Implementacja interfejsów `IEquatable`, `IComparable`, `IComparer`

- Autor: *Krzysztof Molenda*
- Wersja: 2019-10-03

Krok 1 - realizacja klasy, implementacja pojęcia "taki sam"

1. Utwórz klasę `Pracownik` opisującą obiekt, zawierający:
   - `Nazwisko` (typu `string`),
   - `DataZatrudnienia` (typu `DateTime`),
   - `Wynagrodzenie` (typu `decimal`).
2. Dostęp do danych zrealizuj jako publiczne *properties* (read-write).
   - dla `Nazwisko` zaimplementuj obcinanie spacji przed i po
   - dla `DataZatrudnienia` zaimplementuj: data zatrudnienia nie później niż dzisiaj, w przeciwnym przypadku `throw new ArgumentException()`
   - dla `Wynagrodzenie` zaimplementuj: przy próbie podstawienia wartości ujemnej, przypisz `0`.
3. Przesłoń metodę `ToString()` zwracającą tekstową reprezentację obiektu w formie:

   `(Nazwisko, DataZatrudnienia, Wynagrodzenie)`
4. Zaimplementuj konstruktor (trójargumentowy). Zaimplementuj konstruktor domyślny (bezargumentowy), przyjmując nazwisko `"Anonim"`, datę przyjęcia dzisiaj oraz wynagrodzenie `0` PLN.
5. Zaimplementuj publiczne *property* `CzasZatrudnienia` zwracające aktualny czas zatrudnienia pracownika w pełnych miesiącach (miesiąc = 30 dni).
6. Zmodyfikuj metodę `ToString()` tak, aby po dacie zatrudnienia wypisywała czas zatrudnienia w miesiącach podany w nawiasie.

   `(Nazwisko, DataZatrudnienia (CzasZatrudnienia), Wynagrodzenie)`
7. Aby określić pojęcie "taki sam" zaimplementuj w klasie `Pracownik` interfejs `IEquatable<Pracownik>`. Dwa obiekty typu `Pracownik` są takie same, jeśli mają takie same nazwiska, daty zatrudnienia i wynagrodzenia. Powinieneś również przesłonić metodę `Equals()` oraz `GetHashCode()` w taki sposób, aby wyniki wszystkich były spójne. Przeczytaj:
   - https://docs.microsoft.com/en-us/dotnet/api/system.iequatable-1
   - https://blogs.msdn.microsoft.com/jaredpar/2009/01/15/if-you-implement-iequatablet-you-still-must-override-objects-equals-and-gethashcode/
8. Zaimplementuj statyczną metodę `Equals(Pracownik p1, Pracownik p2)` przesłaniając tę, zdefiniowaną w klasie `Object`.
9. Zaimplementuj przeciążenie operatora `==` (równocześnie musisz przeciążyć `!=`).
10. Przetestuj w `Main()` poprawność powyższych implementacji.

Kod po kroku 1

Krok 2 - implementacja naturalnego porządku w klasie

1. Utwórz listę 5 pracowników:
   - dwóch mających te same nazwiska,
   - dwóch zatrudnionych w tym samym roku i miesiącu,
   - dwóch mających to samo wynagrodzenie.
2. Wypisz tę listę w porządku oryginalnym.

3. Posortuj listę (metoda `Sort<T>` zdefiniowana w klasie `List<T>`). Dlaczego nie można tego zrobić?

4. Zadeklaruj implementację przez klasę `Pracownik` interfejsu `IComparable<Pracownik>` i zaimplementuj go. Ustal naturalny porządek w zbiorze pracowników: najpierw według nazwiska, potem według daty zatrudnienia i na końcu według wynagrodzenia - wszystko rosnąco.

5. Posortuj listę według naturalnego porządku zdefiniowanego w klasie `Pracownik`.

Kod po kroku 2

Krok 3 - wykorzystanie definicji zewnętrznego porządku sortowania

Dla listy utworzonej w poprzednim kroku wykonaj:

1. Posortuj listę najpierw według czasu zatrudnienia (w miesiącach), a później według wynagrodzenia - wszystko rosnąco:
   - utwórz klasę o nazwie `WgCzasuZatrudnieniaPotemWgWynagrodzeniaComparer` implementującą interfejs `IComparer<Pracownik>`,
   - posortuj listę (przeciążona metoda `Sort<T>` zdefiniowana w klasie `List<T>` wymagająca dostarczenia obiektu typu `IComparer`),
   - wypisz listę i sprawdź poprawność sortowania.
2. Posortuj listę najpierw według czasu zatrudnienia (w miesiącach), a później kolejno według nazwiska i wynagrodzenia:
   - posortuj listę (przeciążona metoda `Sort<T>` zdefiniowana w klasie `List<T>` wymagająca dostarczenia delegata typu `Comparison`),
   - wypisz listę i sprawdź poprawność sortowania.
3. Posortuj listę (metoda z `Comparison`) kolejno: malejąco według wynagrodzenia, później rosnąco według czasu zatrudnienia.

Kod po kroku 3

Krok 4 - porządkowanie z wykorzystaniem LINQ

Lista pracowników jest typu `List<Pracownik>`, ale również obiektem typu `IEnumerable`. Wykorzystaj metodę `OrderBy` oraz `ThenBy` z klasy `Enumerable` do uporządkowania listy według kolejno: wynagrodzenia, nazwiska. Eksperymentuj.

> Musisz na początku kodu programu użyć dyrektywy `using System.Linq`.

Kod po kroku 4

Krok 5 - własna generyczna metoda sortująca

10. Przetestuj w `Main()` poprawność powyższych implementacji.

Kod po kroku 1

Krok 2 - implementacja naturalnego porządku w klasie

1. Utwórz listę 5 pracowników:
   - dwóch mających te same nazwiska,
   - dwóch zatrudnionych w tym samym roku i miesiącu,
   - dwóch mających to samo wynagrodzenie.
2. Wypisz tę listę w porządku oryginalnym.

Każdy z algorytmów sortowania (uniwersalny) wymagać będzie porównywania oraz przestawiania elementów. Zatem najpierw napisz generyczną metodę `void SwapElements<T>(this IList<T> list, int i, int j)` zamieniającą elementy listy miejscami (`list[i] ⇆ list[j]`).

> Użycie interfejsu `IList<T>` pozwala na rozszerzenie funkcjonalności Twojego sortowania na dowolne listy (zarówno `List<T>` jak i np. `LinkedList<T>` i inne, implementujące ten interfejs). Użycie słowa kluczowego `this` przy pierwszym argumencie (typu `IList`) powoduje, że tworzysz metodę rozszerzającą.

Sprawdź działanie swoich algorytmów na liście obiektów oraz na liście elementów typu strukturalnego (np. `int`).

Kod po kroku 5

Krok 6 - wyszukiwanie w zbiorze elementów

W klasie `Array` oraz w klasie `List<T>` zdefiniowana jest metoda `BinarySearch` efektywnie wyszukująca zadany element.

Zapoznaj się z dokumentacją tej metody. Jakie warunki musi spełniać zbiór, w którym może ona wyszukiwać wystąpienie elementu. Jaka jest jej reakcja w sytuacji, gdy elementu nie ma zbiorze?

Wykonaj stosowne eksperymenty.