OBJEKTNO ORIJENTISANO PROGRAMIRANJE

- domaći zadatak broj 3 -

Funkcionalna specifikacija

Na programskom jeziku C++ implementirati paket klasa koje predstavljaju koncepte baze podataka. Implementirati i glavni program koji poziva test funkciju. Studenti treba da implementiraju svoju test funkciju, ali i da omoguće uključivanje tajne test funkcije, na već opisan način.

Podatak (Data) ima jedinstven generisani ceo broj (*primary key*) i tekstualnu vrednost (*value*) koje se mogu dohvatiti. Pravi se zadavanjem broja i tekstualne vrednosti. Dva podatka se mogu uporediti na jednakost poređenjem sadržaja(==). Podržati i poređenje po sadržaju (vrednosti) na različitost(!=). Podatak moze da se polimorfno kopira (*clone():Data**) i može mu se dohvatiti identifikator. Podatak može da prihvati selekcioni kriterijum (*accept(SelectionCriterion*):void). Primeniti obrazac Visitor*. Samo baza podataka može da napravi novi dinamički podatak na osnovu zadate tekstualne vrednosti (*Database::createData(string value): Data**).

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 podataka (Database) sadrži dinamički alocirane podatke. Sme postojati samo jedna baza podataka u sistemu (obrazac Singleton). Ima svoj pridruženi generator sekvence. Može se postaviti novi generator sekvence (*setGenerator(SequenceGenerator*):void*). Baza podataka moze da izvrši proizvoljnu operaciju (*execute(Operation*):void*). Ukoliko operacija ne uspe, greška se prijavljuje podizanjem izuzetka OperationFailedException. Baza izvršava operaciju na sledeći način (primeniti šablonsku metodu): (1) prvo se pokrene izvršavanje same operacije; (2) ako operacija uspe (ne podigne izuzetak OperationFailedException) smešta se u zapisnik uspešno završenih operacija (transaction log); (3) ako operacija ne uspe, ne čuva se u zapisniku transakcija, ali se njeni eventualni parcijalni efekti na podacima moraju poništiti (*rollback*) i baza se mora vratiti u stanje u kom je bila pre nego što je počela neuspešna operacija.

Apstaktna operacija (Operation) moze da se izvrši (*execute(): void*). Ukoliko operacija iz bilo kog razloga ne može da se izvrši, podiže se izuzetak tipa *OperationFailedException*. Operacija može da se poništi (*cancel(): void*). Poništavanjem, operacija mora da vrati podatke koje je menjala u bazi, u stanje kakvo je bilo pre njenog izvršavanja. Može da se dohvati broj podataka na koje je operacija uticala (*affectedDataCount(): int*).

Insert je konkretna operacija koja ubacuje zadati podatak u bazu podataka. Stvara se zadavanjem tekstualne vrednosti koju treba ubaciti u bazu podataka. Izvršava se tako što napravi podatak i ubaci ga u bazu podataka. Ukoliko se pokuša ubacivanje podatka sa istim identifikatorom, potrebno je prijaviti grešku izuzetkom *IdConflictException*. Operacija se poništava uklanjenjem ubačenih podataka iz baze.

Update je konkretna operacija koja se izvršava tako što menja vrednost onih podataka u bazi, koji zadovoljovaju zadati selekcioni kriterijum. Stvara se zadavanjem nove tekstualne vrednosti i odgovarajućeg selekcionog kriterijuma. Vraća broj izmenjenih podataka. Poništava se tako što treba da svakom izmenjenom podatku vrati staru vrednost.

Select je operacija koja pronalazi podatke na osnovu zadatog selekcionog kriterijuma i vraća dinamičke kopije pronađenih podataka. Pravi se zadavanjem selekcionog kriterijuma i kolekcije u koju treba da smesti rezultat.

Apstraktni selekcioni kriterijum (SelectionCriterion) može da proveri podatak (*check(Data*):void*). Svaki podatak koji prilikom provere zadovoljava kriterijum, stavlja se u internu kolekciju kriterijuma, koja može da se dohvati. (Primeniti projektni obrazac Visitor.)

FindById je selekcioni kriterijum koji se stvara zadavanjem identifikatora objekta i proverava da li zadati podatak ima traženi identifikator. Najviše jedan podatak treba da ima traženi identifikator.

Ukoliko više od jednog podatka zadovoljava FindById kriterijum, prijaviti grešku tipa *IdConflictException*.

FindByValue je selekcioni kriterijum koji se stvara zadavanjem vrednosti po kojoj treba da se pretražuju podaci. Provera podatka se vrši se poređenjem po vrednosti/sadržaju.

Transakcija (Transaction) je složena operacija koja sadrži proizvoljno mnogo drugih operacija. Stvara je baza podataka (*Database::openTransaction():Transaction**). Omogućava dodavanje apstraktne operacije (*addOperation(Operation*)*). Izvršava se tako sto se redom izvršavaju sadržane operacije jedna po jedna, po redosledu u kom su dodavane. Ukoliko se desi da neka operacija ne uspe, sve do tada (uspešno) izvršene operacije moraju da se ponište. Transakcija se poništava tako što propagira poništavanje operacijama koje sadrži.

Test funkcija

U projektu glavnog programa treba da postoji funkcija `void test();` koja testira rad sa velikim brojevima. 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 i na kraju ispiše njeno trajanje u milisekundama, korišćenjem tipova i operacija iz zaglavlja <ctime>.

Tehnički zahtevi i smernice za izradu rešenja

Sve klase i metode moraju biti imenovane prema zahtevima iz domaćeg zadatka.

Sve klase i metode moraju biti imenovane prema zahtevima iz domaćeg zadatka. Poljima klase, zaštićenim od direktnog pristupa, se pristupa isključivo pomoću odgovarajućih metoda za čitanje i pisanje vrednosti polja. Klase treba da uključuju, pored zahtevanih operacija/metoda i sve druge metode koje obezbeđuju njihovo ***bezbedno korišćenje*** (konstruktori, destruktori, operatori). Za smešanje tekstualnih vrednosti upotrebiti tačno onoliko mesta u memoriji koliko je neophodno. Svi izuzeci treba da budu izvedeni iz klase DBException, koja je izvedena iz std::exception!

Glavni program treba da poziva metode/operacije koje obavljaju opisane radnje. Sve metode smestiti u odgovarajuće klase. Programski kod klasa rasporediti u odgovarajuće **.h** i **.cpp** fajlove. Nije dozvoljeno korišćenje globalnih promenljivih za razmenu podataka. Sva razmena podataka između funkcija mora ići preko povratne vrednosti i/ili liste argumenata. U slučaju bilo kakve greške (poziv programa sa neodgovarajućim brojem argumenata komandne linije, neuspešna dodela dinamičke memorije, greška pri radu sa datotekom ili bilo koja druga greška koja se može pojaviti u toku izvršavanja programa), prijaviti grešku podizanjem izuzetka. Greške se obrađuju u try-catch bloku.

VAŽNE NAPOMENE

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

- Klase baze podataka 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();`.
- Na kraju programa potrebno je ispisati na standardnom izlazu vreme trajanja funkcije `test` u milisekundama. Može se iskoristiti kod za merenje vremena na jeziku C++ koji je dat u zadatku 2.8 u materijalima za vežbe.
- NIJE DOZVOLJENO SMESTITI CEO KOD U JEDAN PROJEKAT ILI CPP fajl!

12.12.2015. godine sa predmeta