1. Koristila sam objektno orijentisano programiranje. Klasa ***HitniPacijent*** je ***sealed*** (koja je naslijeđena iz klase Pacijent, a klasa Pacijent je naslijeđena iz klase Osoba). Sealed znači da je to finalna verzija te klase, te se ona više ne može nasljeđivati u neke druge klase. Fja koja je sealed ne može biti bazna klasa. Sealed u C sharpu ima isto značenje kao final u c++ programskom jeziku.

Polimorfizam sam koristila u klasama Osoba, i u klasama koje su izvedene iz nje, tj. Pacijent, Doktor. Virtualne metode u klasi Osoba su Ispisi i UnosOsobe. U klasi pacijent i doktor prvo pozivamo fje iz bazne klase, i to ukoliko je u pitanju konstruktor base(parametri), a ukoliko su u pitanju ove dvije navedene fje base.UnosOsobe(), odnosno base.Ispisi().

Klasu ***Klinika*** sam implementirala sa ***partial*** specifikatorom pristupa. To znaci da se definicija klase može implementirati u više različitih foldera koji su u istom asembliju.

Za ***nazive instanciranih objekata*** sam koristila ono što se tražilo, tj. „brindexa\_broj“ . To sam postigla tako što sam u klasama stavila ***static atribut*** tipa string koji sam inicijalizirala vrijednošću „000“. Staticki atribut je zajednički za sve instance te klase. Svaki put kada bih instancirala npr. novog pacijenta, automatski bih povecavala i broj tog statičkog atributa za 1. Povecavanje sam vršila tako što sam prvo vršila konverziju u broj, pozivom fje Convert.ToNumber(staticki\_atribut), na taj broj dodavala 1 i potom ponovo vraćala string u staticki atribut. S obzirom da se stringovi mogu „sabirati“ tj. spajati, jednostavnom linijom ime = "klinika17574\_" + ukupno\_klinika; sam dodjeljivala uvijek razlicita imena.

Za svaki mogući unos, sam se pokušala maksimalno zaštiti od pogrešnih ulaznih podataka. To sam postigla raznim funkcijama koje su za cilj imale validaciju, kao i mnogobrojnim try i catch strukturama, namijenjene za slučaj bacanja neočekivanih ili očekivanih izuzetaka.

2.Interfejs se može postići tako što samo cijeli main ubacimo u fju tipa public, npr u klasi Klinika.

3.

***Delegat*** sam iskoristila za fju ***ValidacijaUnosa***, koja je imala zadatak da provjeri da li je odabrana opcija iz pocetnog menija validna.

Delegat sam nazvala MojDelegat, a napravila sam ga na sljedeći način.

U klasi program sam najprije deklarisala delegat

delegate bool MojDelegat(ref int par);

Nakon toga sam instancirala instancu delegata, nazvanu delagatic

MojDelegat delegatic;

a nakon toga sam inicijalizirala delegat delagatic metodom ValidacijaUnosa

delegatic = new MojDelegat(ValidacijaUnosa);

i potom sam umjesto direktnog poziva fje ValidacijaUnosa, koristila delegat.

Konkretno u mom primjeru nisam postigla neku pretjeranu funkcionalnost, čak šta više, svelo se na standardno pozivanje fje, ali preko reference na metodu. Delegati su mnogo korisni ji npr za evente, kod GUI –a.

4. Za smještanje podataka koristila sam tip ***List***<tip> lista\_neka. Po potrebi sam dodavala elemente fjom lista\_neka***.Add***(nesto\_tipa\_tip), analogno i izbacivala elemente iz liste pozivom fje lista\_neka.***Remove***(nesto\_tipa\_tip).

5.Kreiran je dll file, MeineDLL, ubačena klasa Aparat.

6.Poštovala sam pravila programske ***dekompozicije***, koja se zasnivaju na podjeli programskog koda na manje, pojedinačne, smislene cjeline. Tako sam odvojila nekoliko klasa u posebne cjeline solutin-a. To su klinika, osoba, ordinacija, pregled, aparat.

7. Meni je isti kao traženi. Sve je obavljeno što je navedeno da se treba uraditi.

Prva opcija služi za dodavanje novog odnosno brisanje već postojećeg pacijenta.

Opcija brisanje pacijenta, služi da pacijenta trajno izbrišemo iz baze podataka Klinike. Brisanje pacijenta se obavlja samo isključivim odabirom te opcije iz menija. Inače, kada se izvrši naplata, podaci o pregledima, se poništavaju, ali ostaju ostali podaci. To je izvršeno sa ciljem, da ukoliko budemo vršili razna istraživanja o našim pacijentima imamo potrebne podatke u bazi kao i da znamo ko su naši redovni pacijenti koji dobijaju dodatnih 10% popusta na cijenu usluga.

Dodavanje novog pacijenta, to je zamišljeno da se obavlja na portirnici, daju se osnovni podaci ime, prezime, datum\_rodjenja, JMBG, adresa,...

Tom prilikom se pacijent odmah ispituje, koje vrste pregleda zeli, te se kreira raspored\_pregleda, koji je tipa List<Pregled>, tu se dodaju trazeni pregledi, a raspored pregleda se kreira sortirano i to na principu, idi prvo na odjel gdje je manji broj pacijenata koji već čekaju. Svaki pregled ima ime ordinacije, svaka ordinacija ima svoju listu cekanja.

Druga stavka menija služi da se prikaze raspored\_pregleda prikazan po redoslijedu.

Treca stavka kreira karton koji sam predstavila dodatnim atributom u klasi Pacijent, u kombinaciji sa 3 dodatna atributa, koji su vezani za trenutne bolesti, prijašnje bolesti pacijenta, te zdravstveno stanje u porodici.

Cetvrta stavka vrsi pretragu kartona pacijenta

Da bismo mogli koristiti petu stavku, pacijent mora imati kreiran elektronski karton. Peta stavka služi da registrujemo, tj. upišemo pregled koji je pacijent obavio.

U sestoj stavki menija, stavila sam 4 moguće opcije. Opcije su prikaz svih doktora, svih pacijenata, prikaz najčešćih dječijih zarazih bolesti, koji je odjel najmanje posjećen u tekućoj godini.

Sedma stavka je naplata. Naplatu sam realizovala tako da svaki pregled bez obzira u kojoj ordinaciji se radi košta 15 KM. S obzirom da imam atribut List<Pregled> svi\_obavljeni\_pregledi, jednostavno broj ob pregleda pomnozim sa 15. Eh ovisno od toga da li se radi o redovnim pacijentima koji dobijaju dodatni popust ili ipak o neredovnim. Da li je pacijent redovan ili nije, provjeravala sam tako što sam napravila atribut broj dolazaka. Svaki put prilikom naplate usluga klinike, taj broj sam povecavala. U slucaju da je broj dolazaka na kliniku > 3 pacijent dobija dodatni popust od 10%. Također pacijent ima pravo da bira 2 načina biranja, gotovinsko ili plaćanje na rate.

Osma stavka predstavlja izlaz iz aplikacije.