

TypeScript

tutorial version 1.8

TypeScript

The image shows the TypeScript website homepage. At the top left is the TypeScript logo, which consists of the letters 'TS' in a white square followed by the word 'TypeScript' in white. To the right of the logo is a navigation menu with the following items: 'Download', 'Docs', 'Handbook', 'Community', 'Playground', and 'Tools'. The main content area has a dark blue background. The headline reads 'TypeScript is JavaScript with syntax for types.' Below this, a paragraph states: 'TypeScript is a strongly typed programming language that builds on JavaScript, giving you better tooling at any scale.' At the bottom of the main content area, there is a white rectangular button with the text 'Try TypeScript Now' and 'Online or via npm' on two lines. To the right of the text is a vertical line, and to the right of the line are three small white squares arranged horizontally.

TS TypeScript Download Docs Handbook Community Playground Tools

TypeScript is JavaScript with syntax for types.

TypeScript is a strongly typed programming language that builds on JavaScript, giving you better tooling at any scale.

Try TypeScript Now
Online or via npm

<https://www.typescriptlang.org/>

TypeScript



Tutorials ▾

Exercises ▾

Certificates ▾

Services ▾

Search...



< W3.CSS C C++ C# BOOTSTRAP REACT MYSQL JQUERY EXCEL

TypeScript tutorial

TS HOME

TS Introduction

TS Get Started

TS Simple Types

TS Special Types

TS Arrays

TS Tuples

TS Object Types

TS Enums

TS Aliases & Interfaces

TS Union Types

TS Functions

TS Casting

TS Classes

TS Basic Generics

TS Utility Types

TypeScript Tutorial

< Home

TypeScript is JavaScript with added syntax for types.

Start learning TypeScript now »

<https://www.w3schools.com/typescript/>

Node.js



[HOME](#) | [ABOUT](#) | [DOWNLOADS](#) | [DOCS](#) | [GET INVOLVED](#) | [SECURITY](#) | [CERTIFICATION](#) | [NEWS](#)

Node.js® is a JavaScript runtime built on [Chrome's V8 JavaScript engine](#).

Download for Windows (x64)

16.14.0 LTS

Recommended For Most Users

17.7.1 Current

Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#) [Other Downloads](#) | [Changelog](#) | [API Docs](#)

Or have a look at the [Long Term Support \(LTS\) schedule](#)

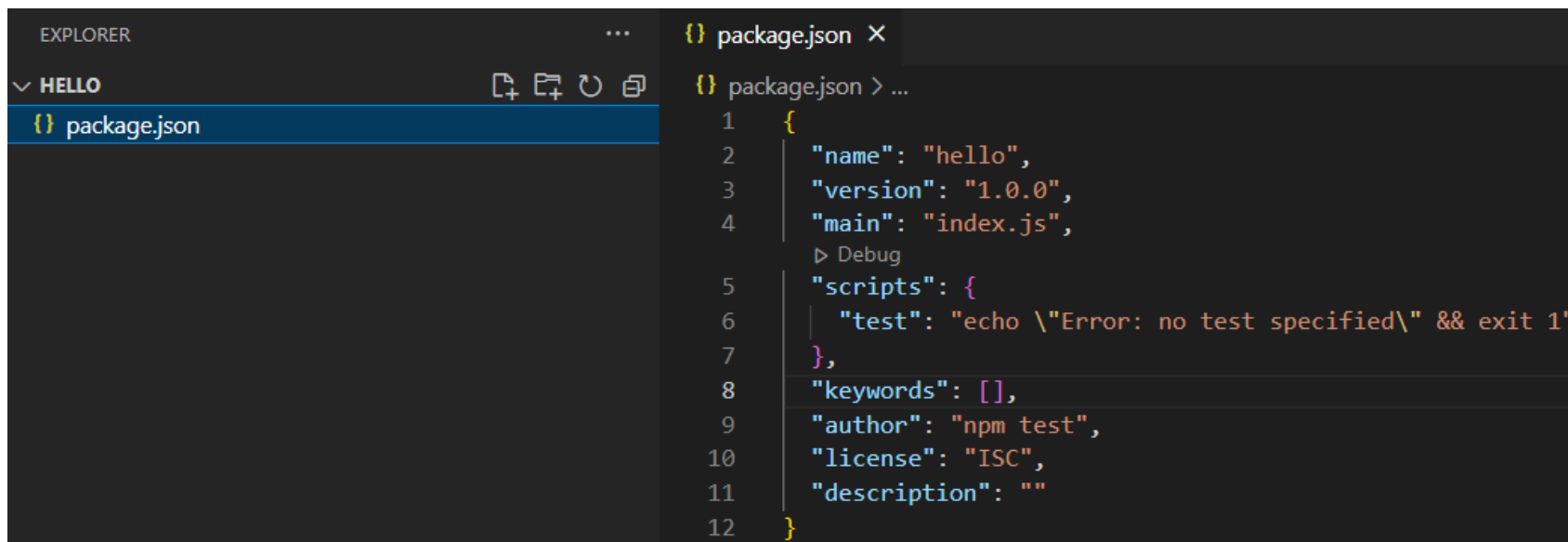
<https://nodejs.org/en/>

Node.js nowy projekt

w Visual Studio Code w katalogu projektu (hello) w terminalu wydajemy polecenie inicjalizujące projekt:

```
npm -y init
```

wygenerowany zostaje plik package.json z konfiguracją projektu



```
EXPLORER
HELLO
  package.json

package.json
1 {
2   "name": "hello",
3   "version": "1.0.0",
4   "main": "index.js",
5   "scripts": {
6     "test": "echo \"Error: no test specified\" && exit 1"
7   },
8   "keywords": [],
9   "author": "npm test",
10  "license": "ISC",
11  "description": ""
12 }
```

Node.js

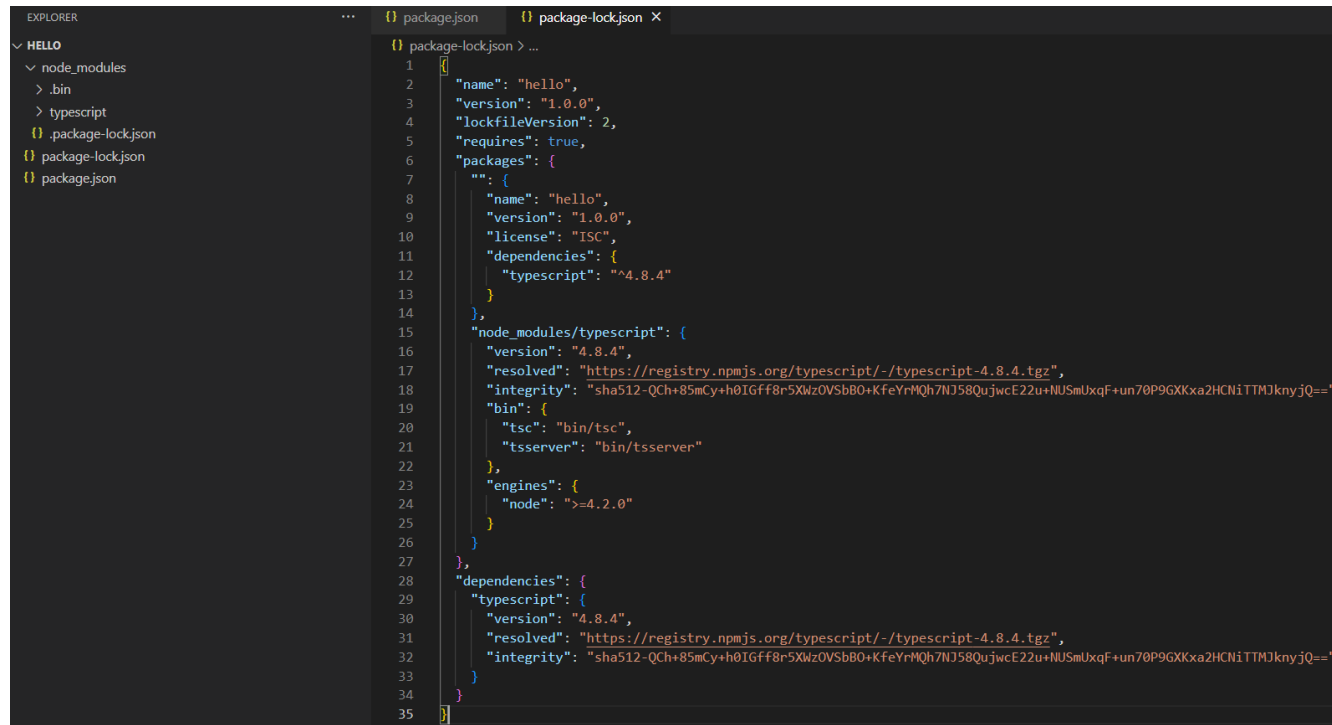
instalacja TypeScript

w Visual Studio Code w katalogu projektu (hello) w terminalu wydajemy polecenie instalujące język TypeScript:

```
npm install -g typescript
```

//g: global

wygenerowany zostaje plik package-lock.json z informacjami dotyczącymi języka oraz katalog node_modules z plikami obsługującymi TypeScript (przy opcji -g pliki mogą być wygenerowane w innym katalogu)



```
1  {}
2  "name": "hello",
3  "version": "1.0.0",
4  "lockfileVersion": 2,
5  "requires": true,
6  "packages": {
7    "": {
8      "name": "hello",
9      "version": "1.0.0",
10     "license": "ISC",
11     "dependencies": {
12       "typescript": "^4.8.4"
13     }
14   },
15   "node_modules/typescript": {
16     "version": "4.8.4",
17     "resolved": "https://registry.npmjs.org/typescript/-/typescript-4.8.4.tgz",
18     "integrity": "sha512-QCh+85mCy+h0IGfF8r5XWz0VS6B0+KfeYrMQh7NJ58QujwcE22u+NU5mUxqf+un70P9GXKxa2HCNiTTMjknjQ==",
19     "bin": {
20       "tsc": "bin/tsc",
21       "tsserver": "bin/tsserver"
22     },
23     "engines": {
24       "node": ">=4.2.0"
25     }
26   }
27 },
28 "dependencies": {
29   "typescript": {
30     "version": "4.8.4",
31     "resolved": "https://registry.npmjs.org/typescript/-/typescript-4.8.4.tgz",
32     "integrity": "sha512-QCh+85mCy+h0IGfF8r5XWz0VS6B0+KfeYrMQh7NJ58QujwcE22u+NU5mUxqf+un70P9GXKxa2HCNiTTMjknjQ=="
33   }
34 }
35 }
```

Node.js wersja TypeScript

sprawdzenie zainstalowanej wersji języka

```
tsc --version
```

```
• Version 5.4.3
```

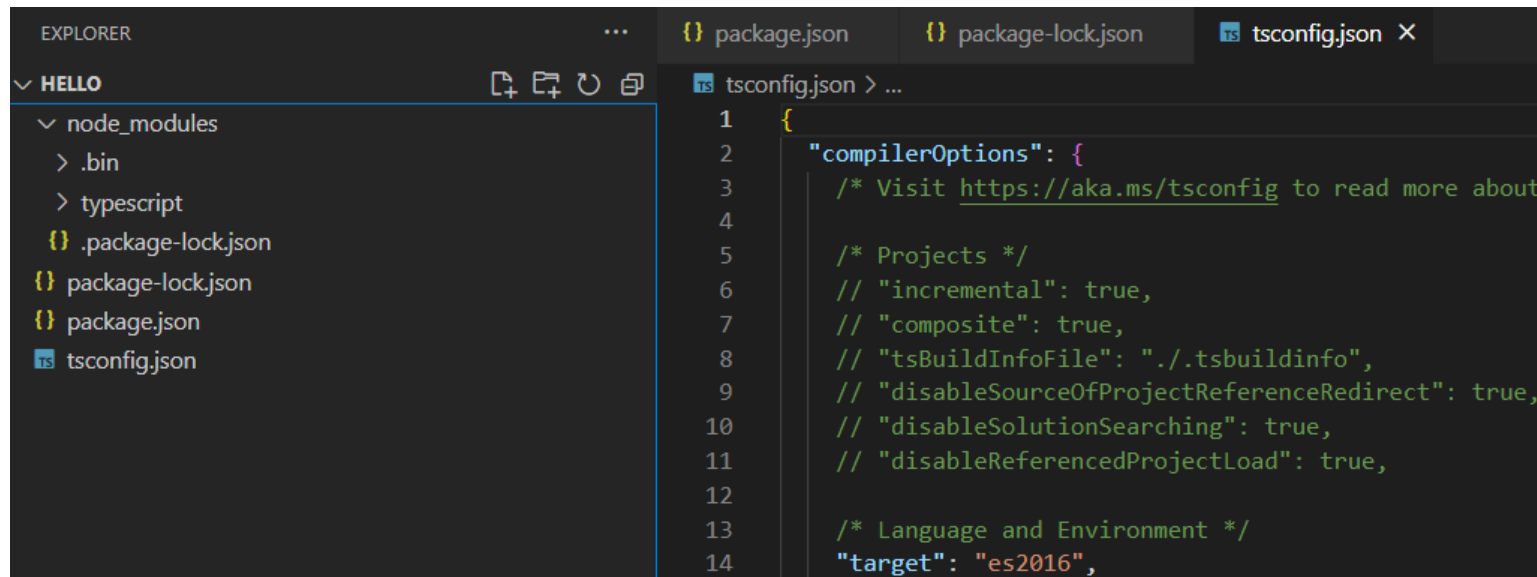
Node.js inicjalizacja TypeScript

w Visual Studio Code w katalogu projektu (hello) w terminalu wydajemy polecenie inicjalizujące język TypeScript:

```
hello> npx tsc -init
```

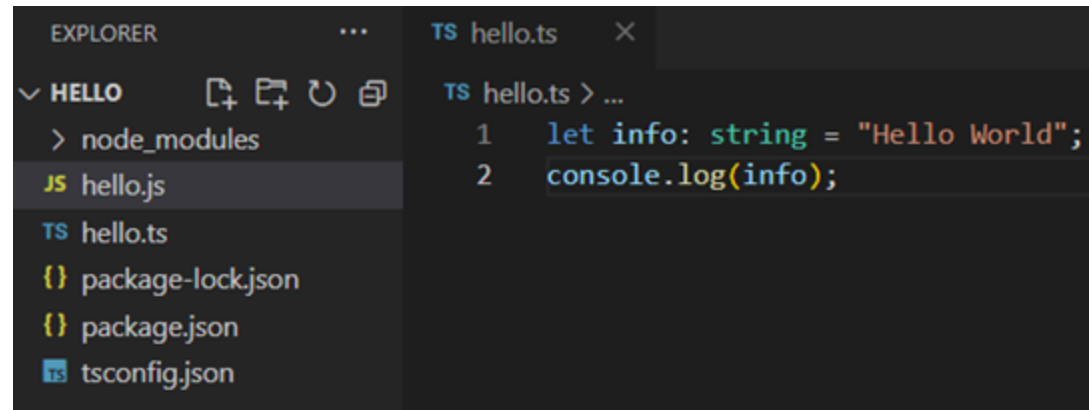
npx: Node Package Execute - uruchomienie modułu z katalogu node_modules
tsc: typescript compiler

wygenerowany zostaje plik tsconfig.json z konfiguracją kompilatora



```
tsconfig.json > ...
1  {
2    "compilerOptions": {
3      /* Visit https://aka.ms/tsconfig to read more about
4
5      /* Projects */
6      // "incremental": true,
7      // "composite": true,
8      // "tsBuildInfoFile": "./.tsbuildinfo",
9      // "disableSourceOfProjectReferenceRedirect": true,
10     // "disableSolutionSearching": true,
11     // "disableReferencedProjectLoad": true,
12
13     /* Language and Environment */
14     "target": "es2016",
```

Node.js plik hello.ts



```
EXPLORER
HELLO
  > node_modules
  JS hello.js
  TS hello.ts
  {} package-lock.json
  {} package.json
  tsconfig.json

TS hello.ts
TS hello.ts > ...
1 let info: string = "Hello World";
2 console.log(info);
```

tworzymy plik hello.ts w języku TypeScript

Node.js TypeScript

aby uruchomić plik ts w środowisku node.js należy go przekonwertować do js

w Visual Studio Code w katalogu projektu (hello) w terminalu kompilujemy plik hello.ts - wydajemy polecenie konwertujące plik hello.ts do języka JavaScript:

```
npx tsc hello.ts
```

wygenerowany zostaje plik w języku JavaScript hello.js który można uruchomić w node.js



The screenshot shows the Visual Studio Code interface with three panels. The left panel is the Explorer, showing a project named 'HELLO' with files: 'node_modules', 'hello.js', 'hello.ts', 'package-lock.json', 'package.json', and 'tsconfig.json'. The middle panel shows the source TypeScript file 'hello.ts' with two lines of code: '1 let info: string = "Hello World";' and '2 console.log(info);'. The word 'string' is circled in red. The right panel shows the compiled JavaScript file 'hello.js' with two lines of code: '1 var info = "Hello World";' and '2 console.log(info);'. The third line in the JS panel is empty. Green arrows point from the text above to the 'string' type in the TS panel and the 'var info' declaration in the JS panel.

plik źródłowy (jest typ string - statyczny mechanizm typowania)

wygenerowany plik po kompilacji (brak typu string - dynamiczny mechanizm typowania)

Node.js TypeScript

```
PS D:\hello> node hello.js  
Debugger attached.  
Hello World  
Waiting for the debugger to disconnect...
```

aby uruchomić plik hello.js wpisujemy polecenie:

```
node hello.js
```

```
npx tsc --watch
```

polecenie uruchamiające tryb nasłuchu – automatyczną kompilację pliku ts po wykryciu w nim zmiany

TypeScript

hello.ts

```
TS hello.ts > ...
1 let tekst: string = "napis"; // typ tekstowy
2 let decimal: number = 3.14; // typ liczbowy dziesiętny
3 let hex: number = 0xffff; // typ liczbow szesnastkowy
4 let octal: number = 0o765; // typ liczbowy ósemkowy
5 let binary: number = 0b1110; // typ liczbowy binarny
6 let logic: boolean = true; // typ logiczny
7
8 let wszystkoJednoCo: any = 23; // typ dynamiczny
9 wszystkoJednoCo = "napis"; // typ string
10 wszystkoJednoCo = false; // typ boolean
11
12 console.log(`wszystkoJednoCo:
13     wartość: ${wszystkoJednoCo}
14     nazwa typu: ${typeof wszystkoJednoCo}`);
15
16 let common: number | string // zmienna może przyjmować
17 // typ tekstowy lub liczbowy
18 let marka: "bmw" | "mercedes" | "ford"; // typ jako zbiór wartości
19 //marka = "opel"; // bład - brak opła w zdefiniowanych wartościach'
20
21 let auta: string[] = ["bmw", "mercedes", "ford"]; // tablica
22 let krotka: [number, boolean] = [25, true]; // krotka - tablica różnych typów
23 enum colors {red, green, blue}; // enum - wyliczenie
24 let czerwony = colors.red; // korzystanie z enum colors
25 console.log(czerwony);
```

statyczne typowanie w TypeScript

hello.js

```
JS hellojs > ...
1 var tekst = "napis"; // typ tekstowy
2 var decimal = 3.14; // typ liczbowy dziesiętny
3 var hex = 0xffff; // typ liczbow szesnastkowy
4 var octal = 501; // typ liczbowy ósemkowy
5 var binary = 14; // typ liczbowy binarny
6 var logic = true; // typ logiczny
7
8 var wszystkoJednoCo = 23; // typ dynamiczny
9 wszystkoJednoCo = "napis"; // typ string
10 wszystkoJednoCo = false; // typ boolean
11
12 console.log(`wszystkoJednoCo:
13     wartość: ${wszystkoJednoCo}
14     nazwa typu: ${typeof wszystkoJednoCo}`);
15
16 var common; // zmienna może przyjmować
17 // typ tekstowy lub liczbowy
18 var marka; // typ jako zbiór wartości
19 //marka = "opel"; // bład - brak opła w zdefiniowanych wartościach'
20
21 var auta = ["bmw", "mercedes", "ford"]; // tablica
22 var krotka = [25, true]; // krotka - tablica różnych typów
23 var colors;
24 (function (colors) {
25     colors[colors["red"] = 0] = "red";
26     colors[colors["green"] = 1] = "green";
27     colors[colors["blue"] = 2] = "blue";
28 })(colors || (colors = {}));
29 ; // enum - wyliczenie
30 var czerwony = colors.red; // korzystanie z enum colors
31 console.log(czerwony);
32
```

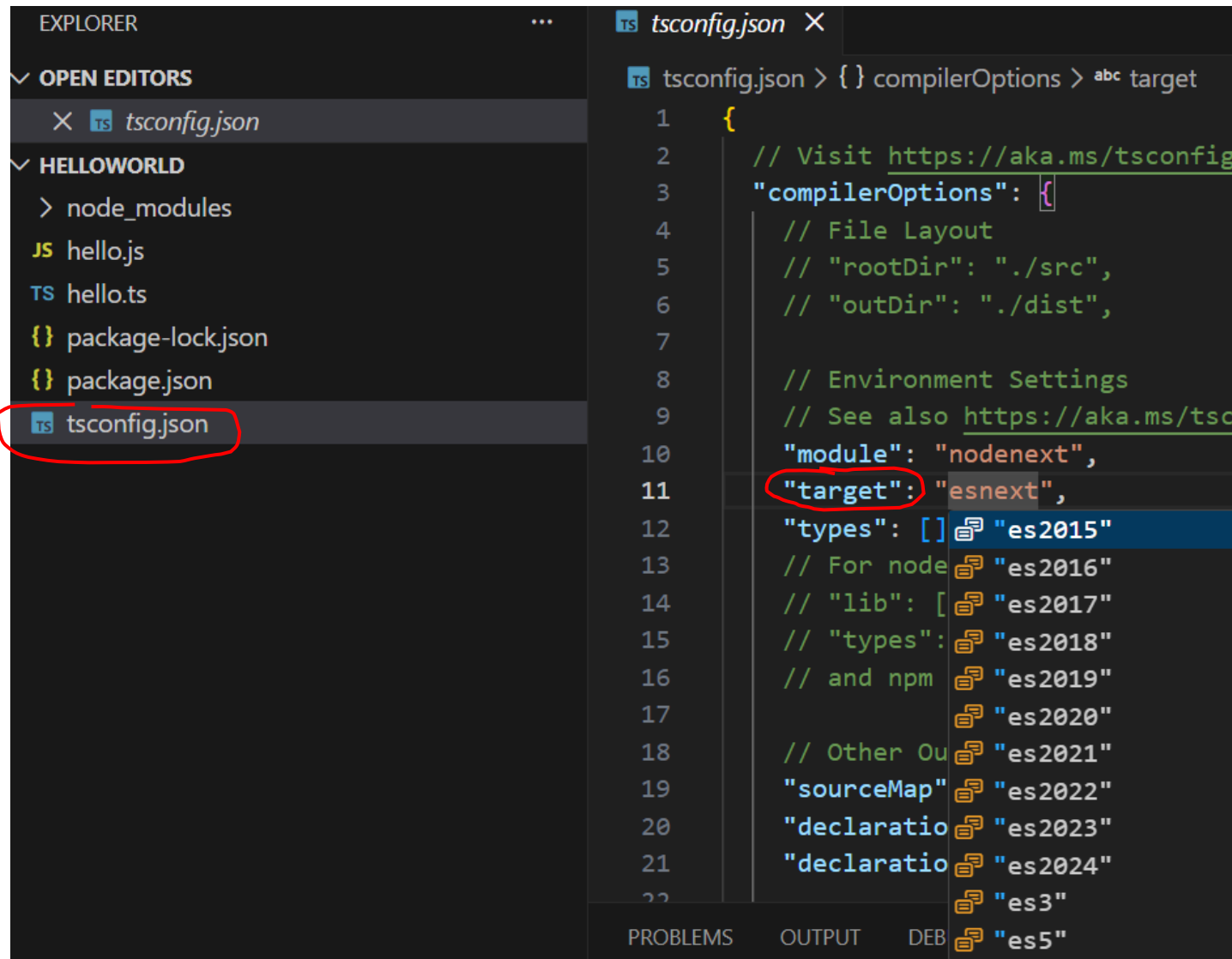
```
PS D:\hello> node hello.js
Debugger attached.
wszystkoJednoCo:
    wartość: false
    nazwa typu: boolean
```

0

dynamiczne typowanie w JavaScript
(wygenerowany plik hello.js może się różnić w zależności od wybranej wersji kompilatora)

uruchomienie hello.js

TypeScript



The image shows a screenshot of the Visual Studio Code editor. On the left, the Explorer sidebar shows a project structure with a file named `tsconfig.json` highlighted with a red circle. The main editor window displays the contents of `tsconfig.json`. The file is a JSON configuration for the TypeScript compiler. The `"target"` property is highlighted with a red circle, and its value is `"esnext"`. A dropdown menu is open for the `"types"` property, showing a list of ECMAScript versions from `"es2015"` to `"es5"`. The `"es2015"` option is currently selected and highlighted in blue. The breadcrumb at the top of the editor indicates the current path: `tsconfig.json > { } compilerOptions > abc target`.

```
1 {
2   // Visit https://aka.ms/tsconfig
3   "compilerOptions": {
4     // File Layout
5     // "rootDir": "./src",
6     // "outDir": "./dist",
7
8     // Environment Settings
9     // See also https://aka.ms/tsconfig
10    "module": "nodenext",
11    "target": "esnext",
12    "types": [ "es2015"
13              // For node "es2016"
14              // "lib": [ "es2017"
15              // "types": "es2018"
16              // and npm "es2019"
17                    "es2020"
18              // Other Ou "es2021"
19    "sourceMap" "es2022"
20    "declaratio "es2023"
21    "declaratio "es2024"
22                    "es3"
23                    "es5"
```

wybór wersji
kompilatora TypeScript

TypeScript - obiekty

napisz stronę internetową o podanej funkcjonalności. Całość ma się składać z plików index.html oraz app.ts (nie piszemy skryptu w js tylko w ts, plik ts przekonwertujemy do js w środowisku node.js)

skrypt w ts pobiera wartości pól do obiektu samochod

Samochód

marka Volvo

rok 2014

stan bardzo dobry

pokaz

marka: Volvo

rok: 2014

stan: bardzo dobry

skrypt wyświetla informacje z obiektu samochod po naciśnięciu na przycisk „pokaz”

TypeScript – wbudowane typy DOM

W TypeScriptie dla formularzy HTML używa się wbudowanych typów DOM, takich jak `HTMLInputElement`.

Najczęściej spotykane typy to:

Najważniejsze typy elementów formularza

- `HTMLInputElement` – `<input>`
- `HTMLTextAreaElement` – `<textarea>`
- `HTMLSelectElement` – `<select>`
- `HTMLOptionElement` – `<option>`
- `HTMLButtonElement` – `<button>`
- `HTMLFormElement` – `<form>`
- `HTMLLabelElement` – `<label>`

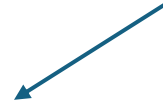
Wszystkie one dziedziczą po `HTMLElement`.

TypeScript - obiekty

```
<!DOCTYPE html>
<html>
<head>
  <meta charset='utf-8'>
  <title>TypeScript</title>
</head>
<body>
  <p>Samochód</p>
  marka<input id="marka" type="text"><br>
  rok<input id="rok" type="text"><br>
  stan<input id="stan" type="text"><br>
  <button id="btn-pokaz">pokaz</button>

  <p id="info"></p>
  <script src='app.js'></script>
</body>
</html>
```

index.html



app.ts



```
let info = document.getElementById("info");
let btnPokaz = document.getElementById("btn-pokaz");

btnPokaz.onclick = function(){
  // obiekt samochod
  let samochod: { marka: string, rok: number, stan: string }={
    marka : (<HTMLInputElement>document.getElementById("marka")).value,
    rok : parseInt((<HTMLInputElement>document.getElementById("rok")).value),
    stan : (<HTMLInputElement>document.getElementById("stan")).value
  }

  info.innerHTML = `marka: ${ samochod.marka}<br>rok: ${samochod.rok}<br>stan: ${samochod.stan}`;
}
```

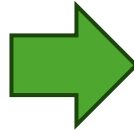
app.js powstaje po wygenerowaniu
z pliku app.ts

TypeScript - obiekty - errors

tsconfig.json – należy wyłączyć w ts sprawdzanie wartości null

/

'btnPokaz' is possibly 'null'.



```
104
105     /* Completeness */
106     // "skipDefaultLibCheck": true,           /* Skł
107     "skipLibCheck": true,                   /* Skł
108     "strictNullChecks": false // nie sprawdza wartosci NULL
109
110 }
111 }
```

Property 'value' does not exist on type 'HTMLElement'



Based on Tomasz Nurkiewicz's answer, the "problem" is that typescript is typesafe. :) So the `document.getElementById()` returns the type `HTMLElement` which does not contain a `value` property. The subtype `HTMLInputElement` does however contain the `value` property.

So a solution is to cast the result of `getElementById()` to `HTMLInputElement` like this:

```
var inputValue = (<HTMLInputElement>document.getElementById(elementId)).value;
```

<https://stackoverflow.com/questions/12989741/the-property-value-does-not-exist-on-value-of-type-htmlelement>

TypeScript - aliasy

inna nazwa typu zmiennej

Uczeń

imię

nazwisko

wiek

imię: Anna

nazwisko: Kowalska

wiek: 12

skrypt w ts pobiera wartości
pól do obiektu osoba
stworzonego z wykorzystaniem
aliasów

skrypt wyświetla informacje z obiektu osoba stworzonego z wykorzystaniem aliasów
po naciśnięciu na przycisk „pokaż”

TypeScript - aliasy

```
TS app.ts  X
Aliases > TS app.ts > ...
1
2 let info = document.getElementById("info");
3 let btnPokaz = document.getElementById("btn-pokaz");
4
5 // lokalne aliasy podstawowych typów danych
6 type WiekUcznia = number;
7 type ImieUcznia = string;
8 type NazwiskoUcznia = string;
9
10 type Uczeń = {
11     imie: ImieUcznia,
12     nazwisko: NazwiskoUcznia,
13     wiek: WiekUcznia
14 }
15
16 btnPokaz.onclick = function(){
17     // obiekt uczen
18     let osoba: Uczeń = {
19         imie : (<HTMLInputElement>document.getElementById("imie")).value,
20         nazwisko : (<HTMLInputElement>document.getElementById("nazwisko")).value,
21         wiek: parseInt((<HTMLInputElement>document.getElementById("wiek")).value)
22     }
23
24
25     info.innerHTML = `imię: ${osoba.imie}<br>nazwisko: ${osoba.nazwisko}<br>wiek: ${osoba.wiek}`;
26 }
```

<> index.html X

Aliases > <> index.html > ...

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset='utf-8'>
5     <title>TypeScript</title>
6 </head>
7 <body>
8     <p>Uczeń</p>
9     imię<input id="imie" type="text"><br>
10    nazwisko<input id="nazwisko" type="text"><br>
11    wiek<input id="wiek" type="text"><br>
12    <button id="btn-pokaz">pokaż</button>
13
14    <p id="info"></p>
15    <script src='app.js'></script>
16 </body>
17 </html>
```

TypeScript - interfejsy

skrypt w ts pobiera wartości pól do obiektu pracownik stworzonego na podstawie interfejsu Pracownik

Pracownik

imię
nazwisko
wiek
żonaty

imię: Jan
nazwisko: Kowalski
wiek: 34
żonaty: tak

skrypt wyświetla informacje z obiektu pracownik stworzonego na podstawie interfejsu Pracownik po naciśnięciu na przycisk „pokaż”

TypeScript - interfejsy

<> index.html ✕

Interfaces > <> index.html > ...

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset='utf-8'>
5     <title>TypeScript</title>
6 </head>
7 <body>
8     <p>Pracownik</p>
9     imię<input id="imie" type="text"><br>
10    nazwisko<input id="nazwisko" type="text"><br>
11    wiek<input id="wiek" type="text"><br>
12    żonaty<select id="zonaty"><option>tak</option><option>nie</option></select><br><br>
13    <button id="btn-pokaz">pokaż</button>
14
15    <p id="info"></p>
16    <script src='app.js'></script>
17 </body>
18 </html>
```

TypeScript - interfejsy

```
TS app.ts  X
Interfaces > TS app.ts > ...
1
2 let info = document.getElementById("info");
3 let btnPokaz = document.getElementById("btn-pokaz");
4
5 // interfejs Pracownik - zamiast tworzyć kolejne obiekty pracowników,
6 // z których każdy posiada definicję, można utworzyć jeden wspólny interfejs.
7 interface Pracownik {
8     imie: string,
9     nazwisko: string,
10    wiek: number,
11    zonaty: string
12 }
13
14
15 btnPokaz.onclick = function(){
16     // obiekt pracownik
17     let pracownik: Pracownik = {
18         imie : (<HTMLInputElement>document.getElementById("imie")).value,
19         nazwisko : (<HTMLInputElement>document.getElementById("nazwisko")).value,
20         wiek: parseInt((<HTMLInputElement>document.getElementById("wiek")).value),
21         zonaty: (<HTMLInputElement>document.getElementById("zonaty")).value
22     }
23
24
25     info.innerHTML = `imię: ${ pracownik.imie}<br>nazwisko: ${pracownik.nazwisko}
26     | | | | | | | | | | <br>wiek: ${pracownik.wiek}<br>żonaty: ${pracownik.zonaty}`;
27 }
```

TypeScript - klasy

Samochód

marka Mercedes

rok 2015

stan zły

dodaj

pokaz

po naciśnięciu na przycisk „dodaj”
skrypt w ts pobiera wartości pól do
obiektu klasy Car,
dodaje obiekt do tablicy cars
obiektów klasy Car.
oraz wyświetla informacje o
dodanym samochodzie

marka: Volvo

rok: 2023

stan: średni

marka: Mercedes

rok: 2015

stan: zły

po naciśnięciu na przycisk „pokaz”
skrypt wyświetla informacje o
samochodach z tablicy cars
obiektów klasy Car

TypeScript - klasy

<> index.html ✕

klasy > <> index.html > html > body > script

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <meta charset='utf-8'>
5    <title>TypeScript</title>
6  </head>
7  <body>
8    <p>Samochód</p>
9    marka<input id="marka" type="text"><br>
10   rok<input id="rok" type="text"><br>
11   stan<input id="stan" type="text"><br>
12   <button id="btn-dodaj">dodaj</button>
13   <button id="btn-pokaz">pokaz</button>
14
15   <p id="info"></p>
16   <script src='app.js'></script>
17 </body>
18 </html>
```

TypeScript - klasy

```
TS app.ts ×
klasy > TS app.ts > onclick
1 let info = document.getElementById("info");
2 let btnDodaj = document.getElementById("btn-dodaj");
3 let btnPokaz = document.getElementById("btn-pokaz");
4
5 // definicja klasy Car
6 class Car{
7
8     private type: string;
9     private year: number;
10    private condition: string;
11
12    public constructor(t: string, y: number, c: string){
13        this.type = t;
14        this.year = y;
15        this.condition = c;
16    }
17
18    public getType(): string{
19        return this.type;
20    }
21
22    public getYear(): number{
23        return this.year;
24    }
25
26    public getCondition(): string{
27        return this.condition;
28    }
29 }
```

prywatne pola klasy

konstruktor

publiczne metody klasy

TypeScript - klasy

```
31 // tablica obiektów klasy Car (pusta)
32 const cars: Car[] = [];
33
34
35 // dodanie samochodu do tablicy, wyświetlenie o nim informacji
36 btnDodaj.onclick = function(){
37     let t = (<HTMLInputElement>document.getElementById("marka")).value;
38     let y = parseInt((<HTMLInputElement>document.getElementById("rok")).value);
39     let c = (<HTMLInputElement>document.getElementById("stan")).value;
40
41     // utworzenie obiektu car klasy Car
42     let car = new Car(t,y,c);
43     // wstawienie utworzonego obiektu do tablicy cars
44     cars.push(car);
45     // wyświetlenie informacji o obiekcie
46     info.innerHTML=`marka: ${car.getType()}<br>rok: ${car.getYear()}<br>stan: ${car.getCondition()} `;
47 }
48
49 // wyświetlenie informacji o samochodach znajdujących się w tablicy
50 btnPokaz.onclick = function(){
51     info.innerHTML = "";
52     // pętla po elementach tablicy cars
53     for (const element of cars) {
54         info.innerHTML += `<br>marka: ${element.getType()}<br>rok: ${element.getYear()}
55         |         |         |         |         |         |
56         <br>stan: ${element.getCondition()}<br>`;
57     }
```

TypeScript – klasy

klasa Car alternatywnie


```
// klasa Car
class Car {
  // nie deklarujemy pól klasy, tylko podajemy je w konstruktorze
  public constructor(private type: string, private year: number, private condition: string) {
  }

  public getType(): string {
    return this.type;
  }

  public getYaer(): number{
    return this.year;
  }

  public getCondition(): string {
    return this.condition;
  }
}
```

alternatywny sposób deklaracji pól klasy –
konstruktor je wygeneruje



TypeScript – klasy

klasa Car alternatywnie

```
TS app.ts x
Classes3 > TS app.ts > onclick
1
2 let info = document.getElementById("info");
3 let btnPokaz = document.getElementById("btn-pokaz");
4 let btnDodaj = document.getElementById("btn-dodaj");
5
6 // klasa Car
7 class Car {
8     // konstruktor
9     public constructor(private type?: string, private year?: number, private condition?: string) {
10    }
11    // properties (właściwości)
12    get Type(){
13        | return this.type;
14    }
15
16    get Year(){
17        | return this.year;
18    }
19
20    get Condition(){
21        | return this.condition;
22    }
23
24    set Type(value){
25        | this.type = value;
26    }
27    set Year(value){
28        | this.year = value;
29    }
30
31    set Condition(value){
32        | this.condition = value;
33    }
34 }
35
```

alternatywny sposób deklaracji
pól klasy – konstruktor je wygeneruje,
pola są nieobowiązkowe (znak
zapytania)

properties (właściwości) – zamiast
deklaracji metod geterów i seterów

TypeScript – klasy

klasa Car alternatywnie

```
35
36 const cars: Car[] = [];
37
38 btnDodaj.onclick = function(){
39     const car = new Car();
40
41     car.Type = (<HTMLInputElement>document.getElementById("marka")).value;
42     car.Year = parseInt((<HTMLInputElement>document.getElementById("rok")).value);
43     car.Condition = (<HTMLInputElement>document.getElementById("stan")).value
44
45     cars.push(car);
46
47     info.innerHTML = `marka: ${car.Type}<br>rok: ${car.Year}<br>stan: ${car.Condition}`;
48
49 }
50
51
52 btnPokaz.onclick = function(){
53     info.innerHTML = "";
54     for (const element of cars) {
55         info.innerHTML += `<br>marka: ${element.Type}<br>rok: ${element.Year}<br>stan: ${element.Condition}<br>`;
56     }
57
58 }
```

konstruktor wygeneruje pola klasy (pomimo, że w definicji klasy są one opcjonalne)

z właściwości (properties) korzystamy jak z pól klasy

TypeScript - funkcje

kalkulator

3.3 + 12 = 15.3

+
-
*
/

napisz kalkulator korzystając z
poniższych funkcji
(trzeba je napisać)

```
btn.onclick = function () {  
  // pobieramy liczby oraz operator z formularza  
  var data = getNumbersAndOperator();  
  // liczymy wynik działania  
  var result = count(data);  
  // wypisujemy rezultat  
  printResult(result);  
};
```

kalkulator

3.3 / 0 = nie dzielimy przez zero!

TypeScript - funkcje

```
index.html x
Functions > index.html > html > body > input#a
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>kalkulator</title>
5      <meta charset="utf-8">
6      <link rel="stylesheet" href="main.css">
7      <script src="app.js" defer</script>
8    </head>
9    <body>
10   <h1>kalkulator</h1>
11   <input id="a" type="number">
12   <select id="op">
13     <option>+</option>
14     <option>-</option>
15     <option>*</option>
16     <option>/</option>
17   </select>
18   <input id="b" type="number">
19   <label>=</label>
20   <label id="wynik"></label><br>
21   <button id="btn">oblicz</button>
22 </body>
23 </html>
```

TypeScript - funkcje

```
TS app.ts ×
Functions > TS app.ts > ...
1  let btn = document.getElementById("btn");
2
3  // interfejs Data (dane)
4  interface Data {
5      a: number,
6      b: number,
7      op: string
8  }
9
10 btn.onclick = function(){
11
12     // pobieramy liczby oraz operator z formularza
13     let data = getNumbersAndOperator();
14     // liczymy wynik działania
15     let result = count(data);
16     // wypisujemy rezultat
17     printResult(result);
18
19 }
20
21 // pobiera dane z formularza
22 function getNumbersAndOperator() : Data{
23     let aa = parseFloat((<HTMLInputElement>document.getElementById("a")).value);
24     let bb = parseFloat((<HTMLInputElement>document.getElementById("b")).value);
25     let opop = (<HTMLInputElement>document.getElementById("op")).value;
26     let data: Data = {
27         a : aa,
28         b: bb,
29         op: opop
30     }
31     // zwracamy obiekt typu Numbers
32     return data;
33 }
```

TypeScript - funkcje

```
34
35 // funkcja obliczajaca wynik zwraca liczbe lub napis (komunikat przy dzieleniu przez zero)
36 function count(d: Data) : number | string{
37     let result: number | string = 0;
38     switch (d.op){
39     case "+":
40         result = d.a + d.b;
41         break;
42     case "-":
43         result = d.a - d.b;
44         break;
45     case "*":
46         result = d.a * d.b;
47         break;
48     case "/":
49         if (d.b != 0){
50             result = d.a / d.b;
51         }else{
52             result = "nie dzielimy przez zero!"
53         }
54         break;
55     }
56
57     return result;
58 }
59
60 // przy wypisywaniu wyniku należy uwzględnić 2 typy: string oraz number
61 function printResult(res: number | string){
62     if (typeof res == "string"){
63         document.getElementById("wynik").innerHTML = res;
64     }else{
65         document.getElementById("wynik").innerHTML = res.toString();
66     }
67 }
```

TypeScript – funkcje parametry opcjonalne

parametr opcjonalny

```
function orderIce(yourChoice? :string){  
  console.log(yourChoice || "lody z polewą czekoladową")  
}  
  
orderIce("lody z polewą truskawkową")  
orderIce()
```

lody z polewą truskawkową
lody z polewą czekoladową

wywołanie funkcji z
parametrem opcjonalnym

TypeScript – funkcje named parameters

named parameters



```
function orderCar({type, year, isAfterAccident}: {type:string, year: number, isAfterAccident: boolean }){  
  | console.log(`you choose: \ntype: ${type}\nafter accident:${isAfterAccident}\nyear: ${year}`)  
  }  
  
orderCar({type:"Volvo", year: 2023, isAfterAccident:false});
```

```
you choose:  
type: Volvo  
after accident:false  
year: 2023
```

TypeScript – funkcje rest parameter

rest parameter

```
function add(a: number, b: number, ...tab: number[]): number {  
    let result: number = 0;  
    result = a + b;  
    tab.forEach((x) => result += x);  
    return result;  
}  
console.log(add(1, 2, 3, 4, 5, 6, 7, 8, 9, 10));
```

rest parameter

a b

rest parameter:

- funkcja może mieć tylko jeden taki parametr
- musi być na końcu wszystkich parametrów
- musi być typu Array (tablica)

implementacja interfejsów, dziedziczenie

```
TS app.ts ×
TS app.ts > Swimmable > swim
1 // interfejs jest kontraktem - gdy klasa go implementuje -
2 // musi zaimplementowac wszystkie pola i metody interfejsu
3 interface Flyable {
4   fly(): void;
5 }
6
7 interface Swimmable {
8   distance: number;
9   swim(): void;
10 }
11
12 interface Soundable {
13   volume: number;
14   makeSound(): void;
15 }
16
17 interface Diveable {
18   depth: number;
19   dive(): void;
20 }
21
22 // klasa bazowa
23 class Animal {
24   private name: string;
25
26   constructor(_name: string) {
27     this.name = _name;
28   }
29   protected introduceYourself(){
30     console.log(`nazywam się ${this.name} :)`);
31   }
32 }
```

interfejsy

klasa bazowa

implementacja interfejsów, dziedziczenie

```
app.ts  X
TS app.ts > Swimmable > swim
33
34 // klasa Duck dziedziczy po klasie bazowej Animal wszystkie jej pola i metody
35 // (można dziedziczyć tylko po jednej klasie)
36 // implementuje kilka interfejsów
37 class Duck extends Animal implements Flyable, Swimmable, Soundable, Diveable {
38     distance = 1000;
39     volume = 30;
40     depth = 100;
41     fly(): void {
42         console.log("Potrafię latać! ");
43     }
44     swim(): void {
45         console.log(`Potrafię przepłynąć ${this.distance} m`);
46     }
47     makeSound(): void {
48         console.log(`Kwa kwa o głośności ${this.volume} dB`);
49     }
50     dive():void{
51         console.log(`umiem nurkować na głębokość ${this.depth} m`);
52     }
53     introduce(){
54         this.introduceYourself()
55     }
56 }
57
58 let duck:Duck = new Duck("kaczka");
59 //duck.introduceYourself(); // błąd
60 duck.introduce();
61 duck.fly();
62 duck.swim();
63 duck.makeSound();
64 duck.dive();
```

klasa dziedzicząca po
klasie bazowej
implementująca
interfejsy

```
nazywam się kaczka :)
Potrafię latać!
Potrafię przepłynąć 1000 m
Kwa kwa o głośności 30 dB
umiem nurkować na głębokość 100
```

wzorce projektowe

([ang. design patterns](#)) – uniwersalne, sprawdzone w praktyce rozwiązania często pojawiających się, powtarzalnych problemów projektowych

Design Patterns: Elements of Reusable Object-Oriented Software (1994) is a [software engineering](#) book describing [software design patterns](#). The book was written by [Erich Gamma](#), [Richard Helm](#), [Ralph Johnson](#), and [John Vlissides](#), with a foreword by [Grady Booch](#). The book is divided into two parts, with the first two chapters exploring the capabilities and pitfalls of [object-oriented programming](#), and the remaining chapters describing 23 classic [software design patterns](#). The book includes examples in [C++](#) and [Smalltalk](#).

It has been influential to the field of software engineering and is regarded as an important source for object-oriented design theory and practice. More than 500,000 copies have been sold in English and in 13 other languages.^[1] The authors are often referred to as the Gang of Four (GoF).^{[2][3][4][5]}

Development and publication history [\[edit\]](#)

The book started at a birds-of-a-feather session at the 1990 [OOPSLA](#) meeting,

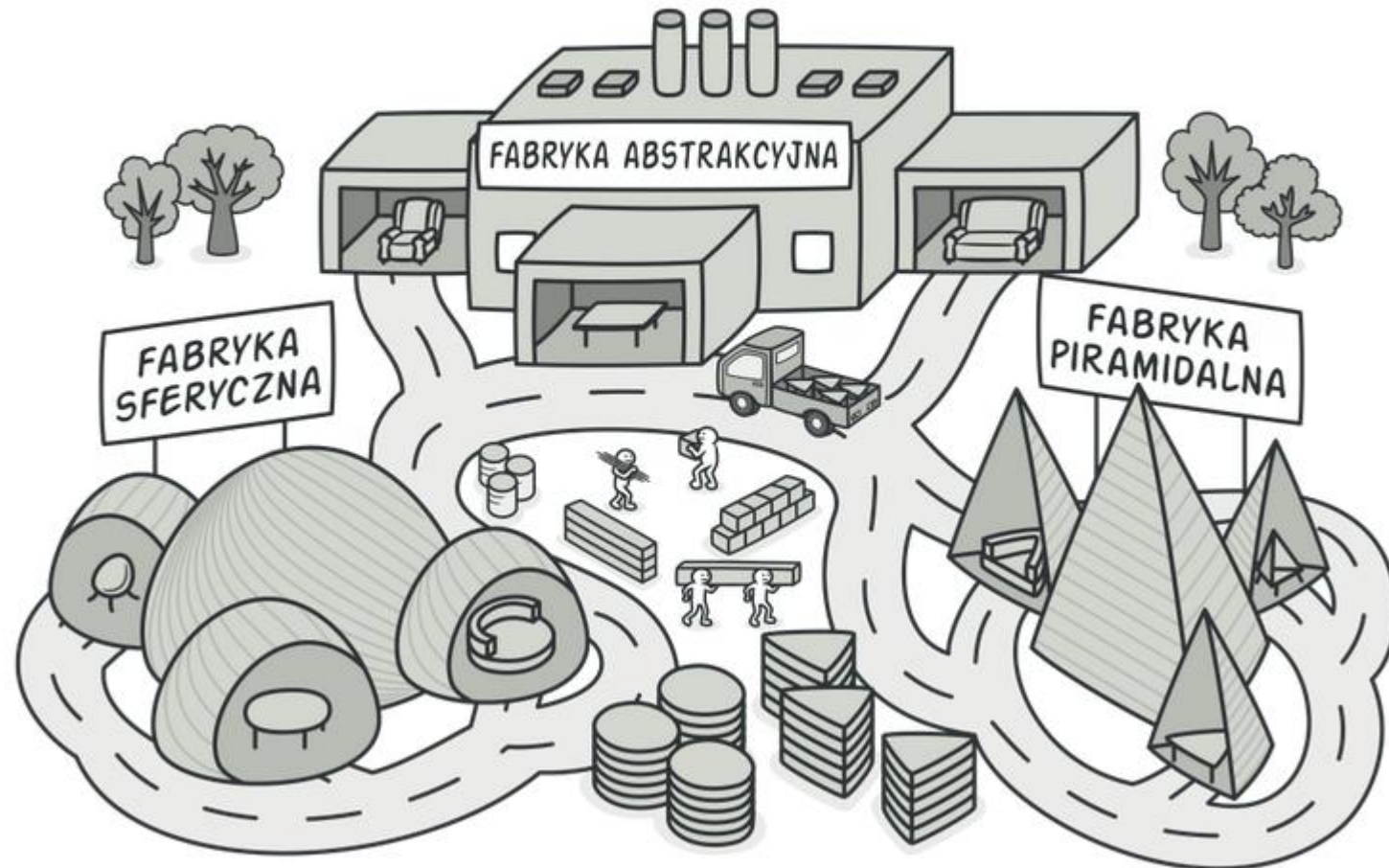
Design Patterns: Elements of Reusable Object-Oriented Software



Author	Erich Gamma Richard Helm Ralph Johnson John Vlissides	ważna książka
Subject	Design patterns , software engineering , object-oriented programming	

Fabryka abstrakcyjna

Fabryka abstrakcyjna jest kreatywnym wzorcem projektowym, który pozwala tworzyć rodziny spokrewnionych ze sobą obiektów bez określania ich konkretnych klas.



Fabryka abstrakcyjna

```
TS app.ts ×
TS app.ts > ...
1 //https://refactoring.guru/design-patterns/abstract-factory
2
3 interface Chair {
4   name: string;
5 }
6
7 class ArtDecoChair implements Chair {
8   name = "Jestem krzesło Art Deco";
9 }
10
11 class ModernChair implements Chair {
12   name = "Jestem krzesło Modern";
13 }
14
15 interface FurnitureFactory {
16   createChair(): Chair;
17 }
18
19 class Furniture {
20   name: string = '';
21 }
22
23 class ArtDecoFurnitureFactory implements FurnitureFactory {
24   createChair(): Chair {
25     return new ArtDecoChair();
26   }
27 }
28
29 class ModernChairFurnitureFactory implements FurnitureFactory {
30   createChair(): Chair {
31     return new ModernChair();
32   }
33 }
34
35 class Client extends Furniture {
36   constructor(
37     _name: string,
38     private furnitureFactory: FurnitureFactory,
39   ) {
40     super(); // uruchamia konstruktor klasy bazowej Furniture
41     this.name = _name;
42
43     console.log(
44       `hello ${this.name}, wyprodukowaliśmy dla Ciebie: ${this.furnitureFactory.createChair().name}`
45     );
46   }
47 }
48 const artDecoFurnitureFactory: ArtDecoFurnitureFactory = new ArtDecoFurnitureFactory();
49 const modernChairFurnitureFactory: ModernChairFurnitureFactory = new ModernChairFurnitureFactory();
50
51 const artDecoClient: Client = new Client("artDecoClient", artDecoFurnitureFactory)
52 const modernChairClient: Client = new Client("modernChairClient", modernChairFurnitureFactory)
53
```

korzystając z kodu obok
zaimplementować resztę
mebli



Product families and their variants.

TypeScript

polecenia konsoli w Visual Studio Code dotyczące TypeScript w środowisku Node.js

```
cd katalog_projektu // przechodzimy do katalogu projektu
npm -y init // inicjalizacja projektu w node.js
npm install typescript // instalacja języka TypeScript w katalogu projektu
npx tsc -init // inicjalizacja TypeScript
npx tsc hello.ts // konwersja pliku hello.ts do języka JavaScript, w wyniku powstaje plik hello.js
node hello.js // uruchomienie pliku hello.js w środowisku node.js
npx tsc --watch // włączenie trybu nasłuchu – automatyczną kompilację pliku hello.ts
// po wykryciu w nim zmiany
```