**OOP Programowanie obiektowe**

**Zad. 1** dynamicSwitch

1. Stwórz klasę Switch, która służy do wielokrotnej, równorzędnej walidacji. Klasa ma mieć metode .add w której dodajemy warunek do sprawdzenia oraz callback, który ma się wywołać jeśli warunek zostanie spełniony (wykona go w isValid). Klasa ma mieć metodę .isValid która iteruje po wszystkich cases sprawdzając każdy dodany wcześniej 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

**class** Switch2 {

cases = [ ]; //warunki i callbacki

conditions = [ ]; //warunki. Ten arr niepotrzebny,moglbym wszystko robic tylko na cases, ale kazali:)

add(warunek, callbackFn) {

**this**.cases.push({ warun: warunek, callFn: callbackFn });

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

}

isValid( ) {

**if** (**this**.conditions.**every**((el => !el))) { //spr czy all war sa niespelnione

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

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

console.**log**('isValid', true) //pogladowe.(do usuniecia)

**return** true;

}

**for** (**let** i = 0; i < **this**.cases.**length**; i++) { //spr który warun jest spelniony

**if** (**this**.cases[i].warun === true) {

**let** j = i;

i = **this**.cases.**length**; //ustawiam by dalej nie robil kolejnej petli gdy znajdzie(jak break)

**this**.cases[j].callFn(); //return? wtedy nie zeruje arrejow

}

}

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

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

**return** false

}

isEmpty() {

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

**return** false;

}

}

**const** test1 = **new** Switch2();

**const** value2 = "test";

test1.add(value2.**length** < 3, () => {

console.**log**('Warunek jest spelniony'); // false

})

test1.add((**typeof** (value2) !== "string"), () => {

console.**log**('Warunek jest spelniony'); // false

})

test1.add((**typeof** (value2) === "string"), () => {

console.**log**('Warunek 3 nie jest spelniony'); // true

})

test1.add((**typeof** (value2) !== "string"), () => {

console.**log**('Warunek 4 nie jest spelniony'); // false

})

test1.isValid();

**Zad. 2** cart

1.Stwórz strukturę danych związaną ze sklepem int.,(pełen opis się w kodzie niżej).

2. Wypracuj obiekt charakteryzujący przedmiot. 3. Wypracuj obiekt charakteryzujący koszyk.

class CartItem {

Ma miec: Nazwę, Kategorię, Cenę, Rabat % na przedmiot, uuid

Ma umożliwiać: określanie rabatu procentowego, dodawanie produkt do kat., zmianę nazwy, ceny lub rabatu}

class Cart {

Ma mieć: uuid, listę wybranych przedmiotów, rabat % na koszyk, kod rabatowy

Ma umożliwiać: dodawanie/usuw. itemu do/z koszyka, zmianę ilości itemu w koszyku, podliczać wartość koszyka z rabatami }

**class** Item {

category = '';

discountValue = 0;

constructor(name, price) {

**this**.name = (**typeof** name === 'string') ? name : error('Zly typ name!');

**this**.price = (**typeof** price === 'number') ? price : error('Zly format price');

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

**this**.discountedPrice = price;

}

changeName(newName) {

**this**.name = newName;

}

changePrice(newPrice) {

**this**.price = newPrice;

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

}

setCategory(category) {

**this**.category = category;

}

setProdDiscount(discountValue) {

**this**.discountValue = discountValue;

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

}

}

**const** przedmiot1 = **new** Item('Wiadro z kijem', 20);

przedmiot1.setProdDiscount(2)

przedmiot1.changePrice(25)

**class** Cart {

selectedProdsList = []

discountCodeValue = 0;

cartDiscountValue = 0;

constructor() {

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

}

addProduct(product) {

**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) {

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

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

// bez " else this.selectedProdsList[indexToRemove].quantity--; " bo od zmiany ilosci jest metoda chhangeQuantity a tu ma niewazne ile to ma usunac ten produkt"

}

changeQuantity(product, pcs) {

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

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

}

setDiscountCode(discountCodeValue) {

**this**.discountCodeValue = discountCodeValue;

}

setCartDiscount(cartDiscountValue) {

**this**.cartDiscountValue = cartDiscountValue;

}

countCartValue() {

**return** ((**this**.selectedProdsList.**map**(e => e.quantity \* e.prodInCart.discountedPrice).**reduce**((acc, prev) => acc + prev)) - (**this**.discountCodeValue + **this**.cartDiscountValue));

}

}

**const** koszyk1 = **new** Cart()

koszyk1.addProduct(przedmiot1)

koszyk1.setCartDiscount(25)

koszyk1.setDiscountCode(20)

console.**log**(koszyk1.selectedProdsList[0])

console.**log**(koszyk1.countCartValue())

/\* Jak robic deklaracje zmiennych w klasach.

a)Jako wlasciwosc obiektu wspolna dla wszystkich - pisac bez let /const / this. TE ktore nie sa dynamiczne.

b) z this (w constructorze) pisac to co przychodzi z parametrem oraz jest dynamiczne (np. new Date, Math, i inne przypisania).

c) w funkcjach zmienne pomocnicze, np 'suma' juz piszemy z const/let\*/

**Zad. 3** addressBook

Stwórz strukturę danych związanych z biblioteką. Wypracuj obiekt contact, contactGroup, adressBook.

**class** Contact {

Ma mieć: Imie, Nazwisko, adres-emial, datę modyfikacji i utworzenia, uuid

Ma umożliwiać: aktualiz. datę modyfikacji, modyfikację imie,nazwiska,email }

**class** Group {

Ma mieć: listę kontaktów oraz nazwę grupy oraz uuid Ma umożliwiać: zmianę nazwy grupy, dod/usun kontakt z grup,sprawdzenie czy kontakt istnieje w grupie }

**class** AddressBook {

Ma mieć: listę all kontaktów, listę grup kontaktów Ma umożliwiać: szukanie kontaktu po frazie, dodaw/usuw/modyf nowych kontaktów i nowych grup}

**class Contact** {

constructor(name, lastName, email) {

**this**.name = name;

**this**.lastName = lastName;

**this**.email = email;

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

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

}

setName(newName) {

**this**.name = newName;

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

}

setLastName(newLastName) {

**this**.lastName = newLastName;

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

}

setEmail(newEmail) {

**this**.email = newEmail;

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

}

}

**class Group** {

contactList = [];

constructor(name) {

**this**.name = name;

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

}

setName(newName) {

**this**.name = newName;

}

addContact(contact) {

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

}

removeContact(contactName, contactLastName) {

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

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

}

isInGroup(contact) {

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

**return** false;

}

}

**class Validator** {

**static** isValidContactName(name) {

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

}

**static** isValidEmail(email) {

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

}

**static** isValidGroupName(name) {

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

}

}

**class AddressBook** {

contacts = [];

groups = [];

createContact(name, lastName, email) {

**if** (!Validator.isValidContactName(name)) **return** 'zla wiadomosc name';

**if** (!Validator.isValidContactName(lastName)) **return** 'zla wiadomosc lastName';

**if** (!Validator.isValidEmail(email)) **return** 'zla wartosc email';

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

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

}

findGroupIndex(groupName) {

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

}

findContactIndex(contactName, contactLastName) {

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

}

deleteContact(contactName, contactLastName) {

//gdyby bylo wiecej contactow o tym samym name i lastname warto szukac tez po email/uuid..

**this**.contacts.splice(**this**.findContactIndex(contactName, contactLastName), 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;

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

}

createGroup(name) {

**if** (!Validator.isValidGroupName(name)) **return** 'zla wartosc name';

**if** (**this**.findGroupIndex(name) !== -1) **return** 'grupa o tej nazwie juz istnieje';

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

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

}

deleteGroup(name) {

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

}

addContactToGroup(contactName, contactLastName, groupName) {

**this**.groups[**this**.findGroupIndex(groupName)].contactList.push(**this**.contacts[**this**.findContactIndex(contactName, contactLastName)]);

}

deleteContactFromGroup(contactName, contactLastName, groupName) {

**const** contactIndexInGroup = **this**.groups[**this**.findGroupIndex(groupName)].contactList.findIndex(e => e.name.**toLowerCase**() === contactName.**toLowerCase**() && e.lastName.**toLowerCase**() === contactLastName.**toLowerCase**());

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

}

findContact(phrase) {

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

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

**return** false;

});

}

changeContactName(currName, currLastName, newName, newLastName) {

**if** (!Validator.isValidContactName(newName)) **return** 'zla wartosc name';

**if** (!Validator.isValidContactName(newLastName)) **return** 'zla wartosc lastName';

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

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

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

//musialem uzyc zmiennej z indx poniewaz po wykonaniu linijki setName juz porownanie sie nie zgadzalo, bo imie bylo zmienione, wiec dzialalo tylko dla jednego z nich

}

changeContactEmail(contactName, contactLastName, newEmail) {

**if** (!Validator.isValidEmail(newEmail)) **return** 'zla wartosc email';

**this**.contacts[**this**.findContactIndex(contactName, contactLastName)].setEmail(newEmail);//setEmail z cl.Contact

}

changeGroupName(currName, newName) {

**if** (!Validator.isValidGroupName(newName)) **return** 'zla wartosc name';

**this**.groups[**this**.findGroupIndex(currName)].setName(newName)//setName z cl.Contact

}

}

**const** book1 = **new** AddressBook();

book1.createContact('Angelika', 'Berkowska', 'angee@gmail.com');

book1.createContact('Marek', "Nowak", 'm.nowak@op.pl');

book1.createGroup('pierwsza');

book1.createGroup('druga');

book1.addContactToGroup('Angelika', 'Berkowska', 'pierwsza');

book1.addContactToGroup('Marek', "Nowak", ' druga ');

// 1. Bez constructora w addressbook? Tak. moze byc classa bez constructora

// 2. Jesli modyfikujemy obiekty stworzone w oparciu o inne klasy, (np w classie AdressBook tworzymy funkcją createContact kontakt, na podstawie klasy Contact), to najlepiej modyfikowac je metodami z tamtej klasy Contact. O ile je posiadaja. jesli nie to pisac nowe. Wtedy tez np. odpowiednio zmienia sie data modyfikacji, a robiac to manualnie piszac nowe metody nie korzystajac z metod z Contact data sie nie zmienia.

//3. classa ponoc ma sie zajmowac 1 odpowiedzuialnoscia. ja mam giganta! ale to nie szkodzi. Ona nadal ma jedna odpowiadzalnosc, czyli zarzadzanie kontaktami solo i w grupach. jedynie walidacje mozna dac do jakies osobnej klassy z statycznymi metodami.

//4. Walidacje robimy tylko w tych metodach, ktore tworza nam obiekty. Tam jest istotne zeby podac w odpowiednim formacie nazwy,dane itd. W funkcjach, find/get/change nie potrzeba. Jak zmieni to zmieni, jak zle wpisze i nie znajdzie to najwyzej nie znajdziei tyle

//5. W wielu metodach w addresBook uzywam innej metody z tej samej klasy jak findContactIndex czy FindgroupIndex dzieki temu oszczedzam pisania kodu

//6. Nalezy pamietac, ze jesli uzywam wewnatrz klassy metody pochodzace tez z tej klasy to przed nimi nalezy rowniez pisac this!!!

**Zad. 4** emailBuilder

Wykorzystując wzorzec projektowy Builder stwórz obiekt json reprezentujacy wszystkie parametry maila, z metodami które będą zmieniać param. from, to, title, cc, bcc, html.

class EmailBuilder{

constructor() {

this.\_mail = {

from: "",

to: "",

title: "",

cc: [],

bcc: [],

html: "",

};

}

**class** Email {

constructor(from, to, title, cc, bcc, html) {

**this**.from = from;

**this**.to = to;

**this**.title = title;

**this**.cc = cc;

**this**.bcc = bcc;

**this**.html = html;

}

}

**class** EmailBuilder {

//warto tu tworzyc constructor z min.1 property(wtedy zwracamy obiekt, a nie classe)

setFrom(from) {

**this**.from = from;

**return** **this**;

}

setTo(to) {

**this**.to = to;

**return** **this**;

}

setTitle(title) {

**this**.title = title;

**return** **this**;

}

setCC(cc) {

**this**.cc = cc;

**return** **this**;

}

setBCC(bcc) {

**this**.bcc = bcc;

**return** **this**;

}

setHTML(html) {

**this**.html = html;

**return** **this**;

}

build() {

**return** **new** Email(**this**.from, **this**.to, **this**.title, **this**.cc, **this**.bcc, **this**.html);

}

//albo od razu zwrot w jsonie.

// return JSON.stringify(new Email(this.from, this.to, this.title, this.cc, this.bcc, this.html));

}

**const** email1 = **new** EmailBuilder().setFrom('jan@op.pl').setTo('ala@wp.pl').setCC([' marek@op.pl', 'ja@gg.pl']).build()

**const** emailWjsonie = JSON.**stringify**(email1)

// wyjasnienie jsona przez Pawla.

//JSON sluzy do zmodyfikowania kodu z js w stringi, by moglby byc odczytane rowniez przez inne aplikacje itd.

**Zad. 5** user Stwórz 2 klasy:

-**Klasa User** ma 2poziomy dostępu: normal i admin. Umożliwia zmianę hasła/ email / poziomu dost. User z poziomem "admin" może zmieniać hasła/email/poziomy innych userow.

**-Klasa App** zarządza relacjami między userami. Zawiera ich listę, tworzy nowych userow o różnych poziomach dostępu.

Stwórz też klase Validator, która będzie posiadała metody stat. Do walidacji usera. Jeżeli któraś z walidacji się nie powiedzie, instancja ma nie być tworzona, tylko ma zwracać error z odpowiednim komunikatem o niepowiedzionej walidacji.

W walidacji upewnij się, że: email jest poprawny,hasło ma min 8znaków, min 1wielką literę i min 1cyfrę i min 1znak specjalny, płeć = Male/female. A data (nieważne jaka wejdzie) do konstruktora musi wejść wformacie MM/DD/YYYY 2023/04/13,a imie/nazwwpisano

class User{

imię

lastName

datę urodzenia

hasło

płeć

adres email

accessLevel dostępu = "user" | "admin";

}

class App{

listOfUsers

createUser(...)

createAdmin(...)

wszystkie metody w których admin ingeruje we właściwości innych użytkowników

}

**class Validator** {

**static** isNameValid(name) {

**return** (/^.+$/**i**.**test**(name))

}

**static** isEmailValid(email) {

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

}

**static** isSexValid(sex) {

**return** (sex === 'male' || sex === 'female')

}

**static** isPasswordValid(password) {

**return** (/^(**?=**.\*[a-z])(**?=**.\*[A-Z])(**?=**.\*\W)(**?!**.\* ).{8,}$/.**test**(password));

}

**static** isAccessLevelValid(accessLevel) {

**return** (accessLevel === 'admin' || accessLevel === 'user');

}

**static** isBirthDateValid(birthDate) {

**return** (/^(1[0-2]|0?[1-9])\/(3[01]|[12][0-9]|0?[1-9])\/([0-9]{2}?[0-9]{2})$/.**test**(birthDate));

}

}

**class User** {

constructor(name, lastName, birthDate, password, sex, email, accessLevel) {

**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) {

**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) {

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

**return** **this**.password = newPassword;

}

setEmail(newEmail) {

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

**return** **this**.email = newEmail;

}

setAccessLevel(newAccessLevel) {

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

**return** **this**.accessLevel = newAccessLevel;

}

}

**class App** {

listOfUsers = [];

createUser(name, lastName, birthDate, password, sex, email) {

**try** {

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

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

} **catch** (error) {

console.**log**(error)

**return**

}

}

createAdmin(name, lastName, birthDate, password, sex, email) {

**try** {

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

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

} **catch** (error) {

console.**log**(error)

**return**

}

}

findUserOrAdminIndex(email) {

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

};

checkAccessLevel(email) {

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

**if** (indexUser === -1) **return** -1; //nie ma takiego usera/admina

**if** (**this**.listOfUsers[indexUser].accessLevel === 'user') **return** -1;//nie masz uprawnien do zmiany

**return** indexUser;

}

setPassword(adminEmail, userEmail, newPassword) {

**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, userEmail, newEmail) {

**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, userEmail, newAccessLevel) {

**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**

}

}

}

**const** myApp = **new** App();

myApp.createAdmin('Jan', 'Jaki', '12/13/2000', 'a!Dja4@#', 'male', 'aaa@op.pl')

myApp.createUser('Ula', 'Pakos', '10/21/2012', 'Arsadiu&', 'male', 'p@op.pl')

myApp.createUser('Ola', 'Bala', '10/5/1999', 'Jaia\*&%fd', 'male', 'xxx@op.pl')

console.**log**(...myApp.listOfUsers)

**Zad. 6** library-Stwórz strukturę danych związaną z biblioteką

Wypracuj obiekt user, book, booking, library. Dodaj możliwość posiadania więcej niż 1 książki w bibliotece - będzie trzeba zmienić logikę dodawania i usuwania książki z biblioteki (np. gdy książka jest już w bibliotece to dodajemy do niej kolejną sztukę).

-User {Imie, Nazwisko, uuid}

-Book {Tytuł, Autora, uuid, losowe zdjęcie oraz krótki opis}

-Booking {Booking dostaje użytkownika w constructorze i Ma mieć: datę wypożyczenia, datę zwrotu (+7d od wypożyczenia), listę wypożyczonych książek, kara.Ma umożliwiać:

Usuwanie/dodawanie książki do listy wypożyczonych,zwrot,jeśli terminowo to kara=0, jesli nie to naliczenie jakiejś kary.}

-Library {Ma miec: listę książek, listę książek dostępnych (które nie zostały wypożyczone), lista wypożyczeń, lista użytkowników. Ma umożliwiać: dodaw/usuw z listy,

wypożyczanie książki dla usera X, oddanie wypożyczania książki}.

**class User** {

constructor(name, lastName) {

**this**.name = name;

**this**.lastName = lastName;

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

}

}

**class Book** {

constructor(title, author, picture, description) {

**this**.title = title;

**this**.author = author;

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

**this**.picture = picture;

**this**.description = description;

}

}

**class Booking** {

borrowedBookList = [];

constructor(userName, userLastName) {

**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; //5 zl za 1 rozpocz. dzien spoznienia z 1 ksiazka

**this**.chargeSum = 0;

}

findBookIndex(bookTitle) {

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

}

addBookToBorrowedList(book) {

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

}

deleteBookFromBorrowedList(bookTitle) {

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

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

}

countCharge(actualReturnDate) {

**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);

}

returnBook(bookTitle) {

**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`);

}

}

**const** book1 = **new** Book('Pan Tadeusz', 'A.Mickiewicz', 'panTad.jpg', 'Dawno,dawno temu..')

**const** book2 = **new** Book('Potop', 'H.Sienkiewicz', 'hSienk.jpg', 'Byly sobie swinki 3...')

**const** book3 = **new** Book('Marysieńka', 'A.Kopacka', 'aKop.jpg', 'Marysia była bardzo...')

**const** booking1 = **new** Booking('Marek', 'Kondrad')

booking1.addBookToBorrowedList(book1)

booking1.addBookToBorrowedList(book2)

booking1.addBookToBorrowedList(book3)

**class Library** {

allLibraryBookList = [];

availableBookList = [];

bookingsList = [];

userList = [];

findBookIndex(bookTitle, list) {//podaje w param. dana liste w ktorej szukamy.

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

}

addBookToLibrary(book) {//par.jest zmienna z obiektem ksiazki(a nie jej tytul!)

**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) {

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

**if** (bookIndexInAvailables !== -1) **return** **this**.availableBookList[bookIndexInAvailables].quantity++;

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

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

**return** **this**.availableBookList.push(bookIndexInAllbooks);

}

deleteBookFromLibrary(bookTitle) {

**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) {

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

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

**if** (**this**.availableBookList[bookIndexToDelete].quantity === 1) **return** **this**.availableBookList.splice(bookIndexToDelete, 1);

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

}

makeBooking(userName, userLastName, bookTitle) {

**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]);

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

**if** (**this**.availableBookList[bookIndexInAvailablelList].quantity === 1) **return** **this**.availableBookList.splice(bookIndexInAvailablelList, 1);

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

// ZALOZENIEM JEST, ze REZERWACJA jest tworzona NA 1 KSIAZKE!!!

}

returnBooking(userName, userLastName, bookTitle) {

**const** findBookingIndex = **this**.bookingsList.findIndex(e =>

(e.user.name.**toLowerCase**() === userName.**toLowerCase**()) &&

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

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

)//szukam rezerwacji na tego usera i ksiazke

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

**this**.bookingsList[findBookingIndex].returnBook(bookTitle)//metodą classy Booking usuwam z listy pozyczonych i naliczam oplate

**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)])//dodaje oddana ksiazke do listy dostepnych

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

}

}

**const** biblioteka = **new** Library();

biblioteka.addBookToLibrary(book1)

biblioteka.addBookToLibrary(book2)

biblioteka.addBookToLibrary(book3)

biblioteka.addBookToLibrary(book3)

biblioteka.makeBooking('Janusz', 'Wunderbaumer', 'potop');

biblioteka.makeBooking('Jan', 'Janowicz', 'pan tadeusz')

biblioteka.returnBooking('Janusz', 'Wunderbaumer', 'potop')

console.**log**(...biblioteka.bookingsList)