OOP - zadania

Zadanie 1

Klasa Point2D

Zaimplementuj klasę `Point2D`. Klasa powinna zawierać:

- dwa pola typu float: `x`, `y`
- konstruktor bezparametrowy ustawiający wartość pól `x` i `y` na `0`
- konstruktor z dwoma parametrami: `float x`, `float y`
- metody typu getter odpowiedzialne za zwracanie wartości zmiennej: `x`, `y`
- metodę getXY zwracającą współrzędne x i y w postaci tablicy dwuelementowej
- metody typu setter odpowiedzialne za ustawianie wartości pól `x`, `y`
- metodę setXY ustawiającą współrzędne `x` i `y`
- metoda `toString` powinna zwracać łańcuch tekstowy o następującym formacie: `(x, y)`;

Klasa Point3D

Na podstawie klasy Point2D zaimplementuj klasę Point3D. Klasa ta powinna rozszerzać klasę Point2D oraz dodawać następującą implementację:

- pole prywatne typu float: `z`
- konstruktor przyjmujący wartości dla pól: `x`, `y`, `z`
- metodę typu getter odpowiedzialną za zwracanie wartości zmiennej `z`
- metodę `getXYZ` zwracającą współrzędne `x`, `y`, `z` w postaci tablicy trzyelementowej
- metodę typu setter odpowiedzialną za ustawianie zmiennej `z`

- metodę `setXYZ` ustawiającą wartości dla zmiennych `x`, `y`, `z`
- metoda `toString` powinna zwracać łańcuch tekstowy o następującym formacie: `(x, y, z)`;

Zaprezentuj zaimplementowane powyżej rozwiązanie na przykładzie.

Zadanie 2

Klasa Person

Zaimplementuj klasę Staff. Klasa powinna rozszerzać klasę `Person`. Klasa powinna zawierać:

- dwa pola typu String: `name`, `address`
- konstruktor bezparametrowy ustawiający wartość pól `name` i `address` na pusty String
- konstruktor z dwoma parametrami: `String name`, `String address`
- metody typu getter odpowiedzialne za zwracanie wartości zmiennej: `name`, `address`
- metody typu setter odpowiedzialne za ustawianie wartości pól `name`, `address`
- metoda `toString` powinna zwracać łańcuch tekstowy o następującym formacie: `?->?`, gdzie `?` to odpowiednio imię i adres;

Klasa Student

Zaimplementuj klasę `Student`. Klasa ta powinna rozszerzać klasę `Person`. Dodatkowo powinna zawierać:

- trzy pola: typ studiów, rok studiów, koszt studiów
- konstruktor z trzema parametrami: typ studiów, rok studiów, koszt studiów
- metody typu getter dla zdeklarowanych pól
- metody typu setter dla zdeklarowanych pól
- metoda `toString` wyświetlająca szczegółowe informacje o studencie

Klasa Staff

Zaimplementuj klasę `Staff`. Klasa ta powinna rozszerzać klasę `Person`. Dodatkowo powinna zawierać:

- dwa pola: specjalizacja oraz wynagrodzenie
- konstruktor z dwoma parametrami: specjalizacja, wynagrodzenie
- metody typu getter dla zdeklarowanych pól
- metody typu setter dla zdeklarowanych pól
- metodę `toString` wyświetlającą szczegółowe informacje o wykładowcy

Zaprezentuj zaimplementowane powyżej rozwiązanie na przykładzie.

Zadanie 3

Klasa Shape

Zaimplementuj klasę `Shape`. Klasa powinna zawierać:

- pole odpowiedzialne za przechowywanie koloru
- pole odpowiedzialne za przechowywanie informacji o tym czy kolor powinien wypełniać figurę czy nie
- konstruktor bezparametrowy ustawiający wartość pola `color` na `unknown` i `isFilled` na `false`
- konstruktor przyjmujący parametry `color` i `isFilled`
- metody typu `getter` odpowiedzialne za zwracanie wartości pól klasy
- metody typu `setter` odpowiedzialne za ustawianie wartości pól klasy
- nadpisaną metodę `toString` odpowiedzialną za wyświetlanie następującej informacji: `Shape with color of ? and filled/NotFilled`, gdzie ? oznacza wartość kolor, a wartość `filled` / `not filled` powinna zostać zwracana w zależności od pola `isFilled`

Klasa Circle

Zaimplementuj klasę `Circle`, która będzie rozszerzać klasę `Shape` o następujące cechy:

- pole odpowiedzialne za przechowywanie wartości promienia
- konstruktor bezparametrowy ustawiający wartość pola `color` na `unknown` i `isFilled` na `false` oraz pola `radius` na `1`
- konstruktor przyjmujący parametry `color`, `isFilled`, `radius`
- metodę typu `getter` odpowiedzialną za zwracanie wartości pola `radius`
- metodę typu `setter` odpowiedzialną za ustawianie wartości pola `radius`
- metodę `getArea` odpowiedzialną za obliczanie pola powierzchni
- metodę `getPerimeter` odpowiedzialną za obliczanie obwodu
- nadpisaną metodę `toString` odpowiedzialną za wyświetlanie następującej informacji: `Circle with radius=? which is a subclass off y`, gdzie ? oznacza wartość promienia, a wartość `y` powinna być rezultatem wywołania metody `toString` z klasu bazowej

Klasa Rectangle

Zaimplementuj klasę `Rectangle`, która będzie rozszerzać klasę `Shape` o następujące cechy:

- pole szerokość oraz długość będące typem `double`
- konstruktor bezparametrowy ustawiający wartość pola `color` na `unknown` i `isFilled` na `false` oraz pola `width` i `length` na `1`
- konstruktor przyjmujący parametry `color`, `isFilled`, `width` i `length`
- metody typu `getter` do zwracania wartości pól `width`, `length`
- metody typu `setter` do ustawiania wartości pól `width` i `length`
- metodę `getArea` odpowiedzialną za obliczanie pola powierzchni
- metodę `getPerimeter` odpowiedzialną za obliczanie obwodu
- nadpisaną metodę `toString` odpowiedzialną za wyświetlanie następującej informacji: `Rectangle with width=? and length=? which is a subclass off y`, gdzie ? oznacza wartość odpowiednio szerokości i długości, a

wartość `y` powinna być rezultatem wywołania metody `toString` z klasy bazowej

Klasa Square

Zaimplementuj klasę `Square`, która będzie rozszerzać klasę `Rectangle`. Klasa ta nie powinna wprowadzać nowych pól oraz funkcjonalności, ale powinna wymuszać na klasie bazowej zachowanie kwadratu.

Zaprezentuj zaimplementowane powyżej rozwiązanie na przykładzie.

Zadanie 4

Zmodyflkuj implementację przygotowaną w ramach Zadania nr 2. W tym celu dokonaj następującej refaktoryzacji:

- zmodyflkuj klasę `Shape` tak by była to klasa abstrakcyjna
- pola klasy `Shape` powinny być oznaczone modyflkatorem dostępu typu `protected`
- klasa `Shape` powinna zawierać metody abstrakcyjne `getArea` i `getPerimeter`

Wszystkie klasy dziedziczące bezpośrednio lub pośrednio po klasie `Shape` powinny nadpisywać metody abstrakcyjne z klasy nadrzędnej.

Zaprezentuj zaimplementowane powyżej rozwiązanie na przykładzie.

Zadanie 5

Zaimplementuj klasę `Line`, która będzie zawierać (na zasadzie kompozycji) instancję dwóch obiektów `Point2D` z zadania nr 1. Punkty te będą punktem początkowym oraz końcowym odcinka. Ponadto klasa ta powinna implementować:

- konstruktor przyjmujący dwa punkty: początkowy i końcowy

- konstruktor przyjmujący 4 parametry: współrzędne punktu początkowego oraz końcowego
- metody typu `getter` odpowiedzialne za zwracanie punktów: początkowego i końcowego
- metody typu `setter` odpowiedzialne za ustalanie punktów: początkowego i końcowego
- metodę odpowiedzialną za obliczanie długości linii na podstawie ustawionych punktów
- metodę odpowiedzialną za zwracanie współrzędnych punktu będącego środkiem stworzonej prostej

Zaprezentuj zaimplementowane powyżej rozwiązanie na przykładzie.

Zadanie 6

Zaimplementuj interfejs `Movable`, który będzie zawierać definicję wspólnych zachowań dla klas `MovablePoint` i `MovableCircle`. Będą to metody:

- `void moveUp()`
- `void moveDown()`
- `void moveLeft()`
- `void moveRigth()`

Klasa MovablePoint

Klasa `MovablePoint` powinna implementować interfejs `Movable`, a ponadto powinna zawierać 4 pola typu `int`: `x`, `y`, `xSpeed`, `ySpeed`. Pola `x`, `y` powinny definiować współrzędne punktu, natomiast pola `xSpeed`, `ySpeed` powinny określać o ile powinny zmieniać się odpowiednie współrzędne.

- metody `moveUp()` oraz `moveDown()` powinny każdorazowo zwiększać/zmniejszać wartość współrzędnej `y` o wskazaną wartość: `ySpeed`
- metody `moveLeft` oraz `moveRight()` powinny każdorazowo zwiększać/zmniejszać wartość współrzędnej `x` o wskazaną wartość

`xSpeed`

Klasa MovableCircle

Klasa `MovableCircle` powinna implementować interfejs `Movable`, a ponadto powinna zawierać (na zasadzie kompozycji) instancję klasy `MovablePoint`. Dodatkowo powinna zawierać pole niezbędne do określenia promienia koła.

- metody `moveUp()` oraz `moveDown()` powinny każdorazowo zwiększać/zmniejszać wartość współrzędnej `y` punktu `MovablePoint` o wskazaną w nim wartość: `ySpeed`
- metody `moveLeft` oraz `moveRight()` powinny każdorazowo zwiększać/zmniejszać wartość współrzędnej `x` punktu `MovablePoint` o wskazaną w nim wartość: `xSpeed`

Zaprezentuj zaimplementowane powyżej rozwiązanie na przykładzie. Dodatkowo uwzględnij w przykładzie reprezentację polimorflzmu.

Zadanie 7

Zaimplementuj interfejs `GeometricObject`, który będzie zawierać deflnicję wspólnych zachowań dla klas pochodnych:

- `double getPerimeter()`
- `double getArea()`

Klasa Circle

Klasa `Circle` powinna implementować interfejs `GeometricObject`, a ponadto zawierać pole: promień. Metody interfejsu`GeometricObject` powinny zostać zaimplementowane zgodnie z deflnicjami metematycznymi.

Interfejs Resizable

Interfejs `Resizable` powinien deklarować metodę `resize(int percent)`, która ma być odpowiedzialna za przeskalowanie obiektów implementujących

tworzony interfejs.

Klasa ResizableCircle

Klasa `ResizableCircle` powinna implementować interfejs `Resizable`. Metoda `resize` interfejsu powinna zmniejszać procentowo promień koła.

Zaprezentuj zaimplementowane powyżej rozwiązanie na przykładzie.

Wyjątki - zadania

Zadanie 1

Zaimplementuj metodę `divide`, która docelowo ma podzielić dwie liczby będące atrybutami metody. W przypadku, gdy drugi parametr metody jest równy 0, powinien zostać wyrzucony niedomyślny wyjątek: `CannotDivideBy0Exception`.

Zadanie 2

Klasa BookRepository

Zaimplementuj klasę `BookRepository`, która będzie odpowiedzialna za:

- dodawanie obiektów typu `Book`
- usuwanie obiektów typu `Book`
- wyszukiwanie obiektów typu `Book` o wskazanej nazwie
- wyszukiwanie obiektu typu `Book` o wskazanym id
- usuwanie obiektów typu `Book` na podstawie przekazanego `id`

Klasa Book

Klasa `Book` powinna zawierać poniższe pola:

- isbn
- tytuł
- autor
- rok wydania

NoBookFoundException

W przypadku braku jakichkolwiek rezultatów wyszukiwania, powinien zostać wyrzucony wyjątek: `NoBookFoundException` . Wyjątek ten powinien być własnoręcznie zaimplementowanym wyjątkiem, przyjmujący jako parametr konstruktora obiekt typu `String` z informacją jakich elementów nie udało się odszukać.

Klasy i interfejsy - zadania

Zadanie 1

Zaimplementuj klasę `UserValidator`, która w ramach metody `validateEmails` będzie odpowiedzialna za walidację danych użytkownika: email, email alternatywny. W ramach metody `validateEmails` zadeklaruj klasę lokalną `Email`, która będzie odpowiedzialna za formatowanie wskazanego adresu email uwzględniając następujące reguły:

- jeśli wskazany adres email jest pusty, bądź jest nullem należy ustawić mu wartość `unknown`
- jeśli wskazany adres email nie spełnia założeń adresu email, należy ustawić wartość `unknown` (skorzystaj w tym celu z wyrażeń regularnych)

Zadanie 2

Klasa Movie

Zaimplementuj klasę `Movie`, która będzie zawierać pola reprezentujące informacje takie jak: tytuł, reżyser, rok wydania, gatunek, wydawca. Klasa ta powinna zawierać domyślny konstruktor oraz metody typu `getter` i `setter`, oraz nadpisaną metodę `toString`, która będzie odpowiedzialna za zwracanie informacji o właściwościach konkretnego filmu.

Klasa MovieCreator

Zaimplementuj klasę zagnieżdżoną statyczną `MovieCreator`. Klasa ta powinna:

- zawierać pola klasy takie same jak klasa Movie
- zawierać metody umoliżwiające ustawianie konkretnych właściwości filmu. Każda z metod powinna zwracać instancję obiektu, na rzecz którego wywoływana jest metoda

- metodę `createMovie`, która na podstawie ustawionych parametrów stworzy instancję klasy `Movie` i zwróci ją w rezultacie działania metody

Zadanie 3

Klasa Car

Zaimplementuj klasę `Car`, która będzie przechowywać informacje o nazwie i typie samochodu. Klasa ta powinna zwierać metody typu `getter` i `setter`.

Klasa Engine

Zaimplementuj klasę `Engine`, która będzie zagnieżdzoną klasą niestatyczną w ramach klasy `Car`. Klasa ta powinna zawierać pole: typ silnika oraz metodę `setEngine`, która ustawi odpowiedni typ na podstawie typu samochodu. Jeśli typ samochodu będzie równy `economy`, to typ silnika powinien zostać ustawiony na `diesel`. Jeśli typ samochodu bedzie równy `luxury`, to typ silnika powinien zostać zdeflniowany jako `electric`. W innym przypadku typ silnika powinien być równy `petrol`.

Zadanie 4

Interfejs Validator

Zaimplementuj interfejs Validator, który będzie zawierać w swojej deklaracji metodę `boolean validate(T input)`.

Klasa User

Zaimplementuj klasę User, która będzie zawierać: * pola: imię, nazwisko, wiek, login, hasło * konstruktor bezparametrowy * metody typu `setter` i `getter` * metody typu `setter` powinny przyjmować w swoim ciele odpowiednią wartość pola oraz instancję klasy implementującej interfejs `Validator` * metody typu `setter` powinny wywoływać metodę `validate` na podstawie instancji przekazanego obiektu. Parametrem przekazywanym do metody `validate` powinna być wartość pierwszego argumentu

Klasy anoniowe

Mechanizmy walidujące klasy `Validator` powinny zostać zaimplementowane w postaci klas anonimowych. Ponadto poszczególne klasy anonimowe powinny weryfikować poniższe zasady:

- walidacja imienia: imię nie może być puste ani być nullem, powinno zaczynać się od wielkiej litery
- walidacja nazwiska: nazwisko nie może być puste ani być nullem, powinno zaczynać się od wielkiej litery
- walidacja wieku: wartość powinna mieścić się w przedziale o 0 do 150
- login: wartość pola powinna składać się z 10 znaków
- hasło: powinno zawierać znak `!`

Zaprezentuj zaimplementowane powyżej rozwiązanie na przykładzie.

Typ wyliczeniowy - zadania

Zadanie 1

Stwórz klasę enum `Weekday` ze stałymi `MONDAY`, `TUESDAY`, ... `SUNDAY`. Enum powinien zawierać metody `boolean isWeekDay` oraz `boolean isHoliday`. Metoda `isHoliday` powinna zwracać rezultat przeciwny od rezultuatu wywołania metody `isWeekDay`. Dodatkowo w ramach klasy enum powinna być zadeklarowana i zaimplementowana metoda `whichIsGreater`. Metoda ta powinna przyjmować obiekt typu `Weekday`. Metoda ta powinna wyświetlać informacje o tym, że wskazany dzień tygodnia jest poprzednikiem bądź następnikiem dnia tygodnia przekazanego w argumencie. W tym celu skorzystaj z metody `compareTo`.

Zaprezentuj zaimplementowane powyżej rozwiązanie na przykładzie.

Zadanie 2

Stwórz klasę enum `PackageSize` ze stałymi `SMALL`, `MEDIUM`, `LARGE`. Enum powinien przyjmować w konstruktorze dwa parametry:

- minimalny rozmiar paczki w `cm`
- maksymalny rozmiar paczki w `cm`

Dodatkowo enum `PackageSize` powinien zawierać metodę statyczną `getPackageSize`, która przyjmuje na wejściu minimalny i maksymalny rozmiar paczki, a jako rezultat powinna zwracać konkretny obiekt `PackageSize` na podstawie przekazanego rozmiaru paczki.

Zadanie 3

Stwórz klasę enum `TemperatureConvert` ze stałymi `C_F`, `C_K`, `K_C`, `F_C`, `F_K`, `K_F`. Enum powinien zawierać konstruktor przyjmujący trzy parametry:

- jednostkę temperatury na wejściu
- jednostkę temperatury na wyjściu
- instancję interfejsu `Converter` (z metodą `float convert(float tempIn)`) - instancja ta powinna deflnować niezbędne obliczenia w celu konwersji temperatury

Dodatkowo enum `TemperatureConvert` powinien zawierać metodę statyczną `convertTemperature`, która przyjmuje następujące parametry:

- jednostkę temperatury na wejściu
- jednostkę temperatury na wyjściu
- wartość temperatury

Metoda ta powinna zwracać skonwertowaną wartość. Do znalezienia odpowiedniej stałej należy skorzystać z metody `values()`.

Kolekcje - zadania

Zadanie 1

Zaimplementuj klasę `SDAArrayList<T>`, która będzie implementować logikę `ArrayList<T>`. W tym celu zaimplementuj obsługę metod:

- add
- remove
- get
- display

Zadanie 2

Klasa Author

Zaimplementuj klasę `Author`, która będzie zawierać pola: imię, nazwisko, płeć. Uwzględnij wszystkie niezbędne meotody oraz parametry konstruktora. Zaimplementuj klasę `hashCode` i `equals`.

Klasa Book

Zaimplementuj klasę `Book`, która będzie zawierać pola: tytuł, cena, rok wydania oraz lista autorów, gatunek (reprezentowany jako klasa enum). Uwzględnij wszystkie niezbędne metody oraz parametry konstruktora. Zaimplementuj klasę `hashCode` i `equals`.

Klasa BookService

Zaimplementuj klasę `BookService`, która będzie zawierać w sobie listę książek, oraz będzie realizować poniższe metody:

- dodawanie książek do listy

- usuwanie książek z listy
- zwracanie listy wszystkich książek
- zwracanie książek typu `Fantasy`
- zwracanie książek wydanych przed rokiem `1999`
- zwracanie najdroższej książki
- zwracanie najtańszej książki
- zwracanie książki z conajmniej `3 autorów`
- zwracanie listy wszystkich książek posortowanych zgodnie z przekazanym parametrem: `rosnąco/malejąco`
- sprawdzanie czy konkretna książka znajduje się na liście
- zwracanie listy książek napisanych przez konkretnego autora

Zadanie 3

Na podstawie `100` elementowej tablicy z losowo wybranymi wartościami z przedziału `0-50` zaimplementuj następujące funkcjonalności:

- zwróć listę unikalnych elementów
- zwróć listę elementów, które conajmniej raz powtórzyły się w wygenerowanej tablicy

Zadanie 4

Na podstawie zadania nr 2 zaimplementuj metodę, która będzie odpowiedzialna za zwracanie unikalnych par: klucz, wartość. Kluczem powinien być gatunek książki, a wartością jej tytuł.

Zadanie 5

Na podstawie zadania nr 2 zaimplementuj metodę, która będzie odpowiedzialna stworzenie stosu książek posortowanych od ceny najwyższej do najniższej.

Programowanie funkcyjne - zadania

Zadanie 1

Wykorzystując mechanizmy programowania funkcyjnego na podstawie zadanej struktury wyświetl:

- listę wszystkich epizodów
- listę wszystkich filmów
- listę wszystkich nazw sezonów
- listę wszystkich numerów sezonów
- listę wszystkich nazw epizodów
- listę wszystkich numerów epizodów
- listę wszystkich nazw video
- listę wszystkich adresów url dla każdego video
- tylko epizody z parzystych sezonów
- tylko video z parzystych sezonów
- tylko video z parzystych epizodów i sezonów
- tylko video typu Clip z parzystych epizodów i nieparzystych sezonów
- tylko video typu Preview z nieparzystych epizodów i parzystych sezonów

```
enum VideoType {
    CLIP, PREVIEW, EPISODE
}

class Video {
    public String title;
    public String url;
    public VideoType videoType;

    public Video(String title, String url, VideoType videoType) {
        this.title = title;
        this.url = url;
```

```
        this.videoType = videoType;
    }
}

class Episode {
    public String episodeName;
    public int episodeNumber;
    List<Video> videos;

    public Episode(String episodeName, int episodeNumber,
List<Video> videos) {
        this.episodeName = episodeName;
        this.episodeNumber = episodeNumber;
        this.videos = videos;
    }
}

class Season {
    public String seasonName;
    public int seasonNumber;
    List<Episode> episodes;

    public Season(String seasonName, int seasonNumber,
List<Episode> episodes) {
        this.seasonName = seasonName;
        this.seasonNumber = seasonNumber;
        this.episodes = episodes;
    }
}
```

Typy generyczne - zadania

Zadanie 1

Zaprojektuj klasę `Pair`, która w oparciu o typy generyczne będzie umożliwiała przechowanie dowolnej pary obiektów.

Zadanie 2

Zaimplementuj generyczną metodę `countIf`, która na podstawie tablicy dowolnego typu oraz wskazanej funkcji zliczy liczbę elementów spełniających warunek. Funkcją może być dowolony interfejs zaimplementowany anonimowo.

Zadanie 3

Zaimplementuj generyczną metodę `swap`, która będzie odpowiedzialna za zamianę pozycji wskazanych elementów tablicy.

Zadanie 4

Zaprojektuj klasę, która będzie zachowywać się jak biblioteka dla następujących rodzajów mediów:

- książek
- gazet
- filmów

Zaproponuj rozwiązanie w oparciu o typy generyczne. W celu przechowywania danych skorzystaj z tablic, bądź dowolnego API służącego do przechowywania danych.

Zadanie 5

Zaprojektuj klasę, która będzie zachowywać się jak dom dla zwierząt:

- kot
- pies

Zaproponuj rozwiązanie w oparciu o typy generyczne. W celu przechowywania danych skorzystaj z tablic, bądź dowolnego API służącego do przechowywania danych.

Java IO - zadania

Zadanie 1

Napisz program, który wyświetli wszystkie pliki/katalogi zawarte w danym katalogu.

Zadanie 2

Napisz program, który odczyta i wyświetli cały dowolny plik linia po linii.

Zadanie 3

Napisz program, który doda do wskazanego pliku dowolny łańcuch tekstowy.

Zadanie 4

Napisz program, który jest odpowiedzialny za zwrócenie najdłuższego słowa we wskazanym pliku tekstowym.

Zadanie 5

Napisz program, który będzie realizować funkcjonalność parsera CSV.

```
John,Smith,23
Sam,Johnson,40
Andrew,Manly,43
```

Z wykorzystaniem powyższego pliku jego odczytanie powinno skutkować zwróceniem listy trzyelementowej obiektów typu `User` o polach: imię, nazwisko, wiek.

Zadanie 6

Napisz program, który będzie realizować poniższe operacje na obiektach klasy `Movie`:

- dodawanie obiektów
- wyświetlanie listy obiektów

Klasa `Movie` powinna zawierać pola: `tytuł`, `gatunek`, `reżyser`, `rok wydania`. Dodawanie obiektów powinno przesyłać ich zserializowaną formę do pliku. Wyświetlanie listy obiektów powinno umożliwiać deserializację pliku tekstowego w celu konwersji poszczgólnych linii na obiekty.

Programowanie współbieżne i równoległe - zadania

Zadanie 1

Napisz program, który równolegle znajdzie liczby parzyste w dwóch przedziałach: `1000-2000` oraz `14300-17800`.

Zadanie 2

Napisz program, który ma za zadanie rozwiązać poniższy problem.

Na trasie pomiędzy miejscowościami `A` i `B` położony jest most, na którym może znajdować się tylko i wyłącznie jeden samochód. Zaimplementuj mechanizm, który umożliwi zsynchronizowany dostęp do obiektu klasy `Bridge`, obiektom klasy `Car`.

Klasa `Car` powinna zawierać poniższe informacje:

- nazwa samochodu
- typ samochodu

Klasa Bridge powinna zawierać metodę:

- driveThrough, która przyjmie jako parametr obiekt klasy `Car`. Przejazd powineń trwać 5s.

Zadanie 3

Napisz program, który na dwóch osobnych wątkach będzie wykonywać dwa niezależne algorytmy sortowania. Celem programu jest zwrócenie informacji o algorytmie, który wykonał się szybciej. Sortowana tablica `10000` powinna być tablicą losowo wygenerowanych liczb.

Zadanie 4

Napisz aplikację, która będzie synchronizować dostęp do rachunku bankowego. W przypadku, gdy dowolna cykliczna usługa internetowa będzie chciała obciążyć rachunek kwotą wyższą niż aktualnie dostępna, powinno nastąpić wstrzymanie wątku. W momencie, gdy na rachunek zostaną przelane dodatkowe środki, powinno nastąpić wzbudzenie wątku.

Zadanie 5

Napisz strukturę danych, która umożliwi poruszanie się po tablicy w dwóch kierunkach:

- do przodu (`next()`)
- do tyłu (`prev()`)

Struktura danych powinna przechowywać aktualnie przeszukiwany index. Zadbaj o jego dodatkową synchronizację.

Podstawy refleksji - zadania

Zadanie 1

Wykorzystując klasę `Student` o poniższych cechach:

- klasa powinna zawierać pola: imię, nazwisko, nr indeksu, kierunek studiów
- klasa powinna zwierać konstruktor bezparametrowy, oraz konstruktor przyjmujący jako argument wartości dla każdego z pól
- metody typu `setter`
- metody typu `getter`

Wyświetl następujące informacje wykorzystując mechanizm refleksji:

- dostępne metody
- dostępne pola
- dostępne konstruktory

Zadanie 2

Na podstawie klasy `Student` z zadania nr 1 wykonaj następujace operacje wykorzystując mechanizm refleksji:

- tworzenie obiektu z przekazaniem wszystkich wymaganych parametrów
- wywołanie metod typu `getter`
- bezpośrednie modyflkacje wartości pól prywatnych