OBJEKTNO ORIJENTISANO PROGRAMIRANJE

- domaći zadatak broj 3 -

Funkcionalna specifikacija

Na programskom jeziku C++ implementirati paket klasa za rad sa bazom podataka kompanije za letove. U nastavku teksta data je specifikacija svih klasa u sistemu.

Osoba (Person) sadrži ime i prezime koji se zadaju prilikom inicijalizacije. *Putnik (**Passenger**)* je osoba koja poseduje novac koji se zadaje prilikom inicijalizacije, kao i strategiju za kupovinu karte i koja može da se postavi (*setStrategy(BuyingStrategy*):void*). Putniku može da se doda i oduzme novac (*giveMoney(double):void, takeMoney(double):void*). Novac ne može da ode u minus. Putnik može da kupi kartu za let (*Passenger::buyTicket(Airport* src, Airport* dst):bool*). Karta se kupuje tako što se aerodromu sa kojeg putnik želi da pođe prosleđuju informacije o odredišnom aerodromu i vraća spisak letova koji odgovaraju željenoj liniji leta. Na osnovu zadate strategije, bira se jedan od letova i kupuje karta za taj let. Ukoliko putnik nema dovoljno novca za kupivinu, operacija je neuspešna. Putnik sme da poseduje najviše jednu kartu u svakom trenutku. *Član posade (**CrewMember**)* ima svoju ulogu u avionu koja se zadaje prilikom inicijalizacije (*pilot, copilot, other*).

Strategija (BuyingStrategy) za kupovinu može da pronađe kartu koja najviše odgovara strategiji u željenoj listi letova (*findBestTicket(list<Flight*>):Ticket**). Postoji nekoliko tipova straregija: ***CheapTicketStrategy***, ***ExpensiveTicketStrategy***, ***AverageTicketStrategy***, koje traže kartu kao najjeftiniju kartu, najskuplju kartu ili kartu čija je cena najbliža srednjoj vrednosti cena svih letova, respektivno.

Karta (Ticket) sadrži cenu koja se zadaje, i putnika koji joj se dodeljuje pri kupovini. Cena može da se dohvati (*getPrice():double*). Kartu može da stvori samo *Let (Flight)*.

Avion (Plane) se inicijalizuje zadavanjem broja mesta za putnike. Sadrži listu putnika i listu članova posade. Avion sadrži i status (*READY, ONFLIGHT, FINISHED*) koji može da se dohvati (*getStatus():Status*) i postavi (*setStatus(Status):void*). Prilikom inicijalizacije aviona status je *FINISHED*. Avionu mogu da se dodaju putnici (*addPassenger(Passenger*):void*) i da se izbace svi putnici (*clearPassengers():void*), da se dodaju članovi posade (*addCrewMember(CrewMember*):void*) i izbacuju članovi posade (*removeCrewMember(CrewMember*):bool)*. Avion sme da sadrži samo jednog pilota, jednog kopilota i proizvoljan broj ostalih članova posade.

Aerodrom (Airport) se zadaje koordinatama u ravni i brojem pisti. Sadrži kontrolu leta koja se kreira pri stvaranju na osnovu broja pisti. Sadrži i listu aerodroma sa kojima je povezan (između povezanih aerodroma su mogući letovi). Može da se izračuna udaljenost između dva aerodroma (*distance(Airport*, Airport*):double*), da se povežu dva aerodroma (*connect(Airport*, Airport*):void*) i da se ispita da li su dva aerodroma povezana (*connected(Airport*, Airport*):bool*). Može da se dohvati lista svih letova sa datog aerodroma ka drugom aerodromu (*getFlightsTo(Airport* dest):list<Flight*>*).

Kontrola leta (FlightControl) upravlja letovima na zadatom aerodromu. Sadrži određen broj pisti na kojima mogu da se nađu avioni koji poleću sa povezanog aerodroma. Avion može da se postavi na pistu pri čemu mu se menja status u *READY* (*setOnRunway(Plane*):void*). Svi putnici koji su kupili kartu za let datim avionom se dodaju u avion. Greška je ukoliko ne postoji let u kojem avion poleće sa povezanog aerodroma, ukoliko je avion trenutno u letu ili ukoliko nema slobodne piste (*PlaneNotSuitableException, PlaneNotOnGroundException, NoFreeRunwayException*). Avion može da se skloni sa piste (*removeFromRunway(Plane*):void*). Greška je ukoliko avion nije na pisti (*PlaneNotOnRunwayException*). Može da se dopusti poletanje aviona sa piste pri čemu mu se menja status u *ONFLIGHT* (*allowTakeOff(Plane*):void*). Greška je ukoliko avion nije spreman za poletanje i ukoliko avionu nisu dodeljeni pilot, kopilot i najmanje pet ostalih članova posade (*PlaneNotReadyException, NoCrewException*). Može da se dopusti sletanje aviona na pistu pri čemu mu se menja status u *FINISHED* i iskrcavaju svi putnici iz aviona (*allowLanding(Plane*):void*). Svim putnicima se uništavaju karte. Greška je ukoliko avion nije u letu ka odgovarajućem aerodromu (*PlaneCanNotLandException*). Mogu da se dohvate avioni koji su spremni za poletanje

(*readyForTakeOff():list<Plane*>*) sa povezanog aerodroma i avioni koji su spremni za sletanje (*readyForLanding():list<Plane*>*) ka povezanom aerodromu.

Let (Flight) se inicijalizuje zadavanjem jedinstvenog generisanog identifikacionog broja leta, polaznog i dolaznog aerodroma, aviona koji obavlja let i cene po jedinici udaljenosti. Neke informacije mogu da se dohvate (*getId():int, getSrcAirport():Airport*, getDestAirport():Airport*, getPlane():Plane**). Let sadrži listu prodatih karata. Može da se kreira i doda nova karta u listu prodatih (*createTicket():Ticket**). Može da se dohvati cena leta kao proizvod udaljenosti između aerodroma i cene po jedinici udaljenosti (*getPrice():double*). Samo baza podataka može da napravi novi dinamički podatak na osnovu zadatih vrednosti.
(*FlightDatabase::createFlight(int id, Airport* src, Airport* dst, Plane*, double pricePerUnit): Flight**).

Generator sekvence (SequenceGenerator) omogućava generisanje sekvence celih brojeva počevši od 1. Pravi se bez parametara. Moze da vrati sledeći broj u sekvenci (*next():int*).

Baza letova (FlightDatabase) sadrži dinamički alocirane letove. Sme postojati samo jedna baza letova u sistemu. Ima svoj pridruženi generator sekvence. Može se postaviti novi generator sekvence (*setGenerator(SequenceGenerator*):void*). Moguće je dodati nov let u bazu (*insertFlight(Flight*):void*). Greška je ukoliko aerodromi nisu povezani (*AirportsNotConnectedException*). Ukoliko se pokuša ubacivanje leta sa već postojećim identifikacionim brojem u bazi, potrebno je prijaviti grešku izuzetkom *IdConflictException*. Moguće je izbrisati let iz baze (*deleteFlight(Flight*):void*). Greška je ukoliko let ne postoji u bazi (*FlightNotInDatabaseException*). Moguće je dohvatiti listu letova u bazi (*getFlightList():list<Flight*>*). Moguće je dohvatiti listu letova koji odgovaraju zadatoj vrednosti (*findBySrcAirport(Airport* src):list<Flight*>, findByDestAirport(Airport* dst):list<Flight*>, findByPlane(Plane*):list<Flight*>, findByPlaneStatus(Status):list<Flight*>, findByMaxPrice(double max):list<Flight*>*).

Test funkcija

U projektu glavnog programa treba da postoji funkcija `void test();` koja testira rad sa sistemom. Studenti treba da implementiraju datu funkciju i uslovno je prevode ako je definisan makro STUDENT_TEST. Predvideti i postojanje makroa PROF_TEST. Ako su oba makroa definisana, PROF_TEST ima prioritet i tada se prevodi tajni test primer koji će biti dat na odbrani. Funkcija main treba samo da pozove test funkciju.

Tehnički zahtevi i smernice za izradu rešenja

Sve klase i metode moraju biti imenovane prema zahtevima iz domaćeg zadatka. Svaka klasa koja koristi dinamičku memoriju mora biti bezbedna za korišćenje. Programski kod klasa rasporediti u odgovarajuće .h i .cpp fajlove. Ukoliko u zadatku nešto nije dovoljno jasno definisano, treba usvojiti razumnu pretpostavku i na temeljima te pretpostavke nastaviti izgrađivanje svog rešenja. *Svi izuzeci treba da budu izvedeni iz klase DBException!*

VAŽNE NAPOMENE

Za uspešno odbranjen domaći zadatak potrebno je na odbrani pokazati kod podeljen na odgovarajuće projekte, .h i .cpp fajlove.

- Klase sistema smestiti u poseban projekat rešenja koji se prevodi kao statička biblioteka (**`database.lib`**).
- Glavni program napisati u posebnom projektu koji se prevodi kao Win32 Console Application (**`testdz3.exe`**) fajl i koji treba povezati sa statičkom bibliotekom. Glavni program treba implementirati tako da pozove globalnu funkciju `void test();`.
- Studenti treba da implementiraju svoju verziju ove funkcije tako da demonstriraju sve funkcionalnosti sistema.
- Od kolekcija iz STL-a smeju da se koriste `<vector>`, `<list>`, `<stack>`, `<queue>` i `<string>`