Klasa Klient

* Pola: imie, nazwisko, nrtelefonu, nrDowodu, lista klientów.
* Sprawdzenie czy nrDowodu ma odpowiedzi format jak nie to wyjątek.
* Konstruktor Statyczny by ustawiał liste na pustą.
* Konstuktor domyślny który ustawia na puste pole.
* Kontruktor Parametryczny który ustawia pola które się samemu ustawi.
* ToString wypisuje dane.

Klasa Pracownik

* string imie;
* string nazwisko;
* string pesel;
* bool dostepny;
* string nrTelefonu;
* List<Pracownik> pracownicy;

Sprawdzenie czy pesel składa się z 11 liczb.

Sprawdzenie czy telefon się składa z 9 liczb.

Konstruktor domyślny i parametryczny (wiadomo po co na co).

Metody dodania pracownika i wyszukania pracownika (wsm nie wiem czy tu nie powinien być interface ale idk, zrobiłem wyszukanie po nazwisku, ale skalna się doczepi się może być kilka pracowników o tym naziwsku, wiec wymyśl coś Filip).

Metody ale to działka Filipa, rozpędziłem się xd

Klasa Samochód: ABSTRAKCYJNA KLASA

* string marka;
* string model;
* string numerRejestracyjny;
* bool czyDostepny;
* int rokProdukcji;
* EnumStan stan;

Wszystko to samo co w poprzednich klasach

Teraz będą klasy pochodne które są na jedno kopyto.

Klasa Samochód Sportowy: Samochód

* int konieMechaniczne;
* bool podwojnyTlumik;

dziedziczy po samochodzie i dodaje dwa pola jak jest sportowy.

Klasa Samochód Cabriolet : Samochód

EnumDach dach;

To samo co Sportowy

Klasa Samochód klasyczny : Sasmochód

Nie dodałem żadnych pol, bo to takie basic auto.

Klasa Samochód wieloosobowy (w sensie 8 osobowy czy tam 12 czy tam ile może byc)

enum EnumMiejsca { Siedmioosobe, Dziewiecioosobowe}

Klasa Wypożyczalnia:IWypozyczanie

* List<Wypozyczenie> wypozyczenia = new List<Wypozyczenie>();

Inicjuje pustą liste.

ToString wypisuje tą liste.

Oraz zaimprentowane metody z interface IWypozyczanie (jak cos to możesz dodac tam cos jeszcze)

Klasa Wypożyczenie

static int nrWypozyczenia;

DateTime dataWypozyczenia;

DateTime dataZwrotu;

Samochod samochod;

Klient klient;

decimal cenaZaDzienWypozyczenia;

Platnosc platnosc;

Konstruktor STATYCZNY który ustawia nr na 1(potem zwiększa się co jeden, jakie ID)

Kontruktor parametryczny to wiadomo ustawia pola.

Sprawdzenie czy Data Zwrotu jest pozniejsza niż data wypożyczenia, ofc wyjątek.

Metoda kwota() zwraca koszt wypożyczenia iloczyn dni i ceny za dzień wypożyczenia.

ToString wypisuje.

Klasa Płatnosci (tu będzie mordenga)

Implementuje interface IPlatnosci z metodami

* public void DodajPlatnosc();
* public void DokonajPlatnosci();
* public void WyszukajNieZaplacone();
* public void ZmienTerminOddania();

pola :

* + public Wypozyczenie wypozyczenie;
  + public decimal Kwota;
  + public DateTime? dataPlatnosci;
  + public bool Potwierdzona;
  + static List<Platnosc> platnosci = new List<Platnosc>();
* public DateTime? DataPlatnosci { get; set; }

dataPlatnosci może być null, bo jeżeli nie zostało zapłacone to wiesz bez sensu.

Statyczny konstruktor inicjuje pustą liste płatności (taki jakby rachunek bankowy)

Domyślny konstruktor ustawia potwierdzenie na false.

Parametryczny ustawia pola na konkretna wartość, a na końcu dodaje do tej listy tą płatność bo bez sensu zrobić taka metode jak każda płatnośc musi być w liscie.

To string wypisuje listę płatności i uwzględnia czy zostało zapłacone czy nie.

Dodałem klase listaaut która ma w sobie liste ze wszystkimi autami, i wypisuje ją.

No to tak:

Klonujemy klase Wypozyczenie i chyba wszystko działa ale nie umiałam poradzić sobie z tym żeby ID wypożyczenia się nie zwiekszalo wiec tu może ktoś mi pomoze ☹

Jak chodzi o interfejs IComparable to on sortuje Wypozyczenia w Wypozyczalni według nazwisk i imion klientów

Interfejs IEquatable sprawdza czy pracownik jest pracownikiem (to będzie spoko dzialac jak będziemy mieć więcej siedzib. Wtedy będzie można sprawdzuc czy dany pracownik pracuje w jakiejś siedzibie).