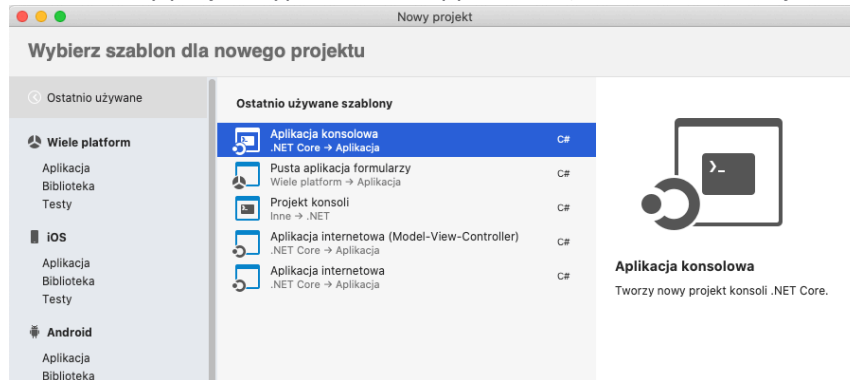
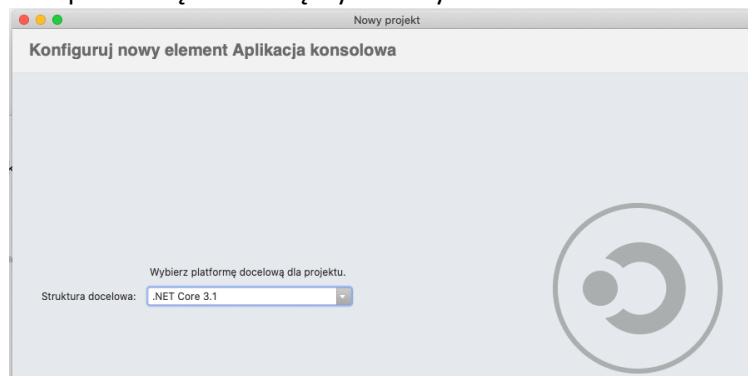


Wprowadzenie do C# – Laboratorium 1

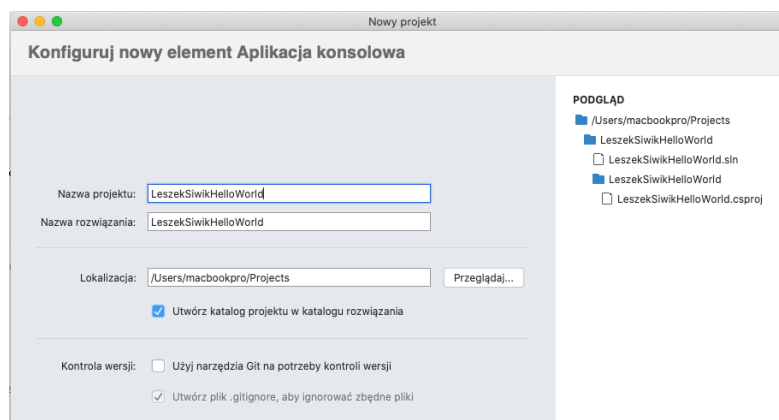
- I. HelloWorld:
 - a. Uruchom Visual Studio
 - b. Stwórz nowy projekt typu Console Application (File -> New -> Project....).



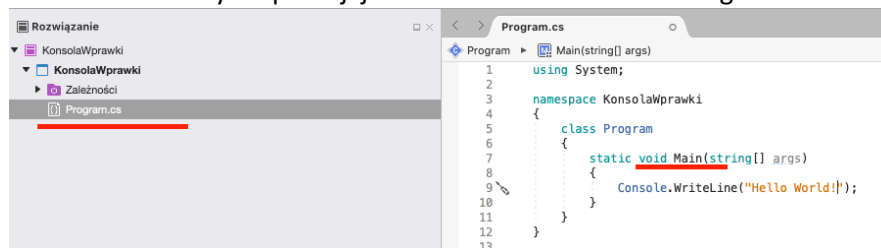
- c.
- d. Jako platformę docelową wybieramy .Net Core 3.1



- e.
- f. Nazwij projekt: LeszekSiwikHelloWorld.



- g.
- h. Punktem startowym aplikacji jest metoda Main w klasie Program



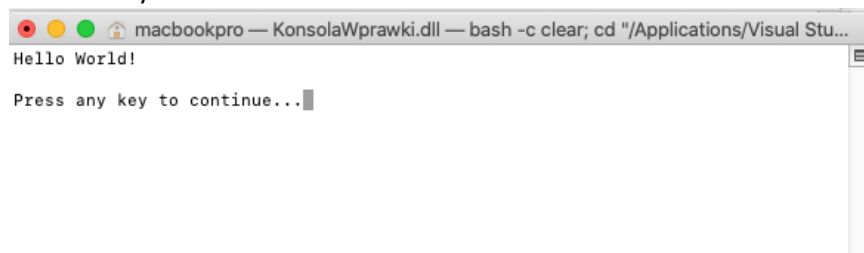
- i.

- j. W wygenerowanym szkieletcie jedyne co się w tej funkcji dzieje to wypisanie na konsole/ekran napisu HelloWorld. Jak widać aby coś wypisać na konsoli używamy np. metody WriteLine klasy Console).

```
1  using System;
2
3  namespace KonsolaWprawki
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              Console.WriteLine("Hello World!");
10         }
11     }
12 }
13
```

- k.
- l. Uruchom aplikację (strzałka w pasku menu na górze VisualStudio albo skróty klawiaturowe dla Maca command+F10, dla Windowsa Ctrl+F9 – więcej jeśli chodzi o skróty:
- i. dla windowsa: <https://docs.microsoft.com/en-us/visualstudio/ide/default-keyboard-shortcuts-in-visual-studio?view=vs-2019>
 - ii. Dla Maca: <https://docs.microsoft.com/en-us/visualstudio/mac/keyboard-shortcuts?view=vsmac-2019>

- m. Oczekiwany efekt:



The screenshot shows a terminal window titled 'macbookpro — KonsolaWprawki.dll — bash -c clear; cd "/Applications/Visual Stu...'. The output of the program is 'Hello World!' followed by a prompt 'Press any key to continue...' with a cursor.

- n.
- o. Jeżeli masz taką sytuację że konsola otwiera się i (po wypisaniu komunikatu) od razu się zamyka możesz dodać na koniec metody Main np. wywołanie metody ReadKey klasy Console (czyli wczytywanie znaku z klawiatury) co spowoduje że program będzie czekał aż naciśniesz cokolwiek na klawiaturze:

```
1  using System;
2
3  namespace KonsolaWprawki
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              Console.WriteLine("Hello World!");
10             Console.ReadKey();
11         }
12     }
13 }
14
```

- p.
- q. Generalnie jak widać po powyższym fragmencie kodu – struktura programu i składnia C# jest bardzo zbliżona (przynajmniej na tym najbardziej ogólnym poziomie) np. do Javy tylko zamiast importów mamy usingi (ale znaczenie takie samo) zamiast nazwy pakietu mamy namespace.
- r. Zmienne deklarujemy w sposób analogiczny do tego znanego z Javy: typ nazwa zmiennej i dalej ew. przypisanie. Zadeklarujemy więc zmienną stringową helloMessage i przypiszmy od razu do niej wartość „Cała naprzód”, a następnie tam gdzie mamy

wywołanie metody Console.WriteLine jako argument do wypisania podajmy to stworzona zmienna zamiast wpisanego tam na sztywno napisu HelloWorld. Czyli:

```
Program ▶ Main(string[] args)
1  using System;
2
3  namespace KonsolaWprawki
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              string helloMessage = "Cala naprzod";
10             Console.WriteLine(helloMessage);
11             Console.ReadKey();
12         }
13     }
14 }
15
```

s.

t. Listę typów podstawowych wraz z przykładami znajdziesz np. tutaj:

https://www.w3schools.com/cs/cs_data_types.asp

u. W analogii do metody Console.WriteLine mamy metodę Console.ReadLine do zczytania danych wprowadzonych przez użytkownika. Metoda ta zwraca stringa. Zatem – zadeklaruj zmienną stringową userValue – przypisz do niej wartość wprowadzoną przez użytkownika a następnie wypisz ją na ekran.

```
1  using System;
2
3  namespace KonsolaWprawki
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              Console.WriteLine("Wpisz cos...");
10             string userValue = Console.ReadLine();
11             Console.WriteLine("Wartosc wprowadzona przez uzytkownika to: " + userValue);
12             Console.ReadKey();
13         }
14     }
15 }
16
```

v.

w. Jak widać powyżej jednym ze sposobów na wypisanie zmiennej wraz z jakimś komunikatem albo inną zmienną jest konkatencja wartości (operator + na łańcuchach znakowych) innym jest posługiwanie się operatorem {} na wskazanie miejsca gdzie ma być wstawiona wartość jakiejś zmiennej jak np. poniżej

```
static void Main(string[] args)
{
    Console.WriteLine("Wpisz cos...");
    string userValue = Console.ReadLine();
    string helloMessage = "Cala naprzod";
    Console.WriteLine("Wartosc stringa wpisana na stale to {0}, a wartosc " +
        "wprowadzona przez uzytkownika to: {1} ", helloMessage, userValue);
    Console.ReadKey();
}
```

x.

y. Ew możemy posłużyć się składnią \$"{}" jak poniżej:

```
7  static void Main(string[] args)
8  {
9      Console.WriteLine("Wpisz cos...");
10     string userValue = Console.ReadLine();
11     string helloMessage = "Cala naprzod";
12     Console.WriteLine($"Pierwszy string: {helloMessage}. I drugi string: {userValue}");
13     Console.ReadKey();
14 }
```

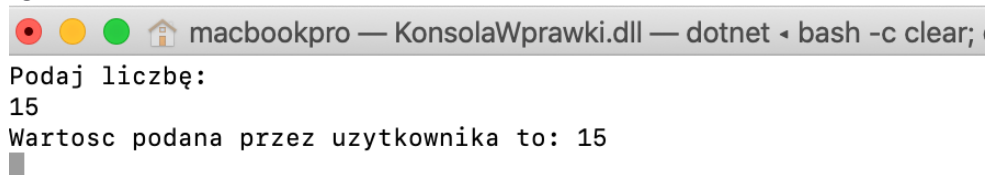
z.

- aa. Funkcja `Console.ReadLine` zwraca stringa. Jeśli chcesz zczytać od użytkownika liczbę musisz zwracany string przekonwertować do liczby np. w taki sposób:

```
static void Main(string[] args)
{
    Console.WriteLine("Podaj liczbę:");
    int userValue = int.Parse(Console.ReadLine());
    Console.WriteLine($"Wartosc podana przez uzytkownika to: {userValue}");
    Console.ReadKey();
}
```

bb.

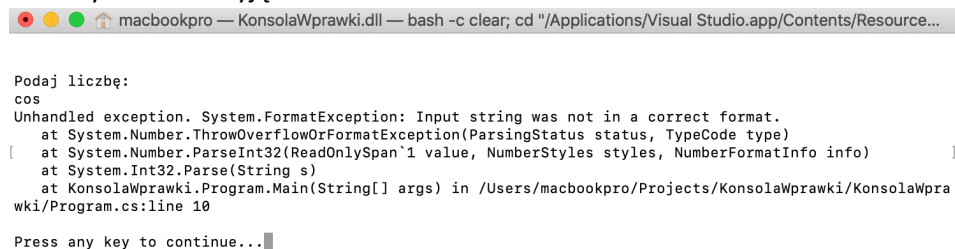
- cc. Aktualnie jeśli użytkownik poda „prawidłową” wartość (liczbę) to kod zadziała zgodnie z oczekiwaniami:



```
macbookpro — KonsolaWprawki.dll — dotnet -c clear;
Podaj liczbę:
15
Wartosc podana przez uzytkownika to: 15
```

dd.

- ee. Natomiast jeśli użytkownik poda wartość której nie da się skonwertować na liczbę to rzucony zostanie wyjątek:



```
macbookpro — KonsolaWprawki.dll — bash -c clear; cd "/Applications/Visual Studio.app/Contents/Resource...
Podaj liczbę:
cos
Unhandled exception. System.FormatException: Input string was not in a correct format.
   at System.Number.ThrowOverflowOrFormatException(ParsingStatus status, TypeCode type)
   at System.Number.ParseInt32(ReadOnlySpan`1 value, NumberStyles styles, NumberFormatInfo info)
   at System.Int32.Parse(String s)
   at KonsolaWprawki.Program.Main(String[] args) in /Users/macbookpro/Projects/KonsolaWprawki/KonsolaWprawki/Program.cs:line 10
Press any key to continue...
```

ff.

- gg. Żeby ta linijka odpowiedzialna za przekonwertowanie wpisanej wartości do liczby nie „wywracała” aplikacji można ten fragment opakować blokiem `try-catch` (napisz `try` i pamiń dwa razy `tab`) i obsłużyć wyjątek choćby wyświetlając użytkownikowi jakiś komunikat jak poniżej:

```
7      static void Main(string[] args)
8      {
9          Console.WriteLine("Podaj liczbę:");
10         try
11         {
12             int userValue = int.Parse(Console.ReadLine());
13             Console.WriteLine($"Wartosc podana przez uzytkownika to: {userValue}");
14         }
15         catch (Exception ex)
16         {
17             Console.WriteLine("Podana wartosc jest nieprawidlowa");
18         }
19         Console.ReadKey();
20     }
21 }
22
```

hh.

- ii. W ramach wprawek – zczytaj od użytkownika dwie liczby i wydrukuj na konsoli ich sumę. Sumowanie powinno umożliwiać dodawanie liczb „z przecinkiem”. Przykładowe rozwiązanie podaje na końcu instrukcji jako Rozwiązanie 1. Zanim zerkniesz postaraj się zrealizować ćwiczenie samodzielnie.

II. Instrukcje warunkowe

- a. Dostępną mamy klasykę jeśli chodzi o instrukcje warunkowe (`if` + dwa razy `tabulator` oraz `switch` + dwa razy `tabulator`)

if (warunek) { Instrukcja1 ... instrukcja n	Jeżeli warunek jest spełniony, wykonywany jest blok kodu poniżej if, w przeciwnym wypadku poniżej else
---	--

<pre> } else { Instrukcja1, ... instrukcja n } </pre>	<pre> int i = 5; if (i > 5) Console.WriteLine(„Wartosc i jest większa od 5”); else Console.WriteLine(„Wartosc i nie jest większa od 5”); </pre>
<pre> switch (wyrażenie) { case wyrażenie_stałe1: instrukcja case wyrażenie_stałe2: instrukcja ... default: instrukcja } </pre>	<p>Służy do obsługi wielowarstwowych sytuacji decyzyjnych (może zastępować wielokrotne wykorzystanie if...else...else...)</p> <pre> int a1 = Int32.Parse(Console.ReadLine()); switch (a1) { case 1: Console.WriteLine(„Wpisales 1”); break; case 2: Console.WriteLine(„Wpisales 2”); break; case 3: Console.WriteLine(„Wpisales 3”); break; default: Console.WriteLine(„Nie wpisałeś ani 1, 2, 3”); break; } </pre>

- b. Zmodyfikuj nasz poprzedni program w ten sposób, że wczytasz z konsoli dwie liczby i znak (+/-/*). W zależności od wprowadzonego znaku proszę wykonać operacje: dodawania/odejmowania/mnożenia/dzielenia liczb. Wprowadzenie innego niż wymienione znaki powinny poinformować użytkownika o błędzie. Użyj instrukcji switch. Moją wersję rozwiązania umieściłem na końcu instrukcji jako Rozwiązanie 2. Tradycyjnie, zanim zerkniesz/zweryfikujesz spróbuj rozwiązać samodzielnie.
- c. Zanim przejdziesz do dalszych części:
 - i. Wydziel kod tego ćwiczenia do osobnej metody (zaznacz kod do wydzielenia, Right Click -> Refactor -> Extract Method.....)
 - ii. Zwin kod metody żeby nie przeszkadzał (zaznacz kod metody i następnie Ctrl+MH)
 - iii. Zakomentuj w mainie wywołanie wyekstrahowanej metody (Ctrl+KC)
 - iv. Postępuj tak z wszystkim kolejnymi ćwiczeniami, dzięki temu będziesz miał w mainie na bieżąco tylko to co potrzebujesz a jednocześnie to co zrobisz – nie zginie 😊

III. Pętle

- a. Mamy do dyspozycji klasykę jeśli chodzi o pętle:

<pre> foreach (element w kolekcji){ } </pre>	<p>Iterowanie po wszystkich elementach kolekcji</p> <pre> foreach (var item in collection) { } </pre>
--	---

for (war_pocz, war_konc, inkrementacja/dekrementacja) { Instrukcja ... }	Pętla for definiuje ilość wykonywanych powtórzeń <code>int a1 = 1;</code> <code>for (int i = 0; i < 10; i++)</code> { <code>a1 += i;</code> } <code>Console.WriteLine(a1); // Wypisze 46</code>
while(warunek) { Instrukcja ... }	Pętla while definiuje ilość wykonywanych zapytań, sterowanych przez warunek. <code>int a = 10;</code> <code>while (a > 0)</code> { <code>Console.WriteLine(a);</code> <code>a--;</code> }
Do{ } while(warunek)	

- Zmodyfikuj działanie naszego kalkulatora, w taki sposób aby działał tak długo aż użytkownik na pytanie czy kontynuować nie odpowie „n”. Swoją wersję rozwiązania podaje jako Rozwiązanie 3 na końcu instrukcji. Tradycyjnie, nim zerkniesz/zweryfikujesz – rozwiąż samodzielnie.
- *Rozszerz kalkulator o możliwość wyliczania silni z wprowadzonej liczby. Zdebuguj liczenie silni dla n=5
- *Zaimplementuj metodę obliczania silni z podanej liczby (podanej przez użytkownika) metodą rekurencyjną

IV. Tablice

- Tablica – najbardziej klasyczna / podstawowa kolekcja elementów tego samego typu
- Przykładowe deklaracje:

```
int[] myInts = new int[7];
string[] booksTitle = new string[100];
string[] stringArray = new string[]
{ "one", "two", "three" };
```

- Napisz program, w którym wczytasz rozmiar tablicy, oraz maksymalną wartość jaką może być przechowywana w komórkach tablicy
- Zadeklaruj tablice o wczytanym rozmiarze i wypełnij ją losowymi wartościami intowymi nieprzekraczającymi zdefiniowanej granicy (metoda Next() na obiekcie klasy Random) (Użyj na tym etapie pętli for. skrót: for + dwa razy tabulator)
- Wydrukuj na konsoli zawartość tablicy (użyj na tym etapie konstrukcji foreach, skrót: foreach + dwa razy tabulator).
- Uruchom i sprawdź działanie programu. Przykładowe działanie:

```
macbookpro — KonsolaWprawki.dll — dotnet • bash -c clear; cd
Podaj rozmiar tablicy:
10
Podaj maksymalna wartosc dozwolona w komorkach tablicy
20

19 13 14 3 10 4 11 2 9 9 █
```

Moją wersję implementacji pokazuje na końcu instrukcji jako rozwiązanie 4.
Tradycyjnie, nim zerkniesz/zweryfikujesz – spróbuj rozwiązać samodzielnie.

V. Sortowanie

- Rozszerz poprzedni program w ten sposób, że po stworzeniu i wypełnieniu losowymi wartościami tablicy posortujesz ją metodą bąbelkową
- Sortowanie bąbelkowe polega na porównywaniu dwóch kolejnych elementów tablicy i zamianie ich kolejności, jeżeli zaburza ona porządek sortowania. Sortowanie kończy się, gdy podczas kolejnego przejścia nie dokonano żadnej zmiany.
- Algorytm sortowania zaimplementuj w osobnej metodzie.
- Dodatkowo przygotuj/wydziel metody-helper odpowiedzialne za:
 - inicjalizację tablicy
 - wypisanie zawartości tablicy na ekran
- Uodpornij program na błędne dane wejściowe, a metodę sortującą na najbardziej typowe błędy wejścia.
- Uruchom i sprawdź działanie programu. Przykładowe działanie:

```
macbookpro — KonsolaWprawki.dll — dotnet • bash -c clear; cd
Podaj rozmiar tablicy:
10
Podaj maksymalna wartosc dozwolona w komorkach tablicy
20
Tablica z nieposortowanymi wartosciami:
9 17 0 19 12 17 19 16 15 13
Tablica z posortowanymi wartosciami:
0 9 12 13 15 16 17 17 19 19
█
```

- Tradycyjnie, swoją wersję rozwiązania prezentuje na końcu instrukcji jako Rozwiązanie 5. Nim zerkniesz/zweryfikujesz – spróbuj rozwiązać samodzielnie.
- Zmodyfikuj poprzednie rozwiązanie w taki sposób że generowana tablica nie jest zwracana z funkcji odpowiedzialnej za wygenerowanie tablicy, ale jest tam przekazywana referencja referencja na tablice (ref int[] arrayRef)

VI. Kolekcje

- Język C# obsługuje praktycznie wszystkie spotykane w informatyce struktury danych, tj. tablice, stosy, kolejki, listy, wykazy asocjacyjne. Klasy kolekcji zostały zdefiniowane w obszarze nazw: System.Collections oraz System.Collections.Generic. Jedną z popularniejszych kolekcji są listy. Przykładowe deklaracje i definicje:

```
ArrayList myArrayList = new ArrayList();
List<int> myList = new List<int>();
myList.Add(12);
myList.Remove(12);
```

- b. Napisz program, który:
 - i. Utworzy dwie listy
 - ii. W pętli for (od 1 do 10) zapyta użytkownika jakie chce wprowadzić liczby do pierwszej listy i doda tam te liczby
 - iii. Usunie z pierwszej listy wszystkie liczby mniejsze od 2, a wszystkie pozostałe skopiuje do listy 2
 - iv. Wypisze listy

VII. Klasy i obiekty

- a. Do definicji klasy – słówko kluczowe class dalej nazwa klasy i w nawiasach {} definicja klasy.

```
class Person
{
    public string Firstname;
    public string Lastname;
}
```

- b. W C# definiując klasy, zamiast definiować prywatne pole z publicznymi metodami typu get, i set posługujemy się „properties” czyli publicznymi polami z ustawionymi od razu „getterem” i „setterem” które ‘przykrywają’ prywatne pole klasy

```
class Person
{
    public String Firstname { get; set; }
    public String Lastname { get; set; }
}
```

- c. Żeby wygenerować property piszemy słówko prop i naciskamy dwa razy tabulator. Zmieniamy typ i nazwę i mamy property gotowe 😊.
- d. Korzystając ze zdefiniowanych property możemy zainstancjonować obiekt klasy Person w ten sposób:

```
static void Main(string[] args)
{
    //SimpleCalulation();
    //ArrayExercise();
    //ArrayListExercise();
    Person person = new Person { FirstName = "Leszek", LastName = "Leszkowski" };
    Console.WriteLine(person);

    Console.ReadKey();
}
```

- e.

- f. Zdefiniuj klasę Animal z polami: Name, Age, Species oraz metodą Move (informująca ze zwierze się porusza)
- g. Stwórz przykładowego zwierzaka, wydrukuj na konsoli jego cechy i „każ mu się przejść”
- h. Uruchom i sprawdź działanie programu. Przykładowe działanie:

- i. Moją wersję implementacji preentuje jako Rozwiązanie 7. Tradycyjnie – nim zerkniesz spróbuj rozwiązać samodzielnie.
- j. Operator dziedziczenia w C# to „:”
- k. Zdefiniuj klasy Fish oraz Dog – obie dziedziczące po klasie Animal. Zdefiniuj w nich metode Move tak aby ryby informowały że płyną a psy że biegna
- l. Stwórz przykładową rybe i przykładowego psiaka – oba obiekty zrob typu takiego jakiej są klasy. Wydrukuj na konsoli ich cechy i „kaz im się poruszać”.

```
private static void ClassExercise()
{
    Animal animal = new Animal { Name = "Puffy", Age = 5, Species = "Tarantula" };
    Console.WriteLine(animal);
    animal.Move();

    Fish fish = new Fish { Name = "Rybus", Age = 3, Species = "Karpik" };
    Console.WriteLine(fish);
    fish.Move();

    Dog dog = new Dog { Name = "Goffy", Age = 4, Species = "York" };
    Console.WriteLine(dog);
    dog.Move();
}
```

- m.
- n. Uruchom i sprawdź działanie programu. Przykładowe działanie:

- o.
- p. Moja wersja implementacji – na końcu instrukcji jako Rozwiązanie 8.
- q. Stwórz teraz obiekt klasy ryba ale niech on będzie typu Animal, czyli wywołanie:

```
Animal animalFish = new Fish { Name = "AnimalFishRybus", Age = 2, Species = "NiAnimalNiFish" };
Console.WriteLine(animalFish);
animalFish.Move();
```

- r.
- s. Uruchom i przetestuj. Wynik będzie podobny do poniższego:

- t.
- u. Zauważ, że tak zainstancjonowana Ryba się „porusza” a nie „płyne” – czyli wywołana została metoda Move z klasy Animal. Jeżeli chcemy aby metoda Move z podklasy (czyli

np. w naszym przypadku z klasy Fish) „przykrywała” metodę z nadklasy to metodę w nadklasie oznaczamy ozdobnikiem „virtual” a metodę w podklasie ozdobnikiem override jak poniżej:

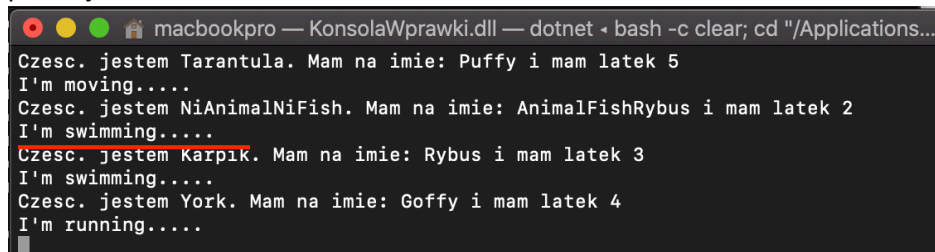
```
public class Animal
{
    public string Name { get; set; }
    public int Age { get; set; }
    public string Species { get; set; }

    public virtual void Move()
    {
        Console.WriteLine("I\'m moving.....");
    }
}

public class Fish : Animal
{
    public override void Move()
    {
        Console.WriteLine("I\'m swimming.....");
    }
}
```

w.

x. Wprowadź te zmiany i uruchom program ponownie. Aktualnie efekt powinien być jak poniżej:



The screenshot shows a terminal window titled 'macbookpro — KonsolaWprawki.dll — dotnet < bash -c clear; cd "/Applications...'. The output of the program is as follows:

```
Czesc. jestem Tarantula. Mam na imie: Puffy i mam latek 5
I'm moving.....
Czesc. jestem NiAnimalNiFish. Mam na imie: AnimalFishRybus i mam latek 2
I'm swimming.....
Czesc. jestem Karpik. Mam na imie: Rybus i mam latek 3
I'm swimming.....
Czesc. jestem York. Mam na imie: Goffy i mam latek 4
I'm running.....
```

y.

z. Czyli aktualnie nasz NiAnimalNiFish już płynie

VIII. Jeśli się nudzisz:

a. Dodaj do programu metode sortowania przez wstawianie

i. **Sortowanie przez wstawianie** (ang. *Insert Sort*, *Insertion Sort*) - jeden z najprostszych [algorytmów sortowania](#), którego zasada działania odzwierciedla sposób w jaki ludzie ustawiają karty - kolejne elementy wejściowe są ustawiane na odpowiednie miejsca docelowe.

ii. Schemat działania:

1. Utwórz zbiór elementów posortowanych i przenieś do niego dowolny element ze zbioru nieposortowanego.
2. Weź dowolny element ze zbioru nieposortowanego.
3. Wyciągnięty element porównuj z kolejnymi elementami zbioru posortowanego póki nie napotkasz elementu równego lub elementu większego (jeśli chcemy otrzymać ciąg niemalejący) lub nie znajdziemy się na początku/końcu zbioru uporządkowanego.
4. Wyciągnięty element wstaw w miejsce gdzie skończyłeś porównywać.
5. Jeśli zbiór elementów nieuporządkowanych jest niepusty wróć do punkt 2.

- b. Dodaj do programu metode sortowania przez wybieranie
- c. **Sortowanie przez wybieranie** - jedna z prostszych metod [sortowania](#) o [złożoności](#) $O(n^2)$. Polega na wyszukaniu elementu mającego się znaleźć na zadanej pozycji i zamianie miejscami z tym, który jest tam obecnie. Operacja jest wykonywana dla wszystkich indeksów sortowanej tablicy.
- d. Algorytm przedstawia się następująco:
 - i. wyszukaj minimalną wartość z tablicy spośród elementów od $i+1$ do końca tablicy
 - ii. zamień wartość minimalną, z elementem na pozycji i
 - iii. Gdy zamiast wartości minimalnej wybierana będzie maksymalna, wówczas tablica będzie posortowana od największego do najmniejszego elementu.
- e. Zmodyfikuj program tak aby użytkownikowi zaprezentowane zostały dostępne metody sortowania i aby użytkownik mógł wybrać metode z ktorej chce skorzystać
- f. Rozszerz poprzednio stworzone „menu” i parametry programu o opcje przeprowadzenia sortowania wszystkimi dostępnymi metodami wraz z pomiarem czasu sortowania dla poszczególnych metod. Wyjściem z programu powinien być (dodatkowo) wynik pomiaru czasu sortowania dla poszczególnych metod sortowania.
- g. Zadanie z * - dodaj implementacje algorytmu quicksort
- h. Zadania z * - dodaj implementacje wyszukiwania binarnego w posortowanej tablicy
Zamiast przeglądać całą tablicę – podziel tablicę na połowe, jeśli szukana wartość jest większa od szukanego klucza zawęż poszukiwania do „górnej” połowy, w przeciwnym przypadku do „dolnej połowy”. Kontynuuj działanie tak długo, aż znajdziesz poszukiwaną wartość albo wyczerpane zostaną możliwości poszukiwania.

IX. Rozwiązania ćwiczeń:

a. Rozwiązanie 1:

```
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              try
10             {
11                 Console.WriteLine("Podaj pierwsza liczbę:");
12                 double firstNumber = double.Parse(Console.ReadLine());
13                 Console.WriteLine("Podaj druga liczbę:");
14                 double secondNumber = double.Parse(Console.ReadLine());
15                 Console.WriteLine($"{firstNumber}+{secondNumber} = {firstNumber+secondNumber}");
16             }
17             catch (Exception ex)
18             {
19                 Console.WriteLine("Podana wartosc jest nieprawidlowa");
20             }
21             Console.ReadKey();
22         }
23     }
24 }
```

i.

macbookpro — KonsolaWprawki.dll — dotnet

Podaj pierwsza liczbę:

[6,7

Podaj druga liczbę:

[4,5

6,7+4,5 = 11,2

ii.

b. Rozwiązanie 2:

```
7      static void Main(string[] args)
8      {
9          double firstNumber = 0, secondNumber = 0, result = double.NaN;
10         string operation = "";
11         try
12         {
13             Console.WriteLine("Podaj pierwsza liczbę:");
14             firstNumber = double.Parse(Console.ReadLine());
15             Console.WriteLine("Podaj druga liczbę:");
16             secondNumber = double.Parse(Console.ReadLine());
17             Console.WriteLine("Podaj dzialanie do wykonania: [+,*/]");
18             operation = Console.ReadLine();
19             switch (operation)
20             {
21                 case "+":
22                     result = firstNumber + secondNumber;
23                     break;
24                 case "-":
25                     result = firstNumber - secondNumber;
26                     break;
27                 case "*":
28                     result = firstNumber * secondNumber;
29                     break;
30                 case "/":
31                     if (secondNumber != 0)
32                     {
33                         result = firstNumber / secondNumber;
34                     }
35                     break;
36             }
37         }
38         catch (Exception ex)
39         {
40             printErrorMessage();
41         }
42         if (!double.IsNaN(result))
43         {
44             Console.WriteLine($"{firstNumber}{operation}{secondNumber} = {result}");
45         }
46         else {
47             printErrorMessage();
48         }
49     }
```

i.

c. Rozwiązanie 3:

i.
d. Rozwiązanie 4

```
class Program
{
    static void Main(string[] args)
    {
        SimpleCalculation();
        Console.ReadKey();
    }

    private static void SimpleCalculation()
    {
        double firstNumber = 0, secondNumber = 0, result = double.NaN;
        string operation = "";
        string continuation;
        do
        {
            try
            {
                if (!double.IsNaN(result))
                {
                    Console.WriteLine("Czy kontynuowac [y/n]");
                    continuation = Console.ReadLine();
                } while (!continuation.Equals("n"));
            }
            Environment.Exit(0);
        }

        private static void printErrorMessage()
        {
            Console.WriteLine("Podane liczby i/lub operator sa nieprawidlowe");
        }
    }

    private static void generateRandomValueArray()
    {
        Console.WriteLine("Podaj rozmiar tablicy:");
        int arraySize, arrayMaxValue;
        int.TryParse(Console.ReadLine(), out arraySize);
        Console.WriteLine("Podaj maksymalna wartosc dozwolona w komorkach tablicy");
        int.TryParse(Console.ReadLine(), out arrayMaxValue);

        if (arraySize > 0)
        {
            int[] myArray = new int[arraySize];
            Random random = new Random();

            for (int i = 0; i < arraySize; i++)
            {
                myArray[i] = random.Next(arrayMaxValue);
            }

            printArrayValues(myArray);
        }
    }

    private static void printArrayValues(int[] myArray)
    {
        Console.WriteLine();
        foreach (var item in myArray)
        {
            Console.Write(item + " |");
        }
    }
}
```

e. Rozwiązanie 5

```
static void Main(string[] args)
{
    //SimpleCalulation();

    int[] myArray = GenerateRandomValueArray();

    Console.WriteLine("Tablica nieposortowana:");
    printArrayValues(myArray);

    arrayBubbleSort(myArray);

    Console.WriteLine("Tablica posortowana:");
    printArrayValues(myArray);

    Console.ReadKey();
}

i. private static int[] GenerateRandomValueArray()
{
    Console.WriteLine("Podaj rozmiar tablicy:");
    int arraySize, arrayMaxValue;
    int.TryParse(Console.ReadLine(), out arraySize);
    Console.WriteLine("Podaj maksymalna wartosc dozwolona w komorkach tablicy");
    int.TryParse(Console.ReadLine(), out arrayMaxValue);
    int[] myArray = null;
    if (arraySize > 0)
    {
        myArray = new int[arraySize];
        Random random = new Random();

        for (int i = 0; i < arraySize; i++)
        {
            myArray[i] = random.Next(arrayMaxValue);
        }
    }
    return myArray;
}

ii. private static void arrayBubbleSort(int[] myArray)
{
    bool arraySorted;
    do
    {
        arraySorted = true;
        for (int i = 0; i < myArray.Length - 1; i++)
        {
            if (myArray[i] > myArray[i + 1])
            {
                int tmp = myArray[i];
                myArray[i] = myArray[i + 1];
                myArray[i + 1] = tmp;
                arraySorted = false;
            }
        }
    } while (!arraySorted);
}

iii.
```

```

private static void printArrayValues(int[] myArray)
{
    Console.WriteLine();
    foreach (var item in myArray)
    {
        Console.Write(item + " ");
    }
    Console.WriteLine();
}

```

f.

g. Rozwiązanie 6

```

private static void GenerateRandomList(ArrayList firstList)
{
    Random random = new Random();
    Console.WriteLine("Podaj liczbę elementów listy");
    int listSize = int.Parse(Console.ReadLine());

    Console.WriteLine("Podaj górne ograniczenie wartości elementów listy");
    int listMaxValue = int.Parse(Console.ReadLine());
    for (int i = 0; i < listSize; i++)
    {
        firstList.Add(random.Next(listMaxValue));
    }
}

```

i.

```

private static void ArrayListExercise()
{
    ArrayList secondList = new ArrayList();

    ArrayList firstList = new ArrayList();
    GenerateRandomList(firstList);
    Console.WriteLine("Pierwsza lista przed przekształceniami");
    printListValues(firstList);
    Console.WriteLine("Druga lista przed przekształceniami");
    printListValues(secondList);

    transformList(secondList, firstList);
    Console.WriteLine("Pierwsza lista po przekształceniach");
    printListValues(firstList);
    Console.WriteLine("Druga lista po przekształceniach");
    printListValues(secondList);
}

```

```

private static void printListValues(ArrayList arrayList)
{
    Console.WriteLine();
    foreach (var item in arrayList)
    {
        Console.Write($"{item} ");
    }
    Console.WriteLine();
}

```

ii.

iii.

```
private static void transformList(ArrayList secondList, ArrayList firstList)
{
    for (int i = firstList.Count - 1; i >= 0; i--)
    {
        if (Convert.ToInt16(firstList[i]) < 10)
        {
            firstList.RemoveAt(i);
        }
        else
        {
            secondList.Add(firstList[i]);
        }
    }

    secondList.Reverse();
}
```

h. Rozwiązanie 7:

```
public class Animal
{
    public string Name { get; set; }
    public int Age { get; set; }
    public string Species { get; set; }

    public void Move()
    {
        Console.WriteLine("I\'m moving.....");
    }

    public override string ToString()
    {
        return $"Czesc. jestem {Species}. Mam na imie: {Name} i mam latek {Age}";
    }
}

Animal animal = new Animal { Name = "Puffy", Age = 5, Species = "Tarantula" };
Console.WriteLine(animal);
animal.Move();
```

i.

ii.

i. Rozwiązanie 8

```
public class Animal
{
    public string Name { get; set; }
    public int Age { get; set; }
    public string Species { get; set; }

    public void Move()
    {
        Console.WriteLine("I\'m moving.....");
    }

    public override string ToString()
    {
        return $"Czesc. jestem {Species}. Mam na imie: {Name} i mam latek {Age}";
    }
}

public class Fish : Animal
{
    public new void Move()
    {
        Console.WriteLine("I\'m swimming.....");
    }
}

public class Dog : Animal
{
    public new void Move()
    {
        Console.WriteLine("I\'m running.....");
    }
}
```

iii.

iv.

```
private static void ClassExercise()
{
    Animal animal = new Animal { Name = "Puffy", Age = 5, Species = "Tarantula" };
    Console.WriteLine(animal);
    animal.Move();

    Fish fish = new Fish { Name = "Rybus", Age = 3, Species = "Karpik" };
    Console.WriteLine(fish);
    fish.Move();

    Dog dog = new Dog { Name = "Goffy", Age = 4, Species = "York" };
    Console.WriteLine(dog);
    dog.Move();
}
```