**Domaća zadaća 2**

***Burza***

Rješenje druge domaće zadaće također predajte kroz sustav Nemesis, a Vaš zadatak je implementirati sustav za praćenje stanja burze. U rješenju se pretpostavlja korištenje više razreda. Razrede je moguće nazvati po želji, no bitno je da bude u prostoru imena DrugaDomacaZadaca\_Burza i da razred koji predstavlja burzu implementira sljedeće sučelje (uz funkcije je napisan kratak opis funkcionalnosti koju je potrebno implementirati):

public interface IStockExchange

{

void ListStock(string inStockName, long inNumberOfShares, Decimal inInitialPrice,

DateTime inTimeStamp); //dodaje dionicu s početnom cijenom na burzu

void DelistStock(string inStockName); //briše dionicu s burze

bool StockExists(string inStockName); //provjerava postoji li tražena dionica na burzi

int NumberOfStocks(); //vraća broj dionica na burzi

void SetStockPrice(string inStockName, DateTime inIimeStamp, Decimal inStockValue); //postavlja

// cijenu dionice za određeno vrijeme

Decimal GetStockPrice(string inStockName, DateTime inTimeStamp); //dohvaća cijenu dionice

// za neko vrijeme

Decimal GetInitialStockPrice(string inStockName); //dohvaća početnu cijenu dionice

Decimal GetLastStockPrice(string inStockName); //dohvaća zadnju cijenu dionice

void CreateIndex(string inIndexName, IndexTypes inIndexType); //stvara novi indeks na burzi

void AddStockToIndex(string inIndexName, string inStockName); //dodaje dionicu u indeks

void RemoveStockFromIndex(string inIndexName, string inStockName); //briše dionicu iz indeksa

bool IsStockPartOfIndex(string inIndexName, string inStockName); //provjerava je li dionica

// u indeksu

Decimal GetIndexValue(string inIndexName, DateTime inTimeStamp); //dohvaća vrijednost indeksa

bool IndexExists(string inIndexName); //provjerava postoji li traženi indeks na burzi

int NumberOfIndices(); //dohvaća broj indeksa na burzi

int NumberOfStocksInIndex(string inIndexName); //dohvaća broj dionica u traženom indeksu

void CreatePortfolio(string inPortfolioID); //stvara novi portfolio na burzi

void AddStockToPortfolio(string inPortfolioID, string inStockName, int numberOfShares); //dodaje

// određeni broj dionica u portfolio

void RemoveStockFromPortfolio(string inPortfolioID, string inStockName, int numberOfShares);

//dodaje određeni broj dionica iz portfolija

void RemoveStockFromPortfolio(string inPortfolioID, string inStockName); //briše dionicu

// iz portfolia

int NumberOfPortfolios(); //dohvaća broj portfolija na burzi

int NumberOfStocksInPortfolio(string inPortfolioID); //dohvaća broj dionica u traženom portfoliju

bool PortfolioExists(string inPortfolioID); //provjerava postoji li traženi portfolio na burzi

bool IsStockPartOfPortfolio(string inPortfolioID, string inStockName); //provjerava nalazi li

// se dionica u portfoliju

int NumberOfSharesOfStockInPortfolio(string inPortfolioID, string inStockName); //dohvaća broj

// dionice u traženom portfoliju

Decimal GetPortfolioValue(string inPortfolioID, DateTime timeStamp); //dohvaća vrijednost

// portfolija u određenom trenutku

Decimal GetPortfolioPercentChangeInValueForMonth(string inPortfolioID, int Year, int Month);

//dohvaća mjeseću promjenu vrijednosti portfolija

}

Kako bi se domaća zadaća mogla ocijeniti bez obzira na imenovanje i implementaciju samog razreda, za dohvat razreda koristit će se oblikovni obrazac tvornice. Zbog toga je potrebno implementirati razred Factory.

public static class Factory

{

public static IStockExchange CreateStockExchange()

{

return new StockExchange();

}

}

U datoteci s definicijom sučelja nalaze se još i definicije StockExchangeException

i IndexTypes čija će uloga biti objašnjena u definiciji zadatka.

Rješenje je potrebno predati u obliku JEDNE .cs datoteke, naziva po izboru, koja će sadržavati sve potrebne razrede i gore navedeni razred Factory. (Napomena: ne uključivati definiciju sučelja IStockEchange u tu datoteku jer se zbog kolizije programski kôd neće moći prevesti.)

Kako bi vam se dodatno olakšalo rješavanje ovog zadatka, u repozitoriju datoteka, u mapi Domaće zadaće se pored teksta ovog zadatka nalazi i arhiva s Visual Studio solutionom, u kojem su već stavljena potrebna sučelja, a također imate i skup osnovnih unit-testova za provjeru koliko ste dobro riješili zadatak. Skup testova pokriva osnovnu funkcionalnost i postoje zahtjevi koji nisu pokriveni testovima koji se nalaze u primjerima (ali hoće biti pokriveno u širem skupu testova po kojem će se evaluirati vaša rješenja), tako da to svakako uzmete u obzir!

**Rok za predaju rješenja je srijeda, 11.siječnja 2016. u 8:00.**

Kako bi se spriječili problemi kod buildanja prilikom prevođenja preporuča se koristiti .NET 3.5 (Može se postaviti pod opcijama projekta).

Zahtjevi na rješenje:

Za razliku od Kalkulatora gdje je većina posla bila implementacija računa za razne operacije, u ovom zadatku naglasak je stavljen na definiciju samog objektnog modela i rad s kolekcijama podataka.

U slučaju BILO KAKVE pogreške ili nedopuštene radnje korisnika je potrebno obavijestiti o stanju. Za to se koristi razred:

public class StockExchangeException : Exception

{

private string \_msg;

public StockExchangeException(string msg)

{

\_msg = msg;

}

}

Prilikom pogreške moguće je definirati bilo kakvu poruku te se tekst poruka neće provjeravati, no potrebno je svaki puta za prijavu pogreške koristiti ovaj razred.

Burza je osnovni objekt te se na njoj trguje određenim skupom dionica, čija se cijena formira na tržištu (i za naš sustav je eksterni faktor – znači, unosi se „izvana“!). Na burzi postoji određeni broj svake „vrste“ dionice (npr. Postoje 4 „IBM“ dionice čija je cijena 100). Osnovna namjena sustava izračunavanje je vrijednosti indeksa definiranih u sustavu. Indeksi mogu biti 2 vrste:

1) **AVERAGE** – vrijednost indeksa se računa kao jednostavan prosjek cijene svih dionica u indeksu

2) **WEIGHTED** – vrijednost indeksa se računa kao težinski prosjek cijene svih dionica u indeksu, gdje je težinski faktor za pojedinu dionicu definiran kao omjer ukupne vrijednosti te dionice na burzi (cijena dionice \* broj dionica) i ukupne vrijednosti svih dionica unutar indeksa

Prilikom dodavanja neke dionice na burzu za dionicu se definira ime, broj tih dionica, početna cijena i trenutak od kojeg ta cijena vrijedi. Ime dionice predstavlja ključ, tj. specifično je za svaku dionicu. Kako bi se spriječile pogreške djelatnika ime dionice neovisno je o veličini slova (tj. „ABC“, „abc“ i „AbC“ su iste dionice). Dionicu je u svakom trenutku moguće maknuti s burze. Za svaku se dionicu u određenim trenucima bilježi cijena. Cijena dionice vrijedi od trenutka u kojem je definirana do trenutka definicije sljedeće cijene. Nije bitan redoslijed unosa cijena, ali za određeni trenutak nije moguće unijeti više zapisa cijena. Nije potrebno provoditi provjere trenutaka unosa cijena kao što bi bilo provjera je li uneseni trenutak < DateTime.Now. Pretpostavlja se da se cijena može unijeti za svaki trenutak. Granularnost definicije vremena je 1ms, tj. unose se datum i vrijeme uključujući milisekunde. Cijena i broj dionica ne mogu biti negativni niti 0.

Prilikom dodavanja nekog indeksa na burzu za indeks se definiraju naziv i tip. Vrijednosti mogućih tipova indeksa definirane su enumom:

public enum IndexTypes

{

AVERAGE = 1,

WEIGHTED = 2

}

Stvaranje indeksa nekog drugog tipa ne smije biti dopušteno u sustavu te je u tom slučaju potrebno obavijestiti korisnika. Za naziv indeksa vrijede ista pravila kao i za naziv dionice. Naziv indeksa je ključ bez obzira na tip indeksa. Indeksu se pridjeljuju dionice. Indeks može sadržavati samo onu dionicu koja postoji na burzi. Ista dionica smije se samo jednom pridijeliti indeksu. Na temelju tih dionica računaju se vrijednosti indeksa za određeni trenutak. Average i Weighted indeksi na različite načine računaju svoju vrijednost. Algoritam za svaki naveden je ranije u tekstu. Rezultate izračuna potrebno je zaokružiti na 3 decimale. Dionice se u svakom trenutku mogu izbaciti iz indeksa (čime se NE brišu s burze). U tom slučaju sustav se ponaša kao da dionice nikada nisu bile u indeksu, tj. nije potrebno pamtiti povijesne podatke o vrijednostima kada su dionice još bile u indeksu.

Na burzi se prate i portfelji. Burza može sadržavati više portfelja. Priliko stvaranja portfelja definira se samo ID tog portfelja. ID portfelja je ključ te je jedinstven za svaki portfelj. Ovdje ne vrijedi pravilo slova kao kod dionica i indeksa. Portfelju se dodaje određeni broj dionica (s time da je jasno da suma broja dionica u svim portfeljima ne može biti veća od ukupnog broja izdanih dionica za svaku pojedinačnu dionicu). Iz portfelja se mogu obrisati sve dionice ili samo određeni broj. Portfelj ne može sadržavati 0 dionica te je u takvom slučaju potrebno obrisati dionicu iz portfelja. U tom slučaju vrijedi isto pravilo kao i kod indeksa (ne pamte se povijesne vrijednosti već se ponaša kao da obrisane dionice nisu nikad ni bile dio portfelja). Vrijednost portfelja računa se kao ukupna vrijednost svih dionica koje korisnik drži u tom portfelju u nekom trenutku. Osim vrijednosti portfelja računa se i postotak mjesečne promjene vrijednosti portfelja. Vrijednost se prikazuje u postocima, tj. množi se sa 100 (40% se prikazuje kao 40) i promjena vrijednost se računa u odnosu na početak mjeseca. Da bi se mogao izračunati postotak mjesečne promjene vrijednosti portfelja potrebno je imati definiranu cijenu SVIH dionica u portfelju za početni i zadnji dan u mjesecu za koji se traži ta vrijednost (uzeti vrijednost dionica prvog dana u 00:00:00 i zadnjeg dana u 23:59:59:999). Sve vrijednosti potrebno je zaokružiti na 3 decimale. Dionice se u svakom trenutku mogu izbaciti iz portfelja.