Czy pod enumem zawsze kryje się int, czy czasem bywa to inny typ?

O, niedziela ma domyślnie numer 0.

14 Klasy

Klasa reprezentująca dane z bazy raczej nie powinna mieć metod. Dorabiamy do niej osobną klasę serwisową, która zajmie się manipulacją tymi danymi.

Statycznych klas pomocniczych używamy do robienia obliczeń i walidacji danych. Nie używać ich do manipulacji obiektami. Zresztą i tak chyba nie da się zmienić, odczytać czy wywołać pól i metod nie–statycznych z metod statycznych.

15 Parametry Metod

Przekazywanie do metody więcej niż 7 parametrów = stwórz z nich nową klasę.

Parametry domyślne dajemy na końcu.

Przy wywołaniu metody można podać wartości parametrów razem z nazwami i wtedy można zmienić kolejność parametrów.

Nowe słowa kluczowe: ref i out. out przy parametrze pisać i w deklaracji i w wywołaniu. To taka forma przekazania parametru przez referencję. ref to to samo co out, tylko nie pozwala na przekazywanie niezainicjalizowanej zmiennej jako argumentu. Raczej się ich nie używa, bo funkcja powinna zwracać jedną rzecz. A jednak out używa się do konwersji typów, w metodach TryParse.

16 Pola i właściwości

Właściwość to pole prywatne z getterem i setterem. Modyfikator dostępu przed właściwością odnosi się do jej gettera i settera. W setterze jest słowo kluczowe value, oznaczające przekazany argument. Można robić właściwości tylko do odczytu, albo przez niewpisywanie settera, albo przez "=> wartość do zwrócenia w getterze" od razu za nazwą właściwości.

17 Zakresy widoczności

Jeśli chcę mieć zależność między projektami, to klikam w solution manager -> projekt -> ppm na dependency -> add project reference.

public – Jak public w innych językach. Dostępne wszędzie gdzie widać daną klasę. Jeśli nie widzisz klasy, to dodaj using ta klasa, ewentualnie referencję w zależnościach.

private – Jak private w innych językach. Dostępne tylko z tej samej klasy.

protected – Dostępne z tej klasy i jej potomków.

internal – Dostępne w obrębie projektu. Domyślny dla klas.

18 Częste błędy - podsumowanie

Nie za dużo argumentów metod.

Dobrać odpowiedni typ do danych. Pytanie pomocnicze: co to jest w świecie rzeczywistym.

Sprawdzać czy użytkownik podał dane dobrego typu.

Uważać żeby nie robić wykluczających się warunków.

19 Robienie projektu

Zaczynamy od wypisania i rozpisania funkcjonalności w komentarzach, razem z pod–opcjami.

Potem zaczynamy je implementować i wykreślać to co już jest zrobione.

Klasa MenuAction z listą akcji w odpowiadającym jej serwisie to prawdopodobnie dobry pomysł. Ciekawe gdzie będą same dane. Aha, w metodzie Initialize, w tej samej klasie co Main.

Do wyboru menu będzie Console.ReadKey(), a nie Console.ReadLine().

Widzę zamianę chara na inta przez TryParse, czyli prawdopodobnie tak to się "ładnie" robi zamiast odejmowania znaku 0, a albo A.

Nie widzę dodatkowych nawiasów klamrowych w różnych gałęziach switcha, ale prawdopodobnie będę chciał je stosować żeby kompilator wiedział, że zmienne lokalne jednej gałęzi (case) nie są widoczne dla kodu z innej. Chodzi mi o nawias klamrowy zaczynający się zaraz za linią zaczynającą case i kończący się zaraz przed albo za jego instrukcją break.

Do sprawdzenia później: Czy można kończyć funkcję nic nie zwracającą instrukcją "return;"? Czy mogę wyczyścić konsolę z poziomu programu i czy mogę przejść do wcześniejszej linii?

Widzę sprawdzenie działania programu przez przejście przez niego linia po linii (breakpoint, step, step, step...).

Najpierw użycie funkcji, a potem pozwolenie IDE żeby ją stworzyło oszczędza nieco czasu, ale trzeba skontrolować wszystkie typy.

Uwaga żeby nie zadeklarować w środku pętli zmiennej która ma przeżywać przejście do nowego obrotu pętli.

Widzę słowa kluczowe this i params, które jeszcze się nie pojawiły. Pewnie niedługo dowiemy się między innymi o użyciu tych słów.