System zarządzania lokalnym serwisem komputerowym posiadającym salon ze sprzętem komputerowym

Autor: Piotr Kaczałko

Spis treści:

I.

Cel: Program ma za zadanie ułatwić i zautomatyzować czynności zarządzania biznesem polegającym na naprawie oraz sprzedaży sprzętu komputerowego.

Reguła biznesowa: serwis powinien przynosić zyski w rozliczeniu rocznym. Przyjętym celem minimalnym jest pokrycie kosztów działalności. Docelowym celem jest natomiast coroczne zwiększanie przychodów o co najmniej 10% względem poprzedniego roku.

Sytuacja gospodarcza: Serwis ulokowany jest w Świętochłowicach. To jedyne miejsce, w którym naprawia się sprzęt w okolicy oraz jedne z 6 lokalnych sklepów komputerowych.

Ograniczenia biznesowe: Prawo reguluje: pensje minimalną, stały podatek od usług, sposób przechowywania danych dotyczących pracowników i klientów; umowa o pracę jest wymagana prawnie.

Pełnione usługi: Sprzedaż detaliczna, przygotowanie sprzętu zakupionego w salonie, szeroki zakres usług informatycznych, diagnostyka i naprawa zepsutego sprzętu

Opis wstępny:

1) Obsługa Klienta:
   Obsługa zajmuje się przetwarzaniem informacji o klientach. Wprowadza dane dotyczące sprzedaży, umawia klientów na wizytę, zleca naprawę sprzętu.

2) Klienci:
   Tutaj znajdują się dane klientów, ich transakcji i zleceń

3) HR (Zasoby ludzkie):
   HR przechowuje wszystkie dane o pracownikach. Zajmuje się też ich zatrudnianiem i zwalnianiem.

4) Magazyn:
   Magazyn przechowuje informację o ilości produktów, o stopniu zapełnienia magazynu i ilości części zamiennych.

5) Serwis:
   W serwisie wykonuje się zlecone usługi. Do systemu wpisuje się wykonane usługi przez pracownika, wraz z cenami i kosztem łącznym.

6) Finanse:

   Finanse zajmują się budżetem. Przypisują pensję pracownikom, księgują wpływy ze sprzedaży i serwisu. Wszyscy pracownicy zarabiają miesięcznie jednakową, bazową kwotę. Obsługa klienta otrzymuje dodatek w zależności od ilości sprzedanych produktów promowanych, mogą otrzymać dodatkową premię po sprzedaży wszystkich tych produktów.
   Technicy otrzymują dodatek od ilości udanych napraw.

7) Statystyki:

   Zajmuje się przetwarzaniem danych o sprzedaży, ilości wykonanych usług i innymi statystykami

Najważniejsze funkcje:

- Wprowadź dane sprzedaży : obsługa
- Umów klienta na wizytę : obsługa
- Zleć usługę : obsługa
- Wyświetl stan magazynu(Procent zapełnienia) : magazyn
- Wyświetl ilość produktów: magazyn
- Dodaj/Usuń produkt : magazyn
- Dodaj naprawę : serwis
- Aktualizuj naprawę : serwis
- Wyświetl pensje : finanse/HR
- Zmień dodatek/premię : finanse/HR
- Zatrudnij/Zwolnij pracownika : HR
- Statystyki sprzedaży : statystyki

II. Scenariusze

Obsługa Klienta:

Wprowadzenie danych sprzedaży
Warunek początkowy: Klient chce kupić przedmioty
Przebieg: 1. Pobierz datę 2. Wpisz nazwę i id produktu 3.Wprowadz cenę. 4 wprowadź nr pracownika. 5.Utwórz id transakcji
Warunek końcowy: Nie wystąpił żaden błąd.

Zlecenie wykonania usługi
Warunek początkowy: klient wprowadził dane usługi
Przebieg: Wprowadź: 1. Imię i nazwisko klienta 2.nr telefonu klienta 3.Nazwa usługi/przyjętego sprzętu 4.id sprzętu lub usługi 5.data
Warunek końcowy: Zlecenie zostało poprawnie przyjęte i nie wystąpiły żadne błędy

Wprowadzenie danych wizyty

Warunek początkowy: klient chce się umówić na wizytę.
Przebieg: Wpisanie: 1.Celu wizyty 2.Imienia i nazwiska klienta 3.nr telefonu klienta 4.daty i godz. wizyty

Warunek końcowy: klient wybrał wolny termin i został zapisany na wybrany dzień.

Klienci:

Wyświetl dane klienta:
Warunek początkowy: Istnieje co najmniej jeden klient.
Przebieg: 1. Pobierz informacje z systemu 2. Wyświetl informacje.
Warunek końcowy: Nie wystąpił żaden błąd.

Wyświetl transakcje klienta:
Warunek początkowy: Istnieje co najmniej jeden klient z dokonanym zakupem lub usługą.
Przebieg: 1. Pobierz informacje z systemu 2. Wyświetl informacje.
Warunek końcowy: Nie wystąpił żaden błąd.

HR:

Wyświetl pracowników
Warunek początkowy: Istnieje co najmniej jeden pracownik
Przebieg: 1.Pobierz dane z systemu 2.wyświetl dane.
warunek końcowy: wyświetlono dostępne dane i nie wystąpił żaden błąd.

Dodaj pracownika
Warunek początkowy:
Przebieg: 1. Wpisz imię pracownika 2.wpisz nazwisko pracownika 3. Wpisz rolę pracownika
Warunek końcowy: Pomyślnie zapisano wpis do bazy

Zwolnij pracownika
Warunek początkowy: wybrano opcję z menu, istnieje co najmniej 1 pracownik.
Przebieg: 1.wybierz kryterium szukania(id, imię, nazwisko) 2. Wpisz wartość odpowiednią dla poprzedniego kroku 3.Potwierdz wykonanie akcji
Warunek końcowy: Usunięto wpis z bazy.

Magazyn:

Wyświetlanie zapełnienia magazynu:
Warunek początkowy:
przebieg: 1. Pobierz dane z systemu 2.wyświetl dane
Warunek końcowy: Pomyślnie wyświetlono dane.

Dodaj nowy produkt:
Warunek początkowy:
przebieg: Wprowadź: 1.Nazwę produktu 2.id produktu 3.Cenę produktu
Warunek końcowy: Pomyślnie dodano produkt.

Usuń produkt:
Warunek początkowy: Wybrany produkt istnieje
przebieg: 1.Wprowadź id produktu 2. Usuń produkt
Warunek końcowy: Pomyślnie usunięto produkt.

Serwis:

Wykonaj usługę:
Warunek początkowy: istnieje zlecenie usługi
Przebieg: 1. Pobierz dane zlecenia 2. Wprowadź wykonane czynności 3.Oblicz cenę 4. Wpisz id pracownika
Warunek końcowy: Wykonano usługę

Finanse:

Wyświetl pensje

Warunek: Istnieje co najmniej jeden pracownik
Przebieg: 1.Pobierz informacje z systemu 2.wyświetl dane.
warunek końcowy: wyświetlono dostępne dane

Zmień pensje

Warunek: Istnieje przypisana pensja do pracownika
Przebieg: 1.wyświetl dane z bazy danych 2. Wpisz id pracownika 3.edytuj pensje 4.wyświetl potwierdzenie
warunek końcowy: potwierdzono zmianę pensji

Zmień premie
Warunek: Istnieje przypisana premia do pracownika
Przebieg 1.wpisz id pracownika 2. Wpisz wysokość premii.
Warunek końcowy: poprawnie wprowadzono dane.

Statystyki

Pokaż statystyki

III. Identyfikacja danych, deklaracja danych i funkcji

1.Typy danych

Obsługa Klienta:

Sprzedaż: int: idSprzedaży, idProduktu,idPracownika,idKlienta; string dataSprzedaży

Zlecenie: int: idKlienta,idZlecenia,idPracownika ; string: nazwa,dataPrzyjęcia

Wizyta: int idKlienta; string: data,cel,godzina

Uslugi: string nazwa, float cena

Sprzedaze: wektor Sprzedaz*,unsigned int liczba_sprzedazy

Zlecenia: wektor Zlecenie*, unsigned int liczba_zlecen, Uslugi *uslugi

Wizyty: wektor Wizyta*, unsigned int liczba_wizyt

Klienci:

Klient: int idKlienta ; string: imięKlienta, nazwiskoKlienta, nrTelefonuKlienta

HR:

Pracownik : Int: idPracownika, rolaPracownika ; string: imiePracownika, nazwiskoPracownika

Pracownicy: Pracownik pracownicy, Pensje pensja, int ilość_pracownikow

Magazyn:

Produkty: int idProduktu; string nazwaProduktu; float cenaProduktu

Adres: string miejsce

Magazyn: wektor Produkty*; int pojemność, liczba_produktow, mutable int ile magazynow , static

Adres *adres

Finanse:

Pensje: Int idPracownika float: pensja, premia

Budżet: float: przychody, wydatki

IV. Moduły

Obsługa:

Klasy: Obsluga, Sprzedaz, Zlecenie, Wizyta, Sprzedaze,Zlecenia,Wizyty

Metody:

Metoda wirtualna `dodaj()` : W zależności od sytuacji przypisuje odpowiednie dla danej klasy dane.

Metoda wirtualna `usun()` : W zależności od sytuacji usuwa odpowiednie dla danej klasy dane.

Sprzedaz:

```
class Sprzedaz  {
private:
	unsigned int idPracownika;
	unsigned int idKlienta;
	unsigned int idSprzedazy;
	unsigned int idProduktu;
	string dataSprzedazy;
public:
	void set_id_klienta(unsigned int);
	void set_id_pracownika(unsigned int);
	void set_id_sprzedazy(unsigned int);
	void set_id_produktu(unsigned int);
	void set_data_sprzedazy(string);
	unsigned int get_id_klienta();
	unsigned int get_id_pracownika();
	unsigned int get_id_sprzedazy();
	unsigned int get_id_produktu();
	string get_data_sprzedazy();

class Sprzedaze :public Obsluga { - klasa agregujaca sprzedaż , dziedziczaca z obsluga
private:
	vector<Sprzedaz*> sprzedaz; - wektor przechowujący dane o sprzedazy
	unsigned int liczba_sprzedazy = 0; - liczba elementow w wektorze
public:
	friend class interfejs;
	Sprzedaze() = default;
	Sprzedaze(int); - konstruktor przyjmujący liczbe elementow do stworzenia w wektorze
	  void sprzedaj(int,int); - funkcja zapełniajaca wektor, przyjmuje informacje o koleno ilości
                                                          klientów i ilości produktów w magazynie
	virtual void dodaj(int,int); - funkcje dodające i usuwające poszczególne indeksy z wektora
	virtual void usun(int);

	~Sprzedaze();
};
```

```
class Zlecenie {
private:
	unsigned int idPracownika;
	unsigned int idKlienta;
	unsigned int idZlecenia;
	unsigned int idProduktu;
	string dataPrzyjecia;
	string godzinaPrzyjecia;
	unsigned int nrUslugi;
public:
```

```
        void set_id_zlecenia(unsigned int);
        void set_id_produktu(unsigned int);
        void set_data(string);
        void set_godzina(string);
        void set_nr_uslugi(unsigned int);
        unsigned int get_id_zlecenia();
        unsigned int get_id_produktu();
        string get_data();
        string get_godzina();
        unsigned int get_nr_uslugi();
        void set_id_klienta(unsigned int);
        void set_id_pracownika(unsigned int);
        unsigned int get_id_klienta();
        unsigned int get_id_pracownika();
};

class Zlecenia :public Obsluga {  klasa agregujaca dane o zleceniach, dziedziczaca z obsluga
private:
        class Uslugi { -klasa zagniezdzona zawierajca informacje o uslugach
        private:
                string nazwaUslugi;
                float cenaUslugi;
        public:
                friend class Zlecenia;
        };
        Uslugi* uslugi;
        vector<Zlecenie*> zlecenia; - wektor zawierający dane o zleceniach
        unsigned int liczba_zlecen = 0; - liczba elementow w wektorze
public:
        friend class interfejs;
        Zlecenia() = default;
       Zlecenia(int); -konstruktor przydziela pamięć dla wektora i tablicy obiektow usługi, przyjmuje
                                                                        liczbe elementow do stworzenia
        void set_cena_uslugi(int, float);
        string get_nazwa_uslugi(int);
        float get_cena_uslugi(int);
        void zlec(int,int); - funkcja wypelniajaca wektor zlecenia
        virtual void dodaj(int,int);- funkcje dodające i usuwające poszczególne indeksy z wektora
        virtual void usun(int);
        ~Zlecenia();
};


class Wizyta  {
private:
        unsigned int idKlienta;
        unsigned int idWizyty;
        string dataWizyty;
        string celWizyty;
        string godzinaWizyty;
public:
        unsigned int get_id_klienta();
        unsigned int get_id_wizyty();
        void set_data(string);
        void set_cel(string);
        string get_data();
        string get_cel();
        void set_id_klienta(unsigned int);
        void set_id_wizyty(unsigned int);
        void set_godzina_wizyty(string);
        string get_godzina_wizyty();
};
```

```
class Wizyty :public Obsluga { klasa przechowujaca w wektorze dane o wizytach
private:
	vector<Wizyta*> wizyty; - tworzenie wektora przechowującego dane o wizytach
	unsigned int ostatnie_id = 0; liczba przechowujaca informacje o ostatnim przydzielonym id
	unsigned int liczba_wizyt = 0; - liczba elementow w wektorze
public:
	friend class interfejs;
	Wizyty()= default;
	Wizyty(int);
	bool czy_wolne(string,string);  -funkcja sprawdzajaca czy dana kolejno data i godzina jest
dostepna
	void zaplanuj(int,int); funkcja wypelniajaca wektor wizyty
	virtual void dodaj(int,int);
	virtual void usun(int);
	~Wizyty();
```

Klienci

Klasy: Klient, Klienci

```
class Klient {
private:
	unsigned int id_klienta;
	string imie_klienta;
	string nazwisko_klienta;
	string nr_tel_klienta;
public:
	void set_id_kl(unsigned int);
	void set_imie_kl(string);
	void set_nazwisko_kl(string);
	void set_nr_tel_kl(string);
	unsigned int get_id_kl();
	string get_imie_kl();
	string get_nazwisko_kl();
	string get_nr_tel_kl();
};
```

```
class Klienci { klasa przechowujaca dane o klientach w wektorze klienci
private:
	vector<Klient*> klienci; - utworzenie wektora
	unsigned int liczba_klientow = 0; - liczba elementow w wektorze

public:
	friend class interfejs;
	 friend ostream& operator<<(ostream&, const Klienci&);          - operator << zwraca wyswietla
                                                                wszystkie dane z wektora klienci
	Klienci(int); - konstruktor tworzący okreslona ilość elementow w wektorze
	unsigned int ile_klientow(); funkcja zwracajaca ilość elementow w wektorze
	void wyswietl_klienta(int); funkcja wyswietlajaca określony indeks wektora
	void pozyskaj_klientow(); funkcja zapelniajaca wektor danymi
	void nowy_klient(); funkcja dodajaca nowy indeks do wektora
```

HR:

Klasy: Pracownik, Pracownicy , Numery

```
class Pracownik {
	int idPracownika;
	string rolaPracownika;
	string imiePracownika;
	string nazwiskoPracownika;
	class Numery { -klasa zawierajaca informacje o num. Telefonu i ich ilości oraz typie
		string numer_telefonu;
		string typ;
	public:
		friend class Pracownik;
	};
	Numery* numer;
	int ilosc_numerow;


public:
	Pracownik();
	void set_id(int id);
	void set_rola(string rola);
	void set_imie(string imie);
	void set_nazwisko(string nazwisko);
	int get_ilosc_nr();
	string get_numer(int i);
	string get_typ(int i);
	int get_id();
	string get_rola();
	string get_imie();
	string get_nazwisko();
	void ustaw_numery();
	~Pracownik();
};


class Pracownicy {
	vector<Pracownik*> pracownicy; - stworzenie wektora zawierającego dane pracownikow
	vector<Pensje*> pensja; - stworzenie wektora zawierającego dane o pensjach
	int ilosc_pracownikow; liczba elementow w wektorze
	unsigned int ostatnie_id = 0; - zmienna przechowujaca informacje o ostatnim nadanym id
public:
	friend class interfejs;
	  friend ostream& operator<<(ostream&, const Pracownicy&);    - operator << wyswieta wszystkie
                                                                      dane z wektora pracownicy
	void operator()(string); -operator()wyswietla informacje czy nazwisko znajduje się w wektorze
	Pracownik* operator[](int); - operator []zwraca pojedynczy indeks wektora pracownicy
	Pracownicy& operator=(Pracownicy &a); operator = tworzy o polowe mniejsza kopie wektora
	Pracownicy() = default;
	Pracownicy(int); -kostruktor tworzy dana liczbe elementow w wektorach
	void zatrudnij_Pracownikow(); funkcja przypisujaca dane do wektorow
	void zmien_role(); - funkcja zmieniajaca role pracownikowi
	void wyswietl_Pracownikow(); - funkcje wyswietlajace wszystkich lub wybranego pracownika
	void wyswietl_Pracownika(int);
	void dodaj_Pracownika(); - funkcje dodające nowego pracownika lub usuniecie określonego prac.
	void usun_Pracownika(int);
	void wyswietl_pensje(); funkcja wyswietlajaca pensje
	void zmien_Premie(); - funkcje zmieniające premie lub pensje
	void zmien_Pensje();
	Pracownicy(const Pracownicy& a);
	~Pracownicy();

 };
```

Magazyn:

Klasy: Produkty, Magazyn

```
class Produkty {
private:
	unsigned int idProduktu;
	string nazwaProduktu;
	float cenaProduktu;
	unsigned int ilosc_na_stanie;
public:
	friend ostream& operator<<(ostream&, const Produkty&); - operator << wyswietla pojedynczy
produkt
	void set_id(int);
	void set_nazwa(string);
	void set_cena(float);
	void set_ilosc(int);
	unsigned int get_id();
	string get_nazwa();
	float get_cena();
	unsigned int get_ilosc();
};



class Adres {
	string miejsce;
public:
	Adres();
	string get_adres();
};



class Magazyn {
	vector<Produkty*> produkty; wektor zawierający dane o produktach
	static Adres* adres; agregacja statyczna klasy adres
	unsigned int ilosc_produktow = 0;
	mutable int ile_magazynow{ 1 };
	unsigned int pojemnosc;
	void zapelnij_magazyn(); funkcja przypisujaca dane do wektora produkty
public:
	friend class interfejs;
	friend ostream& operator<<(ostream&, const Magazyn&); operator wyswietla wszystkie produkty
	Produkty* operator[](int); operator zwraca pojedynczy produkt z wektora
	void set_ile_magazynow(int)const; set dla zmiennej statycznej
	unsigned int rozmiar();
	unsigned int get_ilosc_produktow();
	unsigned int get_ile_magazynow()const;
	static string pokaz_adres(); statyczna funkcja pokazujaca adres magazynow
	static void usun_adres();
	void wyswietl_stan(); funkcja wyswietla wszystkie dane z wektora
	void zapisz(string); funkcja przyjmujaca sciezke do zapisu do pliku elementow z wektora
	void dodaj_produkt(); funkcja dodająca element do wektora
	void usun_produkt(int); funkcja usuwajaca określony indeks z wektora
	Magazyn(int, int); konstruktor przyjmujący ilość elementow do wektora i pojemność magazynu
	~Magazyn();
```

Finanse:

Klasy: Pensje

```
class Pensje{
private:
	int idPracownika;
	float pensja;
	float premia;

public:
	Pensje();
	void set_id(int id);
	void set_pensja(float pen);
	void set_premia(float prem);
	int get_id();
	float get_pensja();

	float get_premia();
```

Los

Moduł zawierający funkcje losujące, służące do m.in. losowego wypełniania wektorów z danymi.

```
	int losuj_int(int); - losuje liczbe calkowita z przedzialu [0,int liczba)
	 int losuj_int(string); -    losuje liczbe calkowita o określonym przez string ilości cyfr np.
                                                                „3” to przedział [100,999]
	float losuj_float(); - losuje liczbe zmiennoprzecinkowa;
	string losuj_imie(); - losuje imie z pliku
	string losuj_nazwisko(); - losuje nazwisko z pliku
	string losuj_nazwe();   losuje nazwe produktu z pliku
	string losuj_nr_tel(); - losuje ciag znakow o dlugosci 9
	string losuj_date(int=2021); losuje date z określonego intem roku
	string losuj_date(string); losuje date oddalona o maks. 2 miesiace w przyszlosc
	string losuj_godzine(); - losuje godzine z zakresu (10:18) z przeskokiem co 15 minut
	string losuj_cel(); losuje cel wizyty z pliku

	string losuj_zlecenie(); losuje rodzaje zlecen z pliku
```

Interfejs

Moduł „nadzorujący” program. Tworzy interfejs dla użytkownika.

```
class interfejs{
	Magazyn* magazyn_o;
	Pracownicy* pracownicy;
	Sprzedaze* sprzedaz;
	Wizyty* wizyty;
	Zlecenia* zlecenia;
	Klienci* klienci;

public:
	interfejs() ; - konstruktor przydziela pamięć obiektom dla wskazników
	void start(); - funkcja rozpoczynajaca program
	void magazyn(); funkcja dla magazynu
	void hr(); funkcja dla hr
	void serwis(); funkcja dla serwisu
	void statystyki(); funkcja dla statystyki
	void obsluga(); funkcja dla obslugi
	void wyswietl_zlecenia(); funkcja wyswietlajaca zlecenia
	void wykonaj_zlecenie();
	void aktualizuj_zlecenie();
	void stat_sprzedazy(); funkcja wyswietlajaca statystyki sprzedaży danego produktu
	void stat_pracownikow(); funkcja wyswietlajaca statystyki prownikow
	 unsigned int znajdz(char,int); funkcja zwracajaca indeks pod którym znajduje się określone id
	                                                                    w odpowiednim wektorze
	int wal(); funkcja pobierajaca od użytkownika liczbe;
```