Programowanie pod Windows
Zestaw 1

Język C# - rozgrzewka

2024-02-20

Liczba punktów do zdobycia: **6/6**
Zestaw ważny do: 2024-03-05

1. (**1p**) Napisać program, który wyznacza zbiór wszystkich liczb natualnych 1 a 100000, które są podzielne zarówno przez każdą ze swoich cyfr z osobna jak i przez sumę swoich cyfr.

2. (**1p**) Przygotować rozwiązanie (Solution) które składa się co najmniej z czterech projektów (Project): dwu aplikacji konsolowych i dwu bibliotek.

   W każdej z bibliotek umieścić po jednej klasie z jedną metodą. Dodać referencje do bibliotek z każdej aplikacji konsolowej. Nie dodawać referencji pomiędzy aplikacjami konsolowymi.

   W każdej z aplikacji konsolowych napisać fragment kodu, który wywołuje kod z obu bibliotek.

   Pokazać jak z poziomu Visual Studio uruchomić jedną z aplikacji konsolowych, potem drugą (Set as startup project...) a potem obie naraz.

   Pokazać jak w każdym z tych sposobów uruchomienia kodu można umieszczać pułapki w kodzie i debugować kod.

3. (**1p**) Zdokumentować (przez umieszczenie odpowiednich komentarzy w kodzie) jeden dowolny program z bieżącej sekcji.

   Wygenerować dokumentację w postaci pliku XML podczas kompilacji. Użyć narzędzia SandCastle Help File Builder (`https://github.com/EWSoftware/SHFB`) do zbudowania pomocy w obsługiwanych przez SandCastle stylach (np. Website).

4. (**1p**) Napisać w C# dowolny program demonstrujący użycie klas (metod, pól, propercji, indekserów, delegacji i zdarzeń) oraz podstawowych konstrukcji składniowych (pętle, instrukcje warunkowe, `switch`) i zdekompilować go za pomocą narzędzia **IlSpy** (`https://github.com/icsharpcode/ILSpy`).

   Otrzymany kod skompilować (ilasm), aby otrzymać plik wynikowy. Plik ten następnie zdekompilować na powrót do języka C#.

   Porównać otrzymane w ten sposób pliki z kodem źródłowym. Jak objawiają się i z czego wynikają różnice?

5. (**1p**) Zademonstrować w praktyce następujące kwalifikatory dostępu do składowych (na przykładzie dostępu do pól lub metod)

   - `public`
   - `protected`

- `internal`
- `private`

6. (**1p**) Zaprezentować w praktyce mechanizm przeciążania sygnatur funkcji (funkcje o tej samej nazwie ale różnych sygnaturach).

   Czy typ zwracany z dwóch lub więcej funkcji przeciążonych może być różny czy zawsze musi być taki sam?

   Pokazać jak funkcja przeciążona może wywoływać inną funkcję przeciążoną zamiast dostarczać własnej implementacji. Pokazać jak funkcja może wywołać funkcję z klasy bazowej zamiast dostarczać własnej implementacji. Pokazać jak przeciążać konstruktory klasy.

Wiktor Zychla