TypeScript

OBIEKTY:

Pierwszy obiekt:

```
const dane = {
    proc: "AMD",
    RAM: "32 GB",
}
console.log(dane.RAM);
```

wynik działania:

```
32 GB
```

Obiekt z funkcją:

```
//Funkcje w obiektach
const zestaw ={
    sniadanie: 40,
    obiad: 120,
    doZaplaty: function(){
        return this.sniadanie+this.obiad
    }
}
console.log("Do zapłaty: "+zestaw.doZaplaty());
```

wynik działania:

```
Do zapłaty: 160
```

PATRZ NASTĘPNA STRONA:

Użycie funkcji w wielu obiektach

```
//Funkcja do wykorzystania w wielu obiektach
function pole():void{
    console.log(`Pole: ${this.sz*this.wys}`);
}
const prostokat1 = {
    sz: 20,
    wys: 40,
    obliczPole : pole
}
const prostokat2 = {
    sz: 4,
    wys: 5,
    obliczPole : pole
}
prostokat1.obliczPole();
prostokat2.obliczPole();
```

Wynik działania:

```
Pole: 800
Pole: 20
```

```
//Typowanie obiektów - czyli przypisanie typu danych do każdego pola obiektu
const hotel: {nazwa: string, standard_pokoi: string[], liczba_pokoi: number} = {
    nazwa: "Polana",
    standard_pokoi: ["basic","premium","vip"],
    liczba_pokoi: 20
}
//Wyświetlanie danych jak wcześniej
```

Można oczywiście przechowywać wiele obiektów w tablicy:

```
const tablica = [
    {imie: "Anna", nazwisko: "Mucha", wiek: 40},
    {imie: "Jak", nazwisko: "Kochanowski", wiek: 20},
    {imie: "Robert", nazwisko: "Zima", wiek: 80}
]
console.log(tablica[0].imie); //wynik: Anna
```

```
//Typowanie obiektów - czyli przypisanie typu danych do każdego pola obiektu
const hotel: {nazwa: string, standard_pokoi: string[], liczba_pokoi: number} = {
    nazwa: "Polana",
    standard_pokoi: ["basic","premium","vip"],
    liczba_pokoi: 20
}
//Wyświetlanie danych jak wcześniej
```

KLASY

Inaczej szablon potrzebny do utworzenia OBIEKTU.

```
//KLASY
class Klient {
    imie: string;
    nazwisko: string;
    wiek: number;

    constructor(imie:string,nazwisko:string,wiek:number){
        this.imie = imie;
        this.nazwisko = nazwisko;
        this.wiek = wiek;
    }
    //metoda w obiekcie
    show(){
        console.log(`Dane: ${this.imie} ${this.nazwisko}  ${this.wiek}`);
    }
}
//Stworzenie obiektu
const kli = new Klient("Anna","Bizka",20);
kli.show(); //użycie metody obiektu
```

Dziedziczenie jest dokładnie tak samo przeprowadzone jak w Javie.

Listing 2.51. Użycie extends do utworzenia klasy potomnej

```
class Klient {
    imie: string;
    nazwisko: string;
    wiek: number;

    constructor(imie: string, nazwisko: string, wiek: number) {
        this.imie = imie;
        this.nazwisko = nazwisko;
        this.wiek = wiek;
    }
    powitanie() { console.log(`Witaj, ${this.imie} ${this.nazwisko}`); }

    czyPelnoletni() {
        if (this.wiek >= 18) return true;
        else return false;
    }
}

class Dane_klienta extends Klient { }

let Nowak = new Dane_klienta("Tadeusz", "Nowak", 44);
let Kowalski = new Dane_klienta("Jan", "Kowalski", 17);

console.log(Nowak.czyPelnoletni());
Kowalski.powitanie();
```

Dziedziczenie wraz z wpisaniem wartości do klasy poprzez konstruktor (słowo kluczowe **super**):
[na następnej stronie]

Listing 2.52. Użycie extends i super do utworzenia klasy potomnej

```
class Klient {
    imie: string;
    nazwisko: string;
    wiek: number;

    constructor(imie: string, nazwisko: string, wiek: number) {
        this.imie = imie;
        this.nazwisko = nazwisko;
        this.wiek = wiek;
    }
    powitanie() { console.log(`Witaj, ${this.imie} ${this.nazwisko}`); }

    czyPelnoletni() {
        if (this.wiek >= 18) return true;
        else return false;
    }
}

class DaneKlienta extends Klient {
    adres: string;

    constructor(imie: string, nazwisko: string, wiek: number, adres: string) {
        super(imie, nazwisko, wiek);
        this.adres = adres;
    }
}
```

Działają tutaj także modyfikatory dostępu:
private – blokowanie dostępu do pola dla każdego,
protected – klasa dziedzicząca ma dostęp do pola klasu po której dziedziczy.

Pola mogą też być typu readonly – czyli podczas tworzenia obiektu przypisujemy wartość do tego pola – **potem nie możemy już zmienić wartości tego pola.**