OBJEKTNO ORIJENTISANO PROGRAMIRANJE

- domaći zadatak broj 3 -

Projektuje se jednostavan softverski sistem za simulaciju rada bankomata. Korisnik interaguje sa sistemom pomoću komandne linije i interaktivnog menija, koji omogućava pokretanje svih dostupnih funkcionalnosti.

Opis koncepata i funkcionalna specifikacija

Bankomat (ATM) ima serijski broj, naziv banke i lokaciju i funkcioniše kao automat stanja. Automat stanja je definisan kao graf prototipskih objekata stanja koji se pravi prilikom stvaranja bankomata. Bankomat ima informaciju o svom trenutnom stanju. Ne sme postojati više od jednog bankomata. Kada bankomat menja stanje, trenutno stanje uvek postaje klon odgovarajućeg prototipskog stanja. Bankomat čuva informaciju o trenutno aktivnoj sesiji. Bankomat može da promeni stanje (changeState(currentState: State, newState: State)). Oba stanja moraju biti klonovi prototipskih. Omogućiti ispis bankomata u izlazni tok: osnovni podaci o bankomatu, plus sve sačuvane sesije.

Platna kartica ima jedinstveni serijski broj, ime klijenta i trenutni iznos sredstava (nenegativan ceo broj). Pravi se zadavanjem imena klijenta, serijski broj joj se automatski generiše. Može da se dopuni zadatim iznosom i mogu da joj se umanje sredstva za zadati iznos. Pokušaj podizanja više novca od iznosa dostupnog na kartici je neregularna situacija i signalizira se izuzetkom WrongAmmountException.

Stanje se pravi zadavanjem naziva uvek kao prototipsko (*isPrototype=true*). Može da se izvrši operacija stanja (execute). Može mu se dodati odredišno stanje u koje se prelazi kad se operacija stanja završi. Prilikom ulaska u stanje, odnosno izlaska iz stanja, predvideti pozivanje odgovarajućih operacija (*onEnter*, *onExit*). Podrazumevano se u operaciji *onEnter* poziva apstraktna operacija *setView* koja prikazuje odgovarajući ekran za interakciju u datom stanju. Podrazumevano, pri napuštanju stanja (*onExit*), ono se dodaje trenutno aktivnoj sesiji. Stanje može da se (polimorfno) klonira (clone), pri čemu klon nije prototipski objekat. Stanje čuva i ishod korisničke interakcije. Omogućiti (polimorfni) ispis (svih relevantnih podataka) stanja. Postoje sledeća stanja: početno stanje, unos kartice, izbor usluge, štampanje izveštaja kartice, podizanje novca. Poziv bilo koje operacije za prototipsko stanje rezultuje podizanjem izuzetka *IllegalStateException*.

U početnom stanju automat čeka na korisnika. Pri ulasku u stanje na ekranu se prikazuje poruka dobrodošlice. Stanje se izvršava tako što se čeka da korisnik inicira interakciju. Ako se unese 1, prelazi se u stanje za učitavanje kartice, a u slučaju 0 završava se program. Prilikom završetka programa, bankomat se ispisuje u fajl čije ime ime format: *atm_<timestamp>.out*.

U stanju „unos kartice" pri ulasku u stanje, korisniku se izlistaju sve kartice koje su već napravljene u programu, označene rednim brojevima. Stanje se izvršava tako što se čeka na unos rednog broja kartice. Ukoliko je redni broj pogrešan, ispisuje se poruka o grešci i "prelazi" u isto stanje. Predvideti opciju da korisnik odustane od učitavanja kartice, pri čemu automat prelazi u početno stanje. Posle učitanja kartice, automat prelazi u stanje "izbor usluge". Ishod stanja je serijskih broj učitane kartice.

U stanju "izbor usluge" korisniku se prikazuju osnovne informacije o kartici (ime klijenta i broj kartice), a zatim spisak dostupnih usluga (označenih rednim brojevima). Ponuđene opcije su: podizanje novca, štampanje izveštaja o kartici, prekid rada. Stanje se izvršava čekanjem na unos rednog broja usluge i prelaskom u odgovarajuće stanje. Ishod stanja je redni broj odabrane usluge.

U stanju "štampanje izveštaja" o kartici prikazuje se poruka "štampanje izveštaja u toku". Stanje se izvršava štampanjem izveštaja u fajl (*report_<timestamp>.out*): ime vlasnika, broj kartice, datum i vreme štampanja, raspoloživa sredstva. Po završetku štampanja, prelazi se u stanje za izbor usluge. Ishod stanja je putanja do fajla u koji je sačuvan izveštaj.

U stanju "podizanje novca" korisniku se prikazuje ekran za zadavanje iznosa za isplatu. Nakon unetog iznosa. Stanje se izvršava tako što se proveri validnost iznosa, pa zatim umanje sredstva na kartici i štampa potvrda. U slučaju neodgovarajućeg iznosa, automat "prelazi" u isto stanje i čeka na ponovni unos za isplatu. Nakon uspešne isplate, automat prelazi u stanje "izbor usluge". Potvrda isplate se štampa u fajl (*receipt_<timestamp>.out*) i sadrži sledeće podatke: ime vlasnika, broj kartice, datum i vreme isplate, isplaćen iznos i preostala sredstva. Ishod stanja kod uspešne isplate je putanja do fajla potvrde.

U slučaju bilo kakve greške, pri kojoj se automat zadržava u istom stanju, odnosno vrši prelaz iz stanja u isto takvo, ishod stanja je poruka o detektovanoj grešci.

Sesija ima svoj jedinstveni identifikator, vreme početka i vreme završetka. Sesija se stvara i započinje u trenutku prelaska iz početnog stanja u stanje za izbor usluge, a završava se prilikom prelaska iz bilo kog stanja u početno stanje (ako taj prelaz postoji u grafu prelaza). Bankomat čuva informacije o svim sesijama. Sesija sadrži sekvencu (kopija) stanja kroz koja je automat prolazio u toku trajanja sesije. Omogućiti ispis sesije, kao i sekvence stanja.

Za sve klase definisati odgovarajuće operacije/operatore za potrebne aktivnosti (na primer, sve klase treba da imaju definisan ispis). Polje <timestamp> u formatima imena izlaznih fajlova se formira na osnovu trenutnog datuma i vremena. Prilikom prikazivanja ekrana po ulasku u stanje, prethodni sadržaj ekrana je potrebno ukloniti (videti: *system("CLS")*).

Tehnički zahtevi i smernice za izradu rešenja

Opisane koncepte implementirati u vidu odgovarajućih klasa. Ukoliko je potrebno pristupati poljima klase zaštićenim od direktnog pristupa, to se radi isključivo pomoću odgovarajućih metoda za čitanje i pisanje vrednosti polja. Za smešanje tekstualnih vrednosti upotrebiti tačno onoliko mesta u memoriji koliko je neophodno. Svaka klasa koja koristi dinamičku memoriju mora imati korektno napisan destruktor, konstruktor kopije i preklopljen operator za dodelu vrednosti.

Glavni program treba da poziva metode/operacije koje obavljaju opisane radnje. Sve metode smestiti u odgovarajuće klase. Programski kod klasa rasporediti u istoimene **`.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 ne sme se program prekidati nasilno, već predvideti dodatne izuzetke. U trenutku detektovanje greške u toku izvršavanja programa, baciti izuzetak, uhvatiti ga na odgovarajućem mestu i tada pokušati oporavak. Program treba prekinuti jedino na eksplicitni zahtev korisnika. Koristiti strukture podataka isključlivo iz standardne biblioteke (ne treba praviti sopstvene).

Napomene:

1. Za uspešno odbranjen domaći zadatak potrebno je na odbrani pokazati kod podeljen na odgovarajuće projekte, .h i .cpp fajlove:
   - klase koje definišu strukturu, ponašanje bankomata treba smestiti u poseban projekat rešenja koji se prevodi kao statička biblioteka (**`atm.lib`**),
   - glavni program napisati u posebnom projektu koji se prevodi kao Win32 Console Application (**`AtmTest.exe`**) fajl i koji treba povezati sa statičkom bibliotekom.
     NIJE DOZVOLJENO SMESTITI CEO KOD U JEDAN PROJEKAT ILI CPP fajl!

21.12.2016. godine sa predmeta

PRILOG

Funkcije programskog jezika C/C++ za rad sa vremenom

U svakom trenutku je moguće od operativnog sistema dobiti informaciju o trenutnom vremenu, kao i o datumu i vremenskoj zoni gde se dati računar nalazi. Jednom kada tačno vreme bude očitano, ne postoji nikakava prepreka da se ono pretvori u odgovarajuću strukturu, odnosno prikaže/konvertuje u željenom formatu. Priloženi programski segment ilustruje predstavljanje tačnog vremena u željeni formatu:

```
#include <ctime>
...
  time_t pIntTime = time(NULL);
  struct tm* currentLocalTime = localtime(&pIntTime);
  char* dateTimeString = calloc(100+1, sizeof(char));
  /* if time setting and memory allocation was succesfull, create complete message */
  if (currentLocalTime && dateTimeString)
    /* format the time as needed */
    strftime(dateTimeString, 100, "%H:%M:%S", currentLocalTime);
```

Dodatne informacije o ovde navedenim funkcijama (i svim ostalim u zaglavlju **ctime**) su dostupne na raznim dokumentacionim stranicama (kako onima koje dolaze uz razvojno okruženje, tako onima dostupnim na Internetu).