**TYPE SCRIPT ZADANIA Z FUNCTIONAL I OBJECT PROGRAMING**

1. **FUNCTIONAL PROGRAMING**

**ZAD 1. findPhraseInArray** Stwórz funkcję(arr:String[], phrase:string). Jeżeli szukana fraza istnieje w arr to zwróc jej wartość i index. Gdy nie istnieje to zwroc info ze nie ma.

type StringCoordinates = [el: string, index: number][];

**const** findPhraseInArray = (array: string[], phrase: string): string | StringCoordinates => {

**let** i = 0;

**let** filteredArr: StringCoordinates = []

array.**forEach**((el: string, index: number) => {

**if** (el.**toUpperCase**().includes(phrase.**toUpperCase**())) {

filteredArr.push([el, index]);

}

i++;

});

**if** (filteredArr.**length** === 0) **return** 'Tablica nie zawiera szukanej frazy';

**return** filteredArr;}

**ZAD 2. getMyAge** Twórz funkcję, która jako arg ma (rok urodzenia). Funkcja zwraca aktualny wiek niezależnie od typu otrzymanego inputa („1999” lub 1999 lub new Date(1990,1,1).

type InputAccepted = Date | number | string;

type GetMyAge = (Input: InputAccepted) => number | **void**;//**\***

**const** getMyAge: GetMyAge= **function** (input: InputAccepted): number | **void** { //**\***

**const** currentYear = **new** Date().**getFullYear**(); **\* nie musze typowac calej f. W 99%**

**if** (**typeof** input === 'number') { wystarczy typow. parametrów i zwrotu

**if** (input >= currentYear) **return**;

**return** currentYear - input;

}

**if** (**typeof** input === 'string' && /^[0-9]{4}$/.**test**(input)) {

**if** (Number(input) >= currentYear) **return**;

**return** currentYear - Number(input);

}

//gdy wpisze blednie np. getMyAge(new Date(2100, "a")) to wyskoczy  
 NaN w consoli, dlatego musze  
 jeszcze zwalidowac zeby sie upewnic ze  
 podana data jest dobrego formatu.   
 W zwyklej JS dziala !isNaN(input), a tu  
 jedynie sprawdze typeof ...getFullYear

**if** (input **instanceof** Date) {

**if** (**typeof** input.**getFullYear**() !== 'function') **return**;

**if** (input.**getFullYear**() >= currentYear) **return**;

**return** currentYear - input.**getFullYear**();

}

**return**;}

**ZAD 3. generateArrayWithRandomNumbers** Rób funkcję, która zwróci Arr o długości podanej jako „howMany-Numbers”. Arr zawiera losowe liczby z zakresu min i max. Rob drógą funkcje, która zwroci Arr z arrayami z poprzedniej funkcji.

**function** generateArrayWithRandomNumbers (howManyNumbers: number = 10, **min**: number = 1, **max**: number = 10): number [ ] {

**const** arr: number [ ] = [ ];

**for** (**let** i = 1; i <= howManyNumbers; i++) {

arr.push(Math.**floor**(Math.**random**() \* (**max** - **min** + 1) + **min**))

}

**return** arr;

}

type GArrayOfArrays = (howManyArrays?: number, howManyNumbers?: number, **min**?: number, **max**?: number) => number[][];

//każdy opcjonalny z ? bo ma defaultowa wartość podana w funkcji

**const** generateArrayOfArrays: GArrayOfArrays=**function** (howManyArrays: number = 10, howManyNumbers: number = 10, **min**: number = 1, **max**: number = 10): number[ ][ ] {

**const** arrOfArr: number[][] = [];

**for** (**let** k = 1; k <= howManyArrays; k++) arrOfArr.push(generateArrayWithRandomNumbers(howManyNumbers, **min**, **max**));

**return** arrOfArr;

}

**ZAD 4. Pagination** Rób funkcję w 1arg tablicę, a w 2 arg obiekt settings. Zwraca entriesOnSelectedPage, który jest arrayem podzielonym według ustawień z settings.---> „actualPageIndex” - numer strony(index wybranej strony(indexujemy od 0))

---> „entriesOnPage” �" ilośc obiektów na jedne stronie(maks.ilość elem zwracana z dataEntries dla wybranej strny)

**const** data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]; **const** settings = { actualPageIdx: 1, entriesOnPage: 2 };

**interface** Settings { //można tez uzyc aliasu,bez roznicy

actualPageIdx: number;

entriesOnPage: number;

}

type PaginateArray = (dataEntries: number[ ], settings: Settings) => number[ ] | **void**;

**const** paginateArray: PaginateArray = ((dataEntries: number[ ], settings: Settings) => number[ ] | **void**; => {

**if** (!Array.**isArray**(dataEntries)) **return**;

**if** (!dataEntries.**every**(el => **typeof** el == 'number')) **return**;

**if** (!dataEntries.**every**(el => Number.isInteger(el))) **return**;

**if** (settings.entriesOnPage > dataEntries.**length**) **return**;

**if** (settings.actualPageIdx < 0 || settings.entriesOnPage < 0) **return**;

**if** (settings.actualPageIdx \* settings.entriesOnPage >= dataEntries.**length**) **return**;

**const** start = (settings.actualPageIdx \* settings.entriesOnPage);

**return** dataEntries.**slice**(start, start + settings.entriesOnPage)

}

**ZAD 5. arrayMethods** Rób funkcje, uzywajac **for** lub **while** by zastapic: .forEach,.map,entries

.filter .reduce .every .some

**function** dolikeForEach<T>(array: T[ ], callbackFn: (el: T, i: number, array: T[ ]) => **void**): **void** {

**for** (**let** i = 0; i < array.**length**; i++) {

**// FOR EACH**

//Nic nie zwraca(tylko undefined) dlatego bez return, callback 3 parametry: element, jego index, array.

(callbackFn(array[i], i, array));

}

};

function makeBigLetters(element:string) { //callback

console.log(element.toUpperCase())

}

-----------------------------------------------------------------------------------------------------------------------------------------

**function** doLikeMap<T,U>(array: T[ ], callbackFn: (el: T, i: number, array: T[ ]) => U): U[ ] { //W Map można podac jeden typ, ale

**const** newArr: U[ ] = [ ]; w petli go zmienic i zwroci inny, dlatego tu uzywam

dwoch typow generycznych **!!!**

**for** (**let** i = 0; i < array.**length**; i++) newArr.push(callbackFn(array[i], i, array));

**// MAP**

//tworzy nowy Arr ze zmodyf. kazdym eleme. podanej tabl. array.map(function (currentValue, index, arr), thisValue)

**return** newArr;

}

function square(element: number) {

return element \* 2;

}

//----------------------------------------------------------------------------------------------------------------------------------------

**function** doLikeFilter<T>(arr: T[ ], callbackFn: (el: T, i: number, arr: T[ ]) => **boolean**): T[ ] {

**const** newArr: T[ ] = [ ];

**// FILTER**

//zwraca nowy Arr z el.spelniajacymi warunek. array.filter(function (currentValue, index, arr), thisValue)

**for** (**let** i = 0; i < arr.**length**; i++) {

**if** (callbackFn(arr[i], i, arr)) newArr.push(arr[i]);

}

**return** newArr;

}

function even(element: number) {

return (element % 2 === 0);

}

//----------------------------------------------------------------------------------------------------------------------------------------

**function** doLikeEvery<T>(array: T[ ], callbackFn: (el: T, i: number, array: T[ ]) => **boolean**): **boolean** {

**for** (**let** i = 0; i < array.**length**; i++)

**// EVERY**

//zwraca false,gdy choc 1 nie pasi. Inaczej da true. array.every(function(currentValue, index, arr), thisValue)

**if** (!callbackFn(array[i], i, array)) **return** false;

**return** true;

}

function isEven(element: number) {

return (element % 2 === 0);

}

//---------------------------------------------------------------------------------------------------------------------------------------

**function** doLikeSome<T>(array: T[ ], callbackFn: (elem: T, i: number, arr: T[ ]) => **boolean**): **boolean** {

**for** (**let** i = 0; i < array.**length**; i++) **if** (callbackFn(array[i], i, array)) **return** true;

**// SOME**

//Zwraca true, jesli min.1 el spelnia warunek.False jesli zaden. array.some(function(value, index, arr), this)

**return** false;

}

function isEven(element: number) {

return (element % 2 === 0);

}

//---------------------------------------------------------------------------------------------------------------------------------------

**function** doLikeEntries<T>(array: T[ ]): [number, T][ ] {

**// ENTRIES**

//zwraca tablice wyliczalnych par key-value. array.entries() brak parametrow

**let** result: [number, T][ ] = [ ];

**for** (**let** i = 0; i < array.**length**; i++) {

**const** key: number = i;

**const** value = array[i];

result.push( [key, value] );

}

**return** result;

}

//---------------------------------------------------------------------------------------------------------------------------------------

**function** doLikeReduce <T,U> (arr: T[ ], callbackFn: (acc: U, curr: T, currInd: number, arr: T[ ]) => U, initial?: U): U {

1. Initial podany w arg może być innego typu niż T,dlatego na jego potrzeby uzywamy tez U. Wtedy w zwrotce tez dajemy U, bo nawet gdyby T = U to nie będzie uwazane za blad. Z tego powodu te acc typujemy jako U, bo acc = initial.  
2.Ale gdy nie dam initiala to acc = arr[0](które ma typ T[ ]) to wyjdzie  
 blad, któremu zapobiegne wymuszając as unknown as U

**let** acc = initial ? initial : arr[0] as unknown as U

**for** (**let** i = ((initial) ? 0 : 1); i < arr.**length**; i++) {

acc = callbackFn(acc, arr[i], i, arr);

}

**return** acc;

}

[{name: 'Pawel', age: 23},{name: 'Arek', age: 22}].**reduce**((acc, el)=> {

**// REDUCE**

//Zwraca pojedyncza skumulowana wartosc. array.reduce(function (total, currentValue, currentIndex, arr), initialValue)

acc.push(el.age.**toString**())

**return** acc

}, [ ] as string[ ]) //przkladowa funkcja.gdzie initial jest innego typu

**ZAD 6. arrayMethods** Uzywajac .reduce zastap .map, .filter, .every, .some

**// MAP**

**function** doLikeMap<T, U>(arr: T[ ], callbackFn: (currVal: T, i: number, arr: T[ ]) => U): U[ ] {

**return** arr.**reduce**((acc: U[ ], currVal, i, arr) => { //przez to,ze map czasem zwraca inny typ niż przyjmie to uzywam tez U

acc.push(callbackFn(currVal, i, arr));

**return** acc;

}

, [ ])

}

function square(el) {

return el.toUpperCase();

}

-------------------------------------------------------------------------------------------------------

**// FILTER**

**function** doLikeFilter<T>(arr: T[ ], callbackFn: (currVal: T, i: number, arr: T[ ]) => **boolean**): T[ ] {

**return** arr.**reduce**((acc: T[ ], currVal, i, arr) => { //gdy nie otypuje tu acc to będzie myślał,ze never lub nie tabela

**if** (callbackFn(currVal, i, arr)) acc.push(currVal); //i nie będzie tolerowac push

**return** acc;

},

[ ])

}

function marek(el) {

return el[0] === el[0].toUpperCase();

}

**// EVERY**

**function** doLikeEvery<T>(arr: T[ ], callbackFn: (currVal: T, i: number, arr: T[ ]) => **boolean**): **boolean** {

**return** arr.**reduce**((acc, currVal, i, arr) => { // nie musze typowac acc, bo w wartosci początkowej\* już wnioskuje ze to boolean

**if** (!callbackFn(currVal, i, arr)) **return** acc = false;

**return** acc;

}, true); // \*

}

function czyWszystkieParzyste(el: number) {

return el % 2 == 0;

}

-------------------------------------------------------------------------------------------------------

**// SOME**

**function** doLikeSome<T>(arr: T[ ], callbackFn: (currVal: T, i: number, arr: T[ ]) => **boolean**): **boolean** {

**return** arr.**reduce**((acc, currVal, i, arr) => { // nie musze typowac acc, bo w wartosci początkowej\* już wnioskuje ze to boolean

**if** (callbackFn(currVal, i, arr)) **return** acc = true;

**return** acc;

}, false) //\*

}

function one(el:number) {

return el % 2 == 0;

}

**ZAD 7. aggregateArraysIntoChunks** twórz f. agregującą wszystkie elementy array na losowej długości chunki. Każdy chunk ma mieć 4-7 elementów, ostatni chunk też powinien być długości 4-7.

**function** aggregateIntoChunks<T>(arr: T[ ] ): T[ ][ ] | string {

**if** (arr.**length** < 4) **return** 'za krotki array';

**let** newArr: T[ ][ ] = [ ];

**const** **max** = 7;

**const** **min** = 4;

**let** sumaChunkow = 0;

**while** ((arr.**length** - sumaChunkow) > 11) {

**let** size = Math.**floor**(Math.**random**() \* (**max** - **min** + 1) + **min**);

newArr.push(arr.**slice**(sumaChunkow, sumaChunkow + size));

sumaChunkow += size;

}

**if** ((arr.**length** - sumaChunkow) <= 7) {

newArr.push(arr.**slice**(sumaChunkow,));

} **else** **if** ((arr.**length** - sumaChunkow) > 7 || (arr.**length** - sumaChunkow) <= 11) {

**let** warunek = false;

**while** (!warunek) {

**let** przedostatni = Math.**floor**(Math.**random**() \* (**max** - **min** + 1) + **min**);

**let** ostatni = (arr.**length** - sumaChunkow) - przedostatni;

**if** (ostatni >= 4 ?? ostatni <= 7 ?? przedostatni + ostatni === arr.**length** - sumaChunkow) {

newArr.push(arr.**slice**(sumaChunkow, sumaChunkow + przedostatni));

newArr.push(arr.**slice**(sumaChunkow + przedostatni,));

warunek = true;

}

}

}

**return** newArr;

}

Zad 8 next page🡪

**ZAD 8. filterWith** Korzystając z .filter twórz funkcję, która przeszuka tablicę obiektów. Zwraca tablicę z szukanym obj. Funkcja ma wykorzystywać RegExp., rekurencje do przeszukan zagnieżdżeń. 0-2 znaków w phrase zwraca [], pow. 2 ma filtrować.

**//Tu był spora tablica obiektów z różnymi typami zmiennych wielokrotnie zagnieżdżonych. Do wglądu w zadaniach w PC**

**function** filterWith(array: object[ ], phrase: string): object[ ] | [ ] {

**if** (/^.{0,2}$/.**test**(phrase)) **return** [ ];

**function** przeszukaj(array: object[], phrase: string): object[] {

**return** array.**filter**((el) => {

**if** (**typeof** el === 'string' || **typeof** el === 'number') {

**if** (el === phrase) **return** true;

}

**if** (**typeof** el === 'object') {

**const** rekurencja = przeszukaj(Object.values(el), phrase);

**if** (rekurencja.**length** > 0) {

**return** true;

}

}

**return** false;

})

}

**return** przeszukaj(array, phrase)

}

1. **OBJECT ORIENTED PROGRAMING**

**ZAD 1. dynamicSwitch** Stwórz klasę Switch, która służy do wielokrotnej, równorzędnej walidacji. Ma mieć metode .add w której dodajemy warunek do sprawdzenia oraz callback, który ma się wywołać jeśli warunek zostanie spełniony(wykona go w isValid). Ma mieć metodę .isValid która iteruje po wszystkich cases sprawdzając każdy dodany warunek. Metoda.isValid zwraca true jeśli wszystkie warunki będą na false. Jeżeli jakikolwiek warunek zostanie spełniony, funkcja przerywa swoje działanie, wywołując przekazany callback dla tego warunku.Po każdym wykonaniu metody warunki i callbacki są czyszczone. Klasa ma mieć metodę.isEmpty która sprawdza czy tablice cases i conditions są puste.Jeśli tak zwraca true

**interface** ISwitch {

cases: Array<{ condition: **boolean**, callbackFn: ( ) => **void** }>;

conditions: **boolean**[ ];

add: (condit: **boolean**, fn: ( ) => **void**) => **void**;

isEmpty: ( ) => **boolean**;

isValid: ( ) => **boolean**;

}

**class** Switch2 **implements** ISwitch {

cases: Array<{ condition: **boolean**, callbackFn: ( ) => **void** }> = [ ];

conditions: **boolean**[ ] = [ ];

constructor( ) { }

add = (condit: **boolean**, fn: () => **void**) => {

**this**.cases.push({ condition: condit, callbackFn: fn });

**this**.conditions.push(condit);

}

isEmpty(): **boolean** {

**if** (**this**.cases.**length** === 0 && **this**.conditions.**length** === 0) **return** true;

**return** false;

}

isValid(): **boolean** {

**if** (**this**.conditions.**every**((el => !el))) {

**this**.cases.**length** = 0;

**this**.conditions.**length** = 0;

**return** true;

}

**const** indexOfTrueCase = **this**.cases.findIndex(el => el.condition === true);

**if** (indexOfTrueCase !== -1) **this**.cases[indexOfTrueCase].callbackFn();

**this**.cases.**length** = 0;

**this**.conditions.**length** = 0;

**return** false;

}

}

**ZAD 2. dynamicSwitch** Stwórz strukturę danych związaną ze sklepem int.,(pełen opis się w kodzie niżej).Cd w plikach..

**interface** InterItem {

name: string;

uuid: number;

price: number;

category: string;

discountValue: number;

discountedPrice: number;

changeName: (newValue: string) => **void**;

changePrice: (newValue: number) => **void**;

setDiscountValue: (newValue: number) => **void**;

setCategory: (newValue: string) => **void**;

}

type wlasciwoscZmieniana = 'name' | 'price' | 'category' | "discountValue";

type Value = string | number;

**class** **Item** **implements** InterItem {

uuid: number;

name: string;

price: number;

discountValue: number;

discountedPrice: number;

category: string = '';

constructor(name: string, price: number, discountValue: number) {

**this**.uuid = Math.**floor**(Math.**random**() \* 1000000000000);

**this**.name = name;

**this**.price = price;

**this**.discountValue = discountValue;

**this**.discountedPrice = **this**.price - **this**.discountValue;

}

changeName = (newValue: string) => {

**this**.name = newValue;

}

changePrice = (newValue: number) => {

**this**.price = newValue;

**this**.discountedPrice = **this**.price - **this**.discountValue;

}

setDiscountValue(newValue: number) {

**this**.discountValue = newValue;

**this**.discountedPrice = **this**.price - **this**.discountValue;

}

setCategory(newValue: string) {

**this**.category = newValue;

}

}

**interface** InterCart {

uuid: number;

selectedProdsList: Array<{ prodInCart: Item, quantity: number }>;

discountCodeValue: number;

cartDiscountValue: number;

addProduct: (product: Item) => **void**;

removeProduct: (product: Item) => **void**;

changeQuantity: (product: Item, quantity: number) => **void**;

setDiscountCode: (discountCodeValue: number) => **void**;

setCartDiscount: (cartDiscountValue: number) => **void**;

countCartValue: ( ) => number;

}

**class** Cart implents InterCart {

uuid: number;

selectedProdsList: Array<{ prodInCart: Item, quantity: number }> = [ ];

discountCodeValue: number = 0;

cartDiscountValue: number = 0;

name: string = '';

age: number = 0

constructor( ) {

**this**.uuid = Math.**floor**(Math.**random**() \* 1000000000000);

}

addProduct(product: Item) { //product jest typem classy Item. Tak. Class tez może być wzorem typu☺

**const** indexToAdd = **this**.selectedProdsList.findIndex(e => e.prodInCart.uuid === product.uuid);

**if** (indexToAdd !== -1) **this**.selectedProdsList[indexToAdd].quantity++;

**else** **this**.selectedProdsList.push({ prodInCart: product, quantity: 1 });

}

removeProduct(product: Item) {

**const** indexToRemove = **this**.selectedProdsList.findIndex(e => e.prodInCart.uuid === product.uuid);

**if** (indexToRemove !== -1) **this**.selectedProdsList.splice(indexToRemove, 1);

}

changeQuantity(product: Item, quantity: number) {

**const** indexToChange = **this**.selectedProdsList.findIndex(e => e.prodInCart.uuid === product.uuid);

**if** (indexToChange === -1) **return**;

**if** (quantity === 0) **return** **this**.selectedProdsList.splice(indexToChange, 1);

**if** (indexToChange !== -1) **return** **this**.selectedProdsList[indexToChange].quantity = quantity;

}

setDiscountCode(discountCodeValue: number) {

**this**.discountCodeValue = discountCodeValue;

}

setCartDiscount(cartDiscountValue: number) {

**this**.cartDiscountValue = cartDiscountValue;

}

countCartValue( ) {

**return** ((**this**.selectedProdsList.**map**(e => e.quantity \* e.prodInCart.discountedPrice).**reduce // …**

((acc, prev) => acc + prev)) – (**this**.discountCodeValue + **this**.cartDiscountValue));

}

}

**ZAD 3. addressBook**  Stwórz strukturę danych związaną z biblioteką: *class Contact* { Ma mieć: Imie, Nazwisko, email, datę modyf, uuid. Umożliwia: aktualiz datę mody, zmiane imienia, nazwiska, email}. *class Group* {Ma mieć: listę kontaktów, nazwę grupy, uuid. Umożliwia: zmianę nazwy grupy, dodanie/usun kontakt z grupy, sprawdzenie czy kontakt istnieje w grupie}. *class AddressBook* {Ma mieć: listę all kontakt, listę grup. Umożliwia: szukanie kontaktu po frazie, dodaw/usuw/zmian nowych kontaktów i grup.}

**interface** IContact{

name: string,

lastName: string,

email: string

modificationDate: Date;

uuid: number;

setName: (param: string) => **void**;

setLastName: (param: string) => **void**;

setEmail: (param: string) => **void**;

}

**class** Contact **implements** IContact {

name: string;

lastName: string;

email: string;

modificationDate: Date;

uuid: number;

constructor(name: string, lastName: string, email: string) {

**this**.name = name;

**this**.lastName = lastName;

**this**.email = email;

**this**.modificationDate = **new** Date();

**this**.uuid = Math.**floor**(Math.**random**() \* 1000000);

}

setName(newName: string) {

**this**.name = newName;

**this**.modificationDate = **new** Date();

}

setLastName(newLastName: string) {

**this**.lastName = newLastName;

**this**.modificationDate = **new** Date();

}

setEmail(newEmail: string) {

**this**.email = newEmail;

**this**.modificationDate = **new** Date();

}

}

**interface** IGroup {

contactList: Array<Contact>;

name: string;

uuid: number;

setName: (param: string) => **void**;

addContact: (param: Contact) => **void**;

removeContact: (contactName: string, contactLastName: string) => **void**;

isInGroup: (contact: Contact) => **boolean**;

}

**class** Group **implements** IGroup {

contactList: Array<Contact> = [ ];

name: string;

uuid: number;

constructor(name: string) {

**this**.name = name;

**this**.uuid = Math.**floor**(Math.**random**() \* 1000000000);

}

setName(newName: string) {

**this**.name = newName;

}

addContact(contact: Contact) {

**this**.contactList.push(contact);

}

removeContact(contactName: string, contactLastName: string) {

**const** indexToRemove = **this**.contactList.findIndex(e => e.name === contactName && e.lastName === contactLastName);

**if** (indexToRemove !== -1) **this**.contactList.splice(indexToRemove, 1);

}

isInGroup(contact: Contact) {

**if** (**this**.contactList.find(e => e.uuid === contact.uuid) !== undefined) **return** true;

**return** false;

}

}

**class** Validator {

**static** isValidContactName(name: string) {

**return** ((/^[a-z]{2,}$/**i**.**test**(name)))

}

**static** isValidEmail(email: string) {

**return** (/^[a-z\d]+[-\w\.]\*@[a-z\d]+([-\w]+\.)[a-z]{2,6}$/**i**.**test**(email));

}

**static** isValidGroupName(name: string) {

**return** (/^.{3,}$/**i**.**test**(name));

}

}

**interface** IAddressBook {

contacts: Array<Contact>;

groups: Array<Group>;

findGroupIndex: (groupName: string) => number;

findContactIndex: (contactName: string, contactLastName: string) => number;

createContact: (name: string, lastName: string, email: string) => **void**;

createGroup: (name: string) => **void**;

deleteContact: (contactName: string, contactLastName: string) => **void**;

deleteGroup: (name: string) => **void**;

addContactToGroup: (contactName: string, contactLastName: string, groupName: string) => **void**;

deleteContactFromGroup: (contactName: string, contactLastName: string, groupName: string) => **void**;

findContact: (phrase: string) => Contact[ ];

changeContactName: (currName: string, currLastName: string, newName: string, newLastName: string) => **void**;

changeContactEmail: (contactName: string, contactLastName: string, newEmail: string) => **void**;

changeGroupName: (currName: string, newName: string) => **void**;

}

**class** AddressBook **implements** IAddressBook {

contacts: Array<Contact> = [ ];

groups: Array<Group> = [ ];

findGroupIndex(groupName: string): number {

**return** **this**.groups.findIndex(e => e.name.**toLowerCase**() === groupName.**toLowerCase**());

}

findContactIndex(contactName: string, contactLastName: string): number {

**return** **this**.contacts.findIndex(e => e.name.**toLowerCase**() === contactName.**toLowerCase**() &&

e.lastName.**toLowerCase**() === contactLastName.**toLowerCase**());

}

createContact(name: string, lastName: string, email: string) {

**if** (!Validator.isValidContactName(name)) **return**;

**if** (!Validator.isValidContactName(lastName)) **return**;

**if** (!Validator.isValidEmail(email)) **return**;

**const** contactInstance = **new** Contact(name, lastName, email);

**this**.contacts.push(contactInstance);

}

createGroup(name: string) {

**if** (!Validator.isValidGroupName(name)) **return**;

**if** (**this**.findGroupIndex(name) !== -1) **return**;

**const** groupInstance = **new** Group(name);

**this**.groups.push(groupInstance)

}

deleteContact(contactName: string, contactLastName: string) {

**const** contactIndex = **this**.findContactIndex(contactName, contactLastName);

**if** (contactIndex === -1) **return**;

**this**.contacts.splice(contactIndex, 1);

**const** groupIndex = **this**.groups.findIndex(e => e.contactList.find(e => e.name.**toLowerCase**() ===

contactName.**toLowerCase**() && e.lastName.**toLowerCase**() === contactLastName.**toLowerCase**()));

**if** (groupIndex === -1) **return**;

**const** groupName = **this**.groups[groupIndex].name;

**this**.deleteContactFromGroup(contactName, contactLastName, groupName);

}

deleteGroup(name: string) {

**this**.groups.splice(**this**.findGroupIndex(name), 1);

}

addContactToGroup(contactName: string, contactLastName: string, groupName: string) {

**this**.groups[**this**.findGroupIndex(groupName)].contactList.push(**this**.contacts[**this**.

findContactIndex(contactName, contactLastName)]);

}

deleteContactFromGroup(contactName: string, contactLastName: string, groupName: string) {

**const** contactIndexInGroup = **this**.groups[**this**.findGroupIndex(groupName)].contactList.findIndex(e =>

e.name.**toLowerCase**() === contactName.**toLowerCase**() && e.lastName.**toLowerCase**() ===

contactLastName.**toLowerCase**());

**if** (contactIndexInGroup !== -1) **this**.groups[**this**.findGroupIndex(groupName)].contactList.splice(contactIndexInGroup, 1);

}

findContact(phrase: string): Contact[ ] {

**return** **this**.contacts.**filter**(e => {

**if** (Object.values(e).**toString**().**toLowerCase**().includes(phrase.**toLowerCase**())) **return** true;

**return** false;

});

}

changeContactName(currName: string, currLastName: string, newName: string, newLastName: string) {

**if** (!Validator.isValidContactName(newName)) **return**;

**if** (!Validator.isValidContactName(newLastName)) **return**;

**const** contactIndex = **this**.findContactIndex(currName, currLastName);

**this**.contacts[contactIndex].setName(newName);

**this**.contacts[contactIndex].setLastName(newLastName);

}

changeContactEmail(contactName: string, contactLastName: string, newEmail: string) {

**if** (!Validator.isValidEmail(newEmail)) **return**;

**this**.contacts[**this**.findContactIndex(contactName, contactLastName)].setEmail(newEmail);

}

changeGroupName(currName: string, newName: string) {

**if** (!Validator.isValidGroupName(newName)) **return**;

**this**.groups[**this**.findGroupIndex(currName)].setName(newName)

}

}

**ZAD 4. email**  Wykorzystując wzorzec proj. Builder stwórz obiekt json reprezentujacy wszystkie parametry maila (parametry znajdują sie w kodzie poniżej) tresc Cd w plikach.

Typowanie było banalne, same string/number array wiec do wglądu w pliku

**ZAD 5. user**  Stwórz 2 klasy: **User**: ma 2poz: normal i admin, umożliwia zmianę hasła, email i poziomu. User z poz admin może zmieniać właśc. Innych userów.**App**: ma listę userów, tworzy nowych userow. Zrób klase **Validator**, co ma met statyczne walidujace usera. Gdy walidacja się nie powiedzie, instancja ma nie być tworzona, tylko zwraca error. (hasło min 8 znak,1 wielką,1 cyfrę, 1 znak specjalny, data (nieważne jaka wejdzie) do konstruktora musi wejść w formacie MM/DD/YYYY 2023/04/13, imię/lastName niepuste

**//class** Validator był banalny sam string/number wiec do wglądu w plikach

**interface** IUser {

name: string,

lastName: string,

birthDate: string,

password: string,

sex: 'male' | 'female',

email: string,

accessLevel: 'user' | 'admin',

formatBirthDate: (birthDate: string) => string;

setPassword: (newPassword: string) => **void**;

setEmail: (newEmail: string) => **void**;

setAccessLevel: (newAccessLevel: 'user' | 'admin') => **void**;

}

**class** User implements IUser {

name: string;

lastName: string;

birthDate: string;

password: string;

sex: 'male' | 'female';

email: string;

accessLevel: 'user' | 'admin';

constructor(name: string, lastName: string, birthDate: string, password: string, sex: 'male' | 'female', email: string, accessLevel: 'user' | 'admin') {

**if** (!Validator.isNameValid(name)) **throw** **new** Error('niewlasciwe name');

**if** (!Validator.isNameValid(lastName)) **throw** **new** Error('niewlascie lastName');

**if** (!Validator.isEmailValid(email)) **throw** **new** Error('niewlasciwy email');

**if** (!Validator.isSexValid(sex)) **throw** **new** Error('niewlasciwa sex');

**if** (!Validator.isPasswordValid(password)) **throw** **new** Error('niewlasciwe password');

**if** (!Validator.isAccessLevelValid(accessLevel)) **throw** **new** Error('niewlasciwy poziom')

**if** (!Validator.isBirthDateValid(birthDate)) **throw** **new** Error('zla data uro');

**this**.name = name;

**this**.lastName = lastName;

**this**.birthDate = **this**.formatBirthDate(birthDate);

**this**.password = password;

**this**.sex = sex;

**this**.email = email;

**this**.accessLevel = accessLevel;

}

formatBirthDate(birthDate: string) {

**const** regex = /^(1[0-2]|0?[1-9])\/(3[01]|[12][0-9]|0?[1-9])\/[0-9]{2}?([0-9]{2})$/;

**const** birthDateInNewFormat = birthDate.**replace**(regex, "$1/$2/$3")

**return** birthDateInNewFormat;

}

setPassword(newPassword: string) {

**if** (!Validator.isPasswordValid(newPassword)) **throw** **new** Error('niewlasciwe password');

**this**.password = newPassword;

}

setEmail(newEmail: string) {

**if** (!Validator.isEmailValid(newEmail)) **throw** **new** Error('niewlasciwy email');

**this**.email = newEmail;

}

setAccessLevel(newAccessLevel: 'user' | 'admin') {

**if** (!Validator.isAccessLevelValid(newAccessLevel)) **throw** **new** Error('niewlasciwy poziom')

**this**.accessLevel = newAccessLevel;

}

}

**interface** IApp {

listOfUsers: User[ ];

createUser: (name: string, lastName: string, birthDate: string, password: string, sex: 'male' | 'female', email: string) => **void**;

createAdmin: (name: string, lastName: string, birthDate: string, password: string, sex: 'male' | 'female', email: string) => **void**;

findUserOrAdminIndex: (email: string) => number;

checkAccessLevel: (email: string) => number;

setPassword: (adminEmail: string, userEmail: string, newPassword: string) => **void**;

setEmail: (adminEmail: string, userEmail: string, newEmail: string) => **void**;

setAccessLevel:(adminEmail: string, userEmail: string, newAccessLevel: 'user' | 'admin')=> **void**;

}

**class** App implements IApp{

listOfUsers: User[] = [];

createUser(name: string, lastName: string, birthDate: string, password: string, sex: 'male' | 'female', email: string) {

**try** {

**const** userInstance = **new** User(name, lastName, birthDate, password, sex, email, 'user');

**this**.listOfUsers.push(userInstance)

} **catch** (error) {

console.**log**(error)

**return**

}

}

createAdmin(name: string, lastName: string, birthDate: string, password: string, sex: 'male' | 'female', email: string) {

**try** {

**const** userInstance = **new** User(name, lastName, birthDate, password, sex, email, "admin");

**this**.listOfUsers.push(userInstance)

} **catch** (error) {

console.**log**(error)

**return**

}

}

findUserOrAdminIndex(email: string) {

**return** **this**.listOfUsers.findIndex(e => e.email.**toLowerCase**() === email.**toLowerCase**());

};

checkAccessLevel(email: string) {

**const** indexUser = **this**.findUserOrAdminIndex(email);

**if** (indexUser === -1) **return** -1;

**if** (**this**.listOfUsers[indexUser].accessLevel === 'user') **return** -1;

**return** indexUser;

}

setPassword(adminEmail: string, userEmail: string, newPassword: string) {

**try** {

**if** (**this**.checkAccessLevel(adminEmail) === -1) **return**;//brak tego usera/uprawnien

**if** (**this**.findUserOrAdminIndex(userEmail) === -1) **return**;

**this**.listOfUsers[**this**.findUserOrAdminIndex(userEmail)].setPassword(newPassword);

} **catch** (error) {

console.**log**(error)

**return**;

}

}

setEmail(adminEmail: string, userEmail: string, newEmail: string) {

**try** {

**if** (**this**.checkAccessLevel(adminEmail) === -1) **return**;

**if** (**this**.findUserOrAdminIndex(userEmail) === -1) **return**;

**this**.listOfUsers[**this**.findUserOrAdminIndex(userEmail)].setEmail(newEmail);

} **catch** (error) {

console.**log**(error)

**return**;

}

}

setAccessLevel(adminEmail: string, userEmail: string, newAccessLevel: 'user'|'admin') {

**try** {

**if** (**this**.checkAccessLevel(adminEmail) === -1) **return**;

**if** (**this**.findUserOrAdminIndex(userEmail) === -1) **return**;

**this**.listOfUsers[**this**.findUserOrAdminIndex(userEmail)].setAccessLevel(newAccessLevel);

} **catch** (error) {

console.**log**(error)

**return**

}

}

}

**ZAD 6. user**  Stwórz strukturę danych związaną z biblioteką, pełen opis znajduję się w plikach

**class** User **implements** IUser {

**interface** IUser {

name: string;

lastName: string;

uuid: number;

}

name: string;

lastName: string;

uuid: number;

constructor(name: string, lastName: string) {

**this**.name = name;

**this**.lastName = lastName;

**this**.uuid = Math.**floor**(Math.**random**() \* 1000000)

}

}

**interface** IBook {

title: string;

author: string;

uuid: number;

picture: string;

description: string;

;

}

**class** Book **implements** IBook {

title: string;

author: string;

uuid: number;

picture: string;

description: string;

constructor(title: string, author: string, picture: string, description: string) {

**this**.title = title;

**this**.author = author;

**interface** IBooking {

borrowedBookList: Book[ ] = [ ];

user: User; //interface lub klasa

borrowDate: Date;

maxReturnDate: Date;

chargeAmount: number;

chargeSum: number;

findBookIndex: (bookTitle: string) => number;

addBookToBorrowedList: (book: Book) => **void**;

deleteBookFromBorrowedList: (bookTitle: string) => **void**;

countCharge: (actualReturnDate: number) => **void**;

returnBook: (bookTitle: string) => **void**;

}

**this**.uuid = Math.**floor**(Math.**random**() \* 1000000);

**this**.picture = picture;

**this**.description = description;

}

}

**class** Booking **implements** IBooking {

borrowedBookList: Book[ ] = [ ];

user: User; //interface lub klasa

borrowDate: Date;

maxReturnDate: Date;

chargeAmount: number;

chargeSum: number;

constructor(userName: string, userLastName: string) {

**this**.user = **new** User(userName, userLastName)

**this**.borrowDate = **new** Date();

**this**.maxReturnDate = **new** Date(**this**.borrowDate.**getTime**() + 7 \* 24 \* 60 \* 60 \* 1000);

**this**.chargeAmount = 5;

**this**.chargeSum = 0;

}

findBookIndex(bookTitle: string) {

**return** **this**.borrowedBookList.findIndex(e => e.title.**toLowerCase**() === bookTitle.**toLowerCase**());

}

addBookToBorrowedList(book: Book) {

**this**.borrowedBookList.push(book);

}

deleteBookFromBorrowedList(bookTitle: string) {

**if** (**this**.findBookIndex(bookTitle) === -1) **return**;

**this**.borrowedBookList.splice(**this**.findBookIndex(bookTitle), 1)

}

countCharge(actualReturnDate: number) {

**const** timeDifference = (**this**.maxReturnDate.**getTime**()) - actualReturnDate;

**if** (timeDifference >= 0) **return** console.**log**('no extra charge');

**const** timeDifferenceInDays = Math.**ceil**(timeDifference / (1000 \* 60 \* 60 \* 24));

**return** **this**.chargeSum += Math.**abs**(**this**.chargeAmount \* timeDifferenceInDays);

}

type Ksiazka = { book: Book, quantity: number };

**interface** ILibrary {

allLibraryBookList: Ksiazka[ ];

availableBookList: Ksiazka[ ];

bookingsList: Booking[ ];

userList: User[ ];

findBookIndex: (bookTitle: string, list: Ksiazka[]) => number;

addBookToLibrary: (book: Book) => **void**;

addBookToAvailableList: (bookTitle: string) => **void**;

deleteBookFromLibrary: (bookTitle: string) => **void**;

deleteBookFromAvailableList: (bookTitle: string) => **void**;

makeBooking: (userName: string, userLastName: string, bookTitle:

string) => **void**;

returnBooking: (userName: string, userLastName: string, bookTitle:

string) => **void**;}

returnBook(bookTitle: string) {

**if** (**this**.findBookIndex(bookTitle) === -1) **return**;

**this**.deleteBookFromBorrowedList(bookTitle);

**const** actualReturnDate = (**new** Date()).**getTime**();

**const** charge = **this**.countCharge(actualReturnDate);

**if** (charge === undefined) **return**;

**return** console.**log**(`Extra charge is ${charge} pln`);

}

}

**class** Library **implements** ILibrary {

allLibraryBookList: Ksiazka[] = [];

availableBookList: Ksiazka[] = [];

bookingsList: Booking[] = [];

userList: User[] = [];

findBookIndex(bookTitle: string, list: Ksiazka[]) {

**return** list.findIndex(e => e.book.title.**toLowerCase**() === bookTitle.**toLowerCase**());

}

addBookToLibrary(book: Book) {

**const** bookIndex = **this**.findBookIndex(book.title, **this**.allLibraryBookList);

**if** (bookIndex === -1) {

**this**.allLibraryBookList.push({ book, quantity: 1 });

**this**.availableBookList.push({ book, quantity: 1 });

**return**;

}

**this**.allLibraryBookList[bookIndex].quantity++;

**this**.addBookToAvailableList(book.title);

}

addBookToAvailableList(bookTitle: string) {

**const** bookIndexInAvailables = **this**.findBookIndex(bookTitle, **this**.availableBookList);

**if** (bookIndexInAvailables !== -1) {

**this**.availableBookList[bookIndexInAvailables].quantity++

**return**

};

**const** bookIndexInAllbooks = **this**.findBookIndex(bookTitle, **this**.allLibraryBookList);

**if** (bookIndexInAllbooks === -1) **return** console.**log**('nie ma takiej ksiazki w bibliotece');

**this**.availableBookList.push(**this**.allLibraryBookList[bookIndexInAllbooks])

}

deleteBookFromLibrary(bookTitle: string) {

**const** bookIndexToDelete = **this**.findBookIndex(bookTitle, **this**.allLibraryBookList);

**if** (bookIndexToDelete === -1) **return**;

**if** (**this**.allLibraryBookList[bookIndexToDelete].quantity === 1) {

**this**.allLibraryBookList.splice(bookIndexToDelete, 1);

**this**.deleteBookFromAvailableList(bookTitle);

**return**;

}

**this**.allLibraryBookList[bookIndexToDelete].quantity--;

**this**.deleteBookFromAvailableList(bookTitle);

}

deleteBookFromAvailableList(bookTitle: string) {

**const** bookIndexToDelete = **this**.findBookIndex(bookTitle, **this**.availableBookList);

**if** (bookIndexToDelete === -1) **return**;

**if** (**this**.availableBookList[bookIndexToDelete].quantity === 1) {

**this**.availableBookList.splice(bookIndexToDelete, 1);

**return**;

}

**this**.availableBookList[bookIndexToDelete].quantity--;

}

makeBooking(userName: string, userLastName: string, bookTitle: string) {

**const** bookIndexInAvailablelList = **this**.findBookIndex(bookTitle, **this**.availableBookList);

**if** (bookIndexInAvailablelList === -1) **return** console.**log**("book unavailable- booking canceled");

**const** newBooking = **new** Booking(userName, userLastName);

**this**.userList.push(newBooking.user);

newBooking.addBookToBorrowedList(**this**.availableBookList[bookIndexInAvailablelList].book);

**this**.bookingsList.push(newBooking)

**if** (**this**.availableBookList[bookIndexInAvailablelList].quantity === 1) {

**this**.availableBookList.splice(bookIndexInAvailablelList, 1);

**return**;

}

**this**.availableBookList[bookIndexInAvailablelList].quantity--;

}

returnBooking(userName: string, userLastName: string, bookTitle: string) {

**const** findBookingIndex = **this**.bookingsList.findIndex(e =>(e.user.name.**toLowerCase**() === userName.**toLowerCase**()) &&

(e.user.lastName.**toLowerCase**() === userLastName.**toLowerCase**()) &&

e.borrowedBookList.find(f => f.title.**toLowerCase**() === bookTitle.**toLowerCase**()))

**if** (findBookingIndex === -1) **return** console.**log**('booking doesn\'t exist');

**this**.bookingsList[findBookingIndex].returnBook(bookTitle)

**this**.userList.splice(**this**.userList.findIndex(e => e.name.**toLowerCase**() === userName.**toLowerCase**() &&

e.lastName.**toLowerCase**() === userLastName.**toLowerCase**()), 1) //usuwam usera z listy userow

**this**.availableBookList.push(**this**.allLibraryBookList[**this**.findBookIndex(bookTitle, **this**.allLibraryBookList)])

**this**.bookingsList.splice(findBookingIndex, 1)

}

}