Softversko - racunarski kolokvijum

- treba da imamo i neki tekstualni fajl sa nekim sql kodom ja msm
- desni klik na solution desno i clean Solution da se klikne kad se zavrsi kolokvijum *https://www.youtube.com/watch?v=ipNFopSQr_c&t=41s* - snimak kako mladen radi racunarski

Prvi primer, treće vežbe, objašnjenje

Itemi se dodaju tako što ideš na *View* pa onda na *Toolbox*. Tu možeš da izabereš šta god želiš da dodaš, a dodavanje ide jednostavnim prevlačenjem.

Nakon toga kreiraš klase sa kojima želiš da radiš, to jest čije objekte želiš da stavljaš u taj listBox koji si kreirao. To radiš ispod i izvan public partial class koja nasledjuje *Form*.

Klase nasleđuješ preko : ukoliko treba da postaviš neke atribute iz prve klase na specifične za tu sad novu, to radiš preko konstruktora bez argumenata, samo unutar njega definišeš šta na koju vrednost treba da bude postavljeno. Override-ovanje ide kao *public override double Izra...*

Kako bi prikazao sve podatke u kreiranom list-viewu, unutar *Form1_Load* klase pozivaš metodu *GetAllItems()*, koju ćeš definisati odmah ispod te metode Form1_Load.

U ovoj metodi *GetAllItems*, kreiramo svaki objekat ponaosob koji želimo da dodamo u listu Artikala i takođe definišemo prodajnu cenu prosleđivanjem *ProdataKolicina* atributa.

Kada kreiramo objekte kreiramo i listu koja je tipa bazne klase u koju ćemo ubacivati te objekte. Možemo takođe i ovde obrisati podatke ukoliko postoje u listBox-u, pozivom metode *.Items.Clear()*

Nakon toga preko foreach petlje ispisujemo elemente iz liste. Elemente ubacujemo u listBox preko metode *.Items.Add(item.ToString())*. A ovaj toString možemo da definišemo u samoj baznoj klasi.

Drugi primer, treće vežbe, objašnjenje

U ovom primeru je bilo potrebno da unosom u textfield polja dodajemo elemente u listBox na dugme. Klasa se kreira regularno kao i u prvom primeru.

Potrebno je kreirati objekat te klase pre svih metoda kao i definisanje liste u koju ćemo sve to smeštati.

Unutar metode Form1, to jest konstruktora inicijalizacije potrebno je inicijalizovati deklarisanu listu *lista = new List<Artikal ()* inače neće raditi.

Dvoklikom na dugme na koje želimo da se podaci uzmu iz textfield-a i ubace u listBox, kreira se metoda za klik tog dugmeta. Unutar njega kreiramo objekat klase koju želimo da sačuvamo. Podatke uzimamo uz pomoć *txtName.Text* i ubacujemo u objekat klase. Ukoliko nam treba neki broj, moramo da uradimo i Convert.

Nakon toga ubacujemo taj objekat klase u listu i stavljamo *textField-ove* na prazno i pozivamo metodu za prikaz svih itema. U toj metodi postavljamo listBox na prazno preko *.Items.Clear()* i preko foreach petlje ispisujemo item-e.

Treći primer, treće vežbe, objašnjenje

Stvar koja je ovde prvi put urađena je to što je pokušaj da se podaci uzmu iz textField-ova stavljena u try-catch blok i uz pomoć *MessageBox.Show()* ispisana odgovarajuća poruka.

Ukoliko želimo da se na klik aplikacija zatvori, idemo klasika dvoklik na dugme i samo *this.Close()*

SourceTree

Skidanje projekta:
U sourcetree-u ide se na File -> Clone / New. U prvom polju kopirati HTTPS link sa github repozitorijuma, to je ono kad se klikne na zeleno dugme Code i kopira se samo taj link. Destination path odaberi sta hoces, to jest gde ces da sacuvas repozitorijum koji skidas. Name isto napisi sta oces i samo na Clone.

Grane (github i sourcetree):
Dok smo na stranici naseg repozitorijuma, ispod naziva imamo da biramo grane i pored koliko ih imamo, kada kliknemo tu, otvara se strana gde klikom na *New branch*, mozemo kreirati novu.

Kako bi videli novu granu u sourcetree-u, pritisnemo na *Fetch* u gornjem delu prozora i na Ok. Novu granu mozemo da vidimo pod *Remotes* pa *origin*. Kopiranje nove grane na lokalni remozitorijum ide desnim klikom na tu novu granu, pa *Checkout origin/Development*. Nakon toga imamo i tu novo kreiranu granu kod nas, na nasem lokalnom repozitorijumu.

Kada zelimo da kreiramo granu direkt u sourcetree programu, jednostavno idemo na Branch u gornjem meniju i kreiramo je.

Pushovanje izmena:
Kada nesto promenis u projektu, unutar SourceTree programa, imas gore levo *Commit* koji ce u trenutku promene da ima 1 kao iznad njega. Ides na to pa onda na *Stage All* sto se nalazi negde na sredini ekrana. Onda dodas neku poruku u delu za poruku u donjem delu ekrana i onda na *Commit* koje se nalazi u donjem desnom uglu. Konacno na kraju ides na *Push* koje se nalazi u gornjem levo uglu nakon *Commit* dugmeta i to je to.

Mladenov klip, objašnjenje

Ukoliko treba primarni kljuc da ti bude ono autoincrement, to ces da uradis tako sto uradis dvoklik na kolonu ili desni klik pa properties i kada ti se otvori properties, nadjes *Identity Specification* i postavis ga na true.

Kada kreiras projekat za *DataLayer* u okviru solution-a, potrebno je da odaberes *Class Library (.NET Framework) dll u zagradi*. Samo ti kreira jednu klasu, nju renamujes u *{ime}Repository*.

U okviru *DataLayer* projekta kreirati novi folder *Models*. U okviru njega Add, New item i class.
Promeniti Internal u public!!!

Povratak u repository klasu, koja na kraju izgleda ovako:

```
public class ProductRepository
{
    private string connString = "Data Source=(localdb)\\MSSQLLocalDB;Initial
Catalog=ShopDB;Integrated Security=True;Connect Timeout=30;Encrypt=False;Trust Server
Certificate=False;Application Intent=ReadWrite;Multi Subnet Failover=False";

    public List<Product> GetAllProducts()
    {

        var productList = new List<Product>();

        using (SqlConnection sqlConnection = new SqlConnection(connString))
        {
            sqlConnection.Open();

            SqlCommand command = new SqlCommand();
            command.Connection = sqlConnection;
            command.CommandText = "SELECT * FROM Products";

            SqlDataReader dataReader = command.ExecuteReader();

            while (dataReader.Read())
            {
                Product p = new Product();
                p.Id = dataReader.GetInt32(0);
                p.Name = dataReader.GetString(1);
                p.Description = dataReader.GetString(2);
                p.ExpiryDate = dataReader.GetDateTime(3);

                productList.Add(p);
            }
        }
        return productList;
    }

    public int InsertProduct(Product product)
    {
        using (SqlConnection sqlConnection = new SqlConnection(connString))
        {
            sqlConnection.Open();

            SqlCommand command = new SqlCommand();

            command.Connection = sqlConnection;
            command.CommandText = string.Format("INSERT INTO Products VALUES('{0}',
'{1}', '{2}')",
                product.Name, product.Description, product.ExpiryDate);

            return command.ExecuteNonQuery();
        }
    }
```

```
        public int DeleteProduct(int id)
        {
            using(SqlConnection sql = new SqlConnection(con))
            {
                sql.Open();
                SqlCommand cmd = new SqlCommand();
                cmd.Connection = sql;

                cmd.CommandText = "DELETE FROM Products WHERE Id=" + id;

                return cmd.ExecuteNonQuery();
            }
        }

        public int UpdateVet(Vet vet)
        {
            using (SqlConnection sql = new SqlConnection(con))
            {
                sql.Open();
                SqlCommand command = new SqlCommand();
                command.Connection = sql;

                command.CommandText = "UPDATE Vets SET FullName = @FullName, Speciality =
@Speciality, YearsExperience = @YearsExperience WHERE Id=" + vet.Id;
                command.Parameters.AddWithValue("@Id", vet.Id);
                command.Parameters.AddWithValue("@FullName", vet.FullName);
                command.Parameters.AddWithValue("@Speciality", vet.Speciality);
                command.Parameters.AddWithValue("@YearsExperience", vet.YearsExperience);
                return command.ExecuteNonQuery();
    }
}
}
```

command.ExecuteReader() se koristi kada se vrsi samo citanje, ako se radi nesto drugo sem citanja, ne ide ExecuteReader. *ExecuteNonQuery* se koristi kad se vrsi updateovanje, insertovanje ili brisanje.

Kreiramo novi projekat na isti nacin kako smo i kreirali *DataLayer* klasu samo sto se ova zove *BusinessLayer*. Klasa se zove *{ime}Business*.

```
public class ProductBusiness
{
    readonly ProductRepository productRepository = new ProductRepository();

    public List<Product> GetProducts()
    {
        return productRepository.GetAllProducts().Where(product => DateTime.Now <
product.ExpiryDate).ToList();
    }

    public bool InsertProduct(Product product)
```

```
    {
        return productRepository.InsertProduct(product) != 0;
    }

        public bool DeleteVet(int id)
        {
            return vetsRepository.DeleteVet(id) != 0;
        }

        public bool UpateVet(Vet vet)
        {
            return vetsRepository.UpdateVet(vet) != 0;
        }
}
```

Voditi racuna o metodi *GetProducts*, zato sto se ovaj uslov razlikuje u zavisnosti sta se trazi u tekstu zadatka. Takodje metoda *InsertProduct* vraca true ukoliko je poziv ove metode u return delu razlicit od 0, a false ukoliko je 0.

Na kraju na pocetnu onu formu, kreirati dizajn kakav se i trazi iz zadatka. Obicno postoji samo jedno dugme, na koje se podaci ubacuju u bazu i automatski ucitavaju iz nje i upisuju u listu.

```
public partial class Form1 : Form
{

    readonly ProductBusiness productBusiness = new ProductBusiness();
    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        RefreshList();
    }

    private void RefreshList()
    {
        listBoxProducts.Items.Clear();

        foreach(Product p in productBusiness.GetProducts())
        {
            listBoxProducts.Items.Add($"{p.Id}, {p.Name}");
        }
    }

    private void buttonInsertProduct_Click(object sender, EventArgs e)
    {
        Product product = new Product()
        {
            Name = textBoxName.Text,
            Description = textBoxDescription.Text,
```

```
            ExpiryDate = dateTimePickerRok.Value
        };

        productBusiness.InsertProduct(product);

        RefreshList();

        textBoxName.Text = string.Empty;
        textBoxDescription.Text = string.Empty;
        dateTimePickerRok.Value = DateTime.Now;
    }

        private void buttonUkloni_Click(object sender, EventArgs e)
        {
            int id = Convert.ToInt32(textBox3.Text);
                vet.DeleteVet(id);
                RefreshList();
        }

        private void buttonUpdate_Click(object sender, EventArgs e)
        {
            Vet v1 = new Vet()
            {
                Id = Convert.ToInt32(textBox3.Text),
                FullName = textBoxFullName.Text,
                Speciality = textBoxSpeciality.Text,
                YearsExperience = Convert.ToInt32(textBox1.Text)
                };

            vet.UpateVet(v1);

            RefreshList();

            textBox3.Text = string.Empty;
            textBoxFullName.Text = string.Empty;
            textBoxSpeciality.Text = string.Empty;
            textBox1.Text = string.Empty;
        }
}
```

Potrebno je pored ovog koda, dodati i reference od prethodno kreiranih klasa, tako sto ce se otici u *Solution Explorer*, odabrati projekat gde se nalazi dizajn za aplikaciju, desni klik na references i add reference.

Na kraju, kada zavrsis sve, potrebno je kod za bazu podataka i podatke staviti u neki tekstualni fajl. Potreban tekst od baze ces da dobijes kad ides desni klik na tabelu i *View Code* a kod za unos podataka ces da dobijes kad odes *View Data* i onda pored broja od *Max Rows* na papiric prvi.