

Ćwiczenia 3 - Powtórzenia z programowania obiektowego 2

C1. Wykonaj w ramach jednego projektu następujące polecenia:

- Stwórz klasę `Car`.
- Stwórz w niej nowe prywatne pola `double pojemnoscSilnika`, `string marka`.
- Stwórz obiekt typu `Car`, czy masz dostęp do jego pól? czy możesz zmienić wartości? (nie zmieniaj pól klasy na publiczne).
- Do klasy `Car` dodaj samodzielnie napisany konstruktor bez parametru.
- Przeciąż konstruktor tworząc nowy: `Car(double pojemnoscSilnika, string marka)`. Niech ten parametryczny konstruktor ustawia wartości pól pojemność silnika oraz marka. (użyj inicjalizatora `this`).
- Stwórz nowy obiekt za pomocą konstruktora parametrycznego.
- Za pomocą debuggera sprawdź jakie wartości na prywatnych polach ma obiekt numer 1, a jakie obiekt 2.
- Zmień modyfikatory dostępu konstruktorów na `private`. Czy jest możliwe utworzenie nowego obiektu tej klasy? Co się stało z konstruktorem domyślnym?
- W klasie `Car` stwórz metodę `Create()`, która będzie zwracać obiekt.
- Stwórz obiekt dzięki metodzie `Create()` (pamiętałeś o dodaniu `static` do metody?)
- Utwórz statyczne pole `int iloscKol`. Następnie stwórz statyczny konstruktor ustawiający wartość tego pola na 4. Jak możesz odwołać się do pola? czy możesz go zmodyfikować? co się stanie jak dodasz `readonly`?
- Uruchom program i w debuggerze sprawdź jaką wartość pola `iloscKol` ma wcześniej stworzony obiekt.
- Do klasy `Car` dodaj destruktor `~Car()`, wewnątrz dodaj następujący kod:

```
Console.WriteLine("Zwalniam pamięć");  
Console.ReadKey();
```

- Uruchom program i zwróć uwagę kiedy następuje zwolnienie pamięci.
- Dodaj w klasie `Car` stałe pole `iloscDrzwi` i ustaw mu wartość 4. Jak możesz się do niego odnieść? czy możesz go zmodyfikować?

C2. Stwórz klasę `Matematyka` i dodaj w niej statyczne pole `pi` i nadaj mu wartość 3.14. Następnie w tej klasie dodaj statyczne metody z jednym parametrem pozwalające obliczyć pole i obwód koła. W klasie `Program` i metodzie `Main` przećwiczyć użycie tych metod.

C3. W klasie `Program` poza metodą `Main` dodaj kod:

```
static string OpiszTyp()  
{  
    return "Metoda bez argumentów";  
}  
static string OpiszTyp(int x)  
{  
    return "Liczba całkowita";  
}  
static string OpiszTyp(string x)  
{  
    return "Łańcuch znaków";  
}  
static string OpiszTyp(int x, int y)  
{
```

```
        return "Dwie liczby całkowite";  
    }
```

W metodzie `Main` przetestuj działanie tych metod.

C4. Stwórz klasę `Konto` z kilkoma polami, metodami i konstruktorami (saldo początkowe, końcowe, przelew między kontami, przelew zewnętrzny, wpłata, wypłata itd.) Następnie utwórz co najmniej 3 obiekty i wykonaj kilka metody na nich.

C5. Stwórz klasę `Telefon`, która będzie zawierała parametry do obsługi bilingu (usługa prepaid). Dodatkowo ma zawierać metody, konstruktory (saldo początkowe, końcowe, doładowanie, wysłanie smsa, rozmowa, połączenie z internetem, usługi premium). Potem stwórz kilka obiektów i wykonaj na nich metody. Na ekranie wyświetl „bilinż” dla stworzonych obiektów.

C6. Stwórz nowy projekt, a w nim wykonaj poniższe operacje:

- Stwórz klasę `Osoba` z polami `imie`, `nazwisko`, `rokUrodzenia`, dodaj domyślny konstruktor i parametryczny, oraz dodaj metodę `WypiszInfo()` wyświetlającą wartości obiektu.
- Stwórz klasę potomną `Student` z dodatkowymi polami `rok`, `numerGrupy`, `numerAlbumu`, stwórz konstruktory i metodę `WypiszInfo()`.
- Ustaw modyfikatory dostępu w klasie „rodzica” na `protected`.
- W klasie `Osoba` dodaj metodę `ObliczWiek()`. Następnie stwórz obiekt z klasy `Student` i spróbuj na nim wywołać metodę `ObliczWiek()`.
- W klasie `Osoba` dodaj pole `miejsceZamieszkania` z modyfikatorem `private`. Spróbuj je wywołać dla obiektu z klasy `Student`. Popraw kod aby wszystko działało.
- Stwórz klasę potomną `StudentPierwszegoRoku` dziedziczną z klasy `Student` i `Osoba`. Czy to możliwe? Co w C# możemy jedynie zrobić?
- Stwórz obiekt typu `Osoba` z klasy `Student` (np. `Osoba student2 = new Student()`). Wywołaj dla niego metodę `WypiszInfo()`.
- Stwórz nowy obiekt `student3` typu `Student`. Następnie stwórz obiekt `osoba3` typu `Osoba` i podstaw za niego `student3`. Następnie stwórz obiekt `student4` typu `Student` i podstaw za niego `osoba3`.

```
Student student3 = new Student();  
Osoba osoba3 = student3;  
Student student4 = osoba3;
```

Dla obiektu `osoba3` wywołaj metodę `WypiszInfo()`. Gdzie jest błąd?

C7. W nowym projekcie stwórz klasę `Osoba` z prywatnymi polami `imie` i `nazwisko`. Następnie stwórz klasę potomną `Pacjent` z prywatnym polem `wiek`. W klasie `Pacjent` dodaj konstruktor parametryczny, który ustawi wszystkie pola z klasy bieżącej i „rodzica”.

C8. Zaprojektuj i zaimplementuj podstawy gry RPG. W grze może istnieć kilka typów bohaterów, lecz na początku będą tylko dwa rodzaje – wojownik i łucznik. W przyszłości planowana jest rozbudowa.

- Wiadomo że wszystkie postacie będą opisane imieniem, poziomem życia, oraz posiadać będą operację która pozwoli obliczyć moc ataku.
- Zaimplementuj wojownika i łuczniaka zgodnie ze schematem:

Łucznik: imię, żywotność (w %), zręczność (liczba całkowita), punkty taktyki (liczba całkowita)

Wojownik: imię, żywotność(w %), siła (liczba całkowita), punkty taktyki (liczba całkowita)

- Zadbaj o hermetyzację w klasach.
- Konstruktor domyślny powinien implementować bohaterów zgodnie ze schematem: Łucznik: Imię="GoblinA", żywotność=100%, zręczność=15, PT = 3; Wojownik: Imię="Orka", żywotność=100%, siła=15,PT=1;
- Pozostałe metody:
 - i. zmiana pkt życia (nie mniej niż 0% i nie więcej niż 100%),

- ii. moc ataku (zręczność/siła * PT * żywotność) UWAGA: w przypadku wojownika gdy żywotność spada poniżej 20% wpada w szal i mnożnik żywotności zmienia się na stałe 150%

Pytania: Czy pomyślałeś o klasie rodzica np.bohater? Czy Łucznik i wojownik dziedziczą po bohaterze? Czy ciężko byłoby dodać kolejnych bohaterów? Czy możliwe byłoby stworzenie klasy DrużynaBohaterów i czy miałyby to sens?

C9. Stwórz przykłady klasy obrazujące inne związki między klasami niż dziedziczenie.

C10. Stwórz nowy projekt. Wykonaj w nim następujące polecenia:

- Napisz w nim pustą klasę `Pojazd` oraz dwie dziedziczące po niej klasy `Samochod` i `Rower`.
- Teraz utwórz interfejs `IJazda`, zawierający deklarację metody `Jedz()` typu `void` (z okna `Solution Explorer` wybierz `Add/New Item/Interface/` ew. `ctr+shit+a`).
- Podepnij interfejs do klas `Samochod` i `Rower` i zaimplementuj metodę `Jedz()` w sposób niejawny wyświetlając w konsoli komunikat "Jadę samochodem" lub "Jadę rowerem".
- Stwórz po jednym obiekcie typu `Samochod` i `Rower`. Dla nowo utworzonych obiektów wywołaj metodę `Jedz()` i zobacz co wyświetli się na ekranie.
- Stwórz nowy interfejs `IMuzyka` z metodą `Klakson()` typu `void` i podepnij go do klasy `Rower`. Następnie za pomocą `Console.Beep` stwórz dźwięk dzwonka roweru (jeśli nie masz opcji usłyszenia użyj wyświetl łańcuch "dryń"). Wywołaj metodę na obiekcie typu `Rower`. Czy wielodziedziczenie interfejsów w C# jest możliwe?
- Narysuj diagram UML projektu.
- Skorzystaj z rzutowania w górę i stwórz obiekt `Rower` rzutowany na typ `Pojazd` (`Pojazd rower2 = new Rower();`). Wywołaj na tym obiekcie metodę `Klakson()`. Czy to jest możliwe?

C11. W nowym projekcie wykonaj poniższe czynności:

- Stwórz klasę `Osoba`
- Stwórz interfejsy `ISport` i `IMuzyka`, dodaj w nich deklarację metody `Graj()` zwracającej typ `string`
- Oba interfejsy podepnij pod klasę `Osoba`, dodaj niejawną implementację interfejsu tak by metoda zwracała `string "gra"`.
- w klasie `Program` i metodzie `Main` stwórz trzy obiekty za pomocą poniższego kodu:

```
Osoba osoba1 = new Osoba();
IMuzyka osoba2 = new Osoba();
ISport osoba3 = new Osoba();
```

- Na powyższych obiektach wywołaj metodę `Graj()` i sprawdź na konsoli co zwraca.
- W klasie `Osoba` dodaj jawną implementację interfejsu `IMuzyka` tak, aby metoda `Graj()` zwracała `string "muzyka"` (nie usuwaj niejawną implementacji). Następnie uruchom program i sprawdź co się zmieniło. Czy pamiętasz, aby usunąć modyfikator dostępu?
- W klasie `Osoba` usuń niejawną implementację interfejsu. Następnie dodaj jawną implementację interfejsu `ISport` tak, aby metoda `Graj()` zwracała `string "sport"`. Uruchom program. Co trzeba poprawić? (są dwa możliwe rozwiązania).

C12. W nowym projekcie wykonaj poniższe czynności:

- Stwórz klasę `Owoc` i klasę potomną `Gruszka`.
- Stwórz interfejs `IJedzenie` i dodaj w nim deklarację metody `Jedzenie()` typu `void`.
- Podepnij interfejs pod obie klasy stworzoną wyżej, dodaj w obu klasach implementację niejawną w postaci wyświetlania jakiegoś stringa na konsoli.
- W klasie `Program` i instrukcji `Main` dodaj kod.

```
Gruszka gruszka1 = new Gruszka();
Owoc gruszka2 = new Gruszka();
IJedzenie gruszka3 = new Gruszka();
gruszka1.Jedzenie();
((Owoc)gruszka1).Jedzenie();
```

```

((Gruszka)gruszka1).Jedzenie();
((IJedzenie)gruszka1).Jedzenie();
Console.WriteLine("----");
gruszka2.Jedzenie();
((Owoc)gruszka2).Jedzenie();
((Gruszka)gruszka2).Jedzenie();
((IJedzenie)gruszka2).Jedzenie();
Console.WriteLine("----");
gruszka3.Jedzenie();
((Owoc)gruszka3).Jedzenie();
((Gruszka)gruszka3).Jedzenie();
((IJedzenie)gruszka3).Jedzenie();
Console.ReadKey();

```

Przed uruchomieniem przeanalizuj co powinno się wyświetlić na konsoli.

C13. W nowym projekcie wykonaj czynności:

- Stwórz klasę `WlasnaKlasa`
- Stwórz interfejs `IAaa` i dodaj w nim deklarację metody `Liczba()` zwracającej typ `int`.
- Stwórz interfejs `IBbb` dziedziczący z interfejsu `IAaa` i dodaj w nim deklarację metody `Wyraz()` zwracającej typ `string`.
- Podejpnij interfejs `IBbb` pod klasę `WlasnaKlasa`. Zastanów się, które metody musisz obowiązkowo zaimplementować.
- Narysuj diagram UML projektu.

C14. W nowym projekcie wykonaj czynności:

- W klasie `Car` dodaj pola `year` (rok produkcji) i `brand` (marka) oraz konstruktor z parametrami (`year`, `brand`).
- W klasie `Program` stwórz kolekcję `cars` i dodaj na nią kilka samochodów za pomocą konstruktora np.:

```

List<Car> cars = new List<Car>();
cars.Add(new Car(2010, "Ford"));
cars.Add(new Car(2016, "Fiat"));
cars.Add(new Car(2012, "Skoda"));
cars.Add(new Car(2010, "BMW"));

```

- Do klasy `Car` dodaj interfejs `IComparable<T>` i metodę `CompareTo`. Następnie posortuj elementy na liście i wyświetl je kolejno na konsoli. Na koniec zmień kolejność sortowania.

C15. Stwórz nowy projekt - Zbiór figur:

Zaimplementuj klasy, które pozwolą reprezentować obiekty w dwuwymiarowej przestrzeni.

Punkt – reprezentowany przez dwie zmienne typu rzeczywistego `X` i `Y`

Koło – reprezentowane przez środek koła (punkt), oraz promień.

Kwadrat – reprezentowany przez 4 punkty krawędziowe.

(Wszystkie powyższe klasy posiadają też pole `nazwa` typu `string`.)

Stwórz potrzebne konstruktory.

Dodaj interfejsy z metodami (podziel je pod względem odpowiedzialności): - metoda `LiczObwód` (obwód punktu = 0) - metoda `PobierzDaneZKlawiatury` - metoda `Wyświetl`

Po zakończeniu implementacji stwórz kilka obiektów, policz ich obwód, a następnie je wyświetl. Na koniec stwórz kolekcję, posortuj obiekty.