.NET platforma

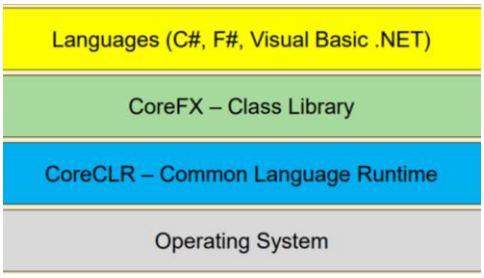
Pojam .NET

- Platforma za razvoj veb, desktop, mobilnih i drugih tipova aplikacija
  - Skup programa, alata i jezika za izvršavanje i razvoj korisničkih aplikacija
  - Besplatna
  - Otvorenog koda
  - Platformski-nezavisna – izvršava se na Linuxu, Windowsu, macOS operativnim sistemima

Istorijat .NET

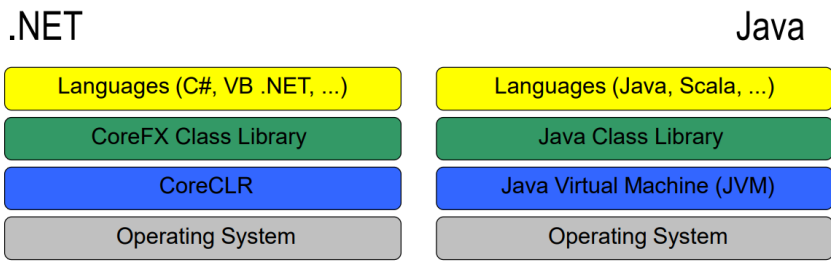
- Razvijena od strane Microsoft korporacije
- Verzija 1.0 se pojavila 2002.
- Inicijalno se zasnivala na softverskom radnom okviru .NET framework
  - Namenjen prvenstveno za Windows operativne sisteme
- Njegov naslednik je platformski nezavisan radni okvir nazvan samo .NET
  - U prvim verzijama se zvao .NET Core
  - Trenutna verzija .NET je .NET 6

Struktura .NET platforme



- Jezici – bilo koji jezik kompatibilan sa CLS specifikacijom (Common Language Specification)
- CoreFX – biblioteka gotovih klasa koje sadrže često korišćene funkcionalnosti
- Core Common Language Runtime (CoreCLR) – virtuelna mašina na kojoj se izvršavaju .NET programi

.NET i Java



Common Language Runtime (CLR)

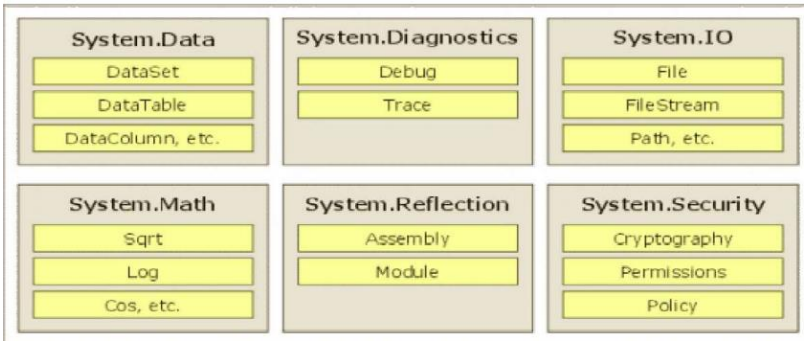
- Svi .NET programi se izvršavaju u okviru CLR
- CLR je sloj između aplikacije i operativnog sistema
- CLR obezbeđuje
  - Upravljanje memorijom
  - Izvršavanje programskih niti
  - Garbage Collection mehanizam
  - Obradu izuzetaka
  - Bezbednosni model

- Verzioniranje (korišćenje odgovarajuće verzije svake komponente)
- …

CoreFX Class Library

- Biblioteka koja sadrži programske entitete (klase, interfejse, ...) sa često korišćenim funkcionalnostima (rad sa fajlovima, grafikom, bazom podataka, ...)
- Programer se oslabađa implementacije značajnog dela programskog koda
- Jezički-neutralna – može se koristiti u bilo kojem jeziku kompatibilnom sa CLS specifikacijom
- Organizovana kao stablo, podeljena u prostore imena (namespace)
- Prostor imena grupiše srodne entitete i omogućuje jedinstvenu identifikaciju entiteta

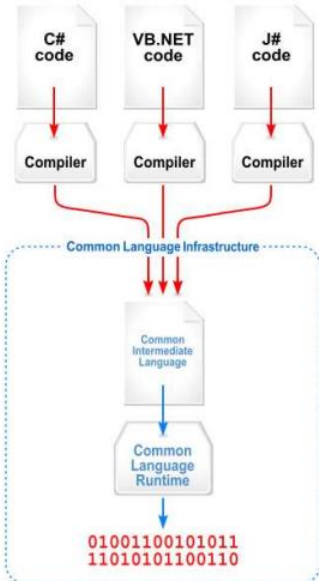
FCL – Prostori imena



Common Language Specification (CLS)

- Specifikacija Common Type System (CTS) definiše tipove podataka kojima CLR može upravljati i načine upravljanja ovim tipovima
- Common Language Specification (CLS) je podskup CTS i ovaj podskup je podržan u svakom .NET jeziku
- CLS obezbeđuje interakciju delova programskog koda pisanih u različitim .NET jezicima
- Svaki jezik može sadržati proizvoljan podskup CTS, ali javno izložene funkcije moraju biti u skladu sa CLS

Prevođenje i izvršavanje .NET programa



- Izvorni kod se prevodi u međukod (MSIL – Microsoft Intermediate Language)
- Međukod se izvršava u okviru CLR
- CLR sadrži JIT (Just In Time) kompajler, koji u toku izvršavanja prevodi međukod u mašinski kod
- Računar na kojem se program izvršava mora sadržati CLR
- CLR se distribuira zajedno sa programom ili se posebno instalira, ako ga OS ne sadrži

.NET izvršni sklop (Assembly)

- Sklop je osnovna logička jedinica za distribuciju .NET programa
- Sklop sadrži module i resurse
- Modul sadrži međukod koji se izvršava u okviru CLR i može imati ugrađene interne resurse
- Resursi su dodatni fajlovi koji se koriste u programu (slika, zvučni zapis, ...)
- Jedan od modula u sklopu mora da sadrži manifest faj
- Manifest putem metapodataka opisuje sadržaj sklopa
- Sklop se distribuira kao EXE ili DLL fajl
- Sklop najčešće sadrži samo jedan modul, jer Visual Studio (najčešće razvojno okruženje za .NET programe) ne podržava sklopove sa više modula

Uvod u C#

Istorija

- Razvijen u Microsoft-u 2002. godine
- Vođa projekta Anders Hejlsberg
- Razvoj jezika započet 1999. (radni naziv “Cool”)
- Kreiran na tragu C++ i Java jezika
- Jezik publikovan 2002.
- Standardizovan od strane ECMA i ISO/IEC
- Trenutna verzija 9.0 (iz novembra 2020)

Generalne osobine jezika

- Jezik opšte namene
- Objedinjuje više programskih paradigmi
  - Stroga tipiziranost
  - Strukturiranost
  - Objektna orijentisanost
  - Imperativnost
  - Deklarativnost
  - Generičnost
- Slobodan format kucanja programskog koda

Identifikatori

- Dozvoljeno je koristiti
  - Velika i mala slova
  - Cifre (identifikator ne sme počinjati cifrom)
  - Karakter _
- Kao identifikator se ne mogu koristiti rezervisane reči:

abstract	do	in	protected	true
as	double	int	public	try
base	else	interface	readonly	typeof
bool	enum	internal	ref	uint
break	event	is	return	ulong
byte	explicit	lock	sbyte	unchecked
case	extern	long	sealed	unsafe
catch	false	namespace	short	ushort
char	finally	new	sizeof	using
checked	fixed	null	stackalloc	virtual
class	float	object	static	void
const	for	operator	string	volatile
continue	foreach	out	struct	while
decimal	goto	override	switch	
default	if	params	this	
delegate	implicit	private	throw	

Promenljive

- Lokacija za skladištenje određene vrednosti
- Konvencija imenovanja
  - camelCase notacija
    -- Malo početno slovo, svaka nova reč počinje velikim slovom
- Mora biti promenljiva deklarisana pre korišćenja
  - Navode se tip i naziv promenljive

- int a;
- Nije moguće koristiti promenljivu pre nego što joj se dodeli i vrednost (definite assignment rule)
- Kompajler će prijaviti grešku

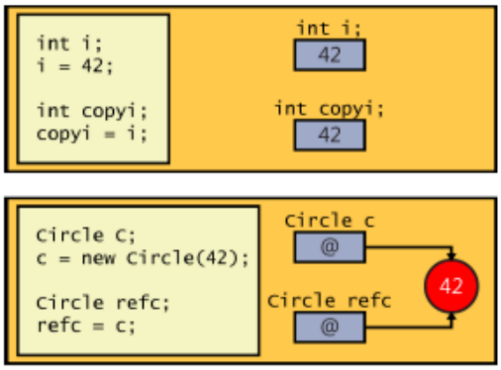
Dodela vrednosti

- Znak =
  cena = 78.99

Tipovi podataka

- Vrednosni tipovi
- instance se skladište u stek memoriji
- Adresni tipovi
- adrese instanci se skladište u stek memoriji
- instance se skladište u heap memoriji
- Garbage Collection
- oslobađanje memorije koju zauzimaju nekorišćeni objekti
- mehanizam je pod kontrolom CLR i automatski se aktivira

Reprezentacija u memoriji



Vrednosni tipovi podataka

- Primitivni tipovi

Data type	Description	Size (bits)	Range	Sample usage
int	Whole numbers (integers)	32	-2^{31} through $2^{31} - 1$	int count; count = 42;
long	Whole numbers (bigger range)	64	-2^{63} through $2^{63} - 1$	long wait; wait = 42L;
float	Floating-point numbers	32	$\pm 1.5 \times 10^{-45}$ through $\pm 3.4 \times 10^{38}$	float away; away = 0.42F;
double	Double-precision (more accurate) floating-point numbers	64	$\pm 5.0 \times 10^{-324}$ through $\pm 1.7 \times 10^{308}$	double trouble; trouble = 0.42;
decimal	Monetary values	128	28 significant figures	decimal coin; coin = 0.42M;
char	Single character	16	0 through $2^{16} - 1$	char grill; grill = 'x';
bool	Boolean	8	True or false	bool teeth; teeth = false;

Vrednosni tipovi podataka

- Enumeracija
- Promenljive ovog tipa mogu dobiti samo vrednost iz skupa diskretnih vrednosti koje enumeracija predviđa

- Npr. promenljiva koja predstavlja dan u nedelji

```
enum Dan {Ponedeljak, Utorak, Sreda, Cetvrtak, Petak};
Dan d = Dan.Petak;
```

- Moglo bi se rešiti i korišćenjem int promenljivih, ali bi
  -- kod bio manje čitak
  -- promenljiva bi mogla da dobije vrednost bilo kojeg celog broja (ne bi bila ograničena na samo nekoliko mogućih vrednosti)

Tipovi podataka

- Enumeracija
- Vrednosti iz enumeracije se mapiraju na brojeve (počevši od nula)

```
Dan d = Dan.Petak;
int br = (int) Dan.Petak;       //br je 4
```

- Može i eksplicitno da se odredi broj na koji se vrednost mapira

```
enum Dan {Ponedeljak = 3, Utorak = 4, Sreda = 5, Cetvrtak = 6, Petak = 7};
```

- Ako se ne odredi broj, uzima se prethodni uvećan za 1

```
enum Dan {Ponedeljak = 3, Utorak, Sreda, Cetvrtak, Petak};
```

Vrednosni tipovi podataka

- Struktura
- Sadrži grupisane podatke i operacije (metode) zajedničke za jedan entitet
- Kao klasa, samo što je vrednosni tip koji se skladišti na steku

```
public struct Test {
public int a, b;
}
Test t;
 t.a = 10;
 t.b = 5;
```

Tipovi podataka

- Adresni tipovi
- klase, interfejsi, nizovi, delegati
- Vrednosti ovog tipa se skladište u heap memoriji
- U stek memoriji se skladišti adresa iz heap memorije na kojoj je promenljiva uskladištena
- Svi složeni tipovi izvedeni od tipa System.Object sa metodama
- ToString
- Equals
- GetHashCode
- Finalize

Automatsko određivanje tipa

- Kompajler može da automatski odredi tip promenljive na osnovu vrednosti koja se dodeljuje

```
var x = 5;
```

- x će biti tipa int nakon gornjeg izraza
- Ako se koristi var, mora se odmah izvršiti dodela vrednosti

Konverzije tipova

- Implicitna konverzija u situacijama kada nema gubljenja podataka
  int a = 10;
  double b = a;
- Eksplicitna konverzija ako može doći do gubitka podataka
  double b = 10;
  int a = (int) b;

String

- Niz karaktera je reprezentovan klasom System.String
- Može se koristiti i alias string
- Adresni tip
- Operatori == i != kreirani tako da porede vrednosti, a ne adrese
- Neizmenjiv (pri svakoj dodeli vrednosti zauzima se nova memorija)
- Operator [] za preuzimanje karaktera na određenoj poziciji
- Postavljanje sadržaja stringa:
  - string s = ”uvod”;
  - string s = “uvod\tstringovi”;
  - string s = “c:\\temp\\test”
  - string s = @”c:\temp\test”;
- Dužina stringa
  - s.Length
- Spajanje stringova
  - string s1 = “uvod”;
  - string s2 = “stringovi”;
  - string s = s1 + “ “ + s2;
- Metode
  - ToLower, ToUpper
  - Replace, Insert, Remove
  - Substring
  - Split
  - Trim, TrimLeft, TrimRight

Aritmetički operatori

- Aritmetički operatori
  - Standardni aritmetički operatori +, -, /, *
  - Operator % za ostatak pri deljenju
  - ++ i -- prefiksni i postfiksni operatori za inkrementiranje i dekrementiranje
- Aritmetički sa dodelom vrednosti
  - +=, -=, *=, /=
- Postoji posebna vrednost Infinity za decimalni broj koji predstavlja beskonačno
- Postoji posebna vrednost NaN za decimalni broj koji predstavlja nevalidnu vrednost

Relacioni operatori

- Porede dve vrednosti
- Rezultat je logička vrednost (boolean)

Operator	Meaning	Example	Outcome if age is 42
<	Less than	age < 21	false
<=	Less than or equal to	age <= 18	false
>	Greater than	age > 16	true
>=	Greater than or equal to	age >= 30	true

Logički operatori

- Negacija !
- Logičko „i“ (konjukcija) &
- Logičko „ili“ (disjunkcija) |
- Ekskluzivno „ili” ^
- Skraćeno izračunavanje
  - Konjukcija je netačna ako je prvi operand netačan
  - Disjunkcija je tačna ako je prvi operand tačan
  - Tada nema potrebe proveravati vrednost drugog operanda
  - Uobičajeno se koriste skraćene varijante čime se izbegavaju nepotrebna izračunavanja
- Skraćena konjukcija &&
- Skraćena disjunkcija ||

Operatori nad bitovima

- Negacija ~
- Konjukcija &
- Disjunkcija |
- Ekskluzivno ili ^
- Pomeranje bitova ulevo <<
- Pomeranje bitova udesno >>

Ostali operatori

- Ternarni operator za uslov
  - U zavisnosti od istinitosti prvog operanda (logički izraz) vraća vrednost drugog ili trećeg operanda
  - x ? y : z
- Podrazumevana vrednost tipa
  - default (0 za vrednosne tipove, null za adresne)

if - else

- if – else

```
if (domaci > gosti)
        ishod = "1";
else if (domaci < gosti)
         ishod = "2";
else ishod = "x";
```

switch

- Zavisno od vrednosti izraza u switch naredbi, izvršava se deo koda u okviru odgovarajuće case naredbe
- ako vrednost ne odgovara nijednoj case naredbi, izvršava se kod definisan u okviru labele default (ako postoji)
- Nema automatskog prelaska u narednu case naredbu
- svaki case se mora završiti naredbom skoka – break (najčešće) ili goto

```
switch (plasman) {
 case 1:
        medalja = "zlato";
        break;
case 2:
         medalja = "srebro";
          break;
case 3:
         medalja = "bronza";
           break;
default:
           medalja = "bez medalje";
           break;
 }
```

Ciklusi - for

```
for (int i = 0; i < tekst.Length; i++)
{
         Console.WriteLine(tekst[i]);
}
```

Ciklusi – while, do - while

```
while (broj > 20)
 {
        Console.WriteLine(broj);
            broj /= 2;
} do {
        Console.WriteLine(broj);
        broj /= 2;
} while (broj > 20);
```

- Razlika – while može da se ne izvrši, do-while se izvršava bar jednom

Ciklusi – break, continue

- break – izlazak iz tekućeg ciklusa
- continue – prekid tekuće iteracije i prelazak na sledeću iteraciju

```
for (int i = 0; i < tekst.Length; ++i) {
        if (tekst[i] == ' ')
                break;
        else if (tekst[i] != 'a')
                continue;
       brojSlovaA++;
}
```

Ciklusi – foreach

- Iteriranje kroz elemente kolekcije
  - foreach (<tip> <identifikator> in <kolekcija>)

```
foreach (char c in tekst) {
        Console.WriteLine(c);
}
```

Komentari

- Jednolinijski //tekst
- Višelinijski /* tekst */
- Mogućnost automatskog generisanja dokumentacije na osnovu specijalnih komentara
- Jednolinijski dokumentacijski komentar ///tekst
- Višelinijski dokumentacijski komentar /** tekst */
- Tagovi za dokumentacijske komentare
  - <remarks>, <summary>, <example>, <exception>, <param>, <permission>, <returns>, <seealso>, <include>

Objektno programiranje u C# jeziku

Objektno programiranje

- Identifikuje entitete sa kojima program operiše
- Za entitete se identifikuju
  - podaci koji se evidentiraju
  - operacije koje se mogu izvršavati nad entitetom
- Apstrakcija
  - Zanemaruju se osobine entiteta koje nisu važne za konkretan problem
- Klasifikacija
  - Entiteti se grupišu po tipu
  - Time su identifikovane klase entiteta
  - Pojedinačni entiteti su primerci svoje klase

Objekat vs Klasa

- Objekat je konkretan entitet
  - Ima podatke koji opisuju njegove osobine
  - Nad tim podacima se može izvršiti programski kod
- Klasa je šablon koji opisuje sve entitete određenog tipa i definiše
  - Koje osobine svi entiteti tog tipa imaju
    -- To su atributi klase
  - Programski kod koji se može izvršiti nad entitetima tog tipa
    -- To su metode klase
- Program u izvršavanju radi sa objektima
  - Objekat ima svoj životni ciklus
  - U toku životnog ciklusa
    -- skladišti i menja podatke
    -- izvršava se programski kod nad podacima
- Klasa je apstrakcija da predstavimo koje podatke skladištimo i koje operacije izvršavamo nad objektima određenog tipa
  - Objekat je primerak (instanca) klase

C# Klase

- Deklaracija klase
  class Krug {

```
//sadržaj klase
}
```

- Ime klase može biti različito od imena fajla
- Jedan fajl može sadržati više klasa
- Instanciranje klase
  - Krug k = new Krug();

C# Članovi klase – atributi i metode

- Atributi
- Metode
- Svojstva
- Ugrađene klase

C# Atributi klase

- private double poluprecnik;
- Atribut se može inicijalizovati pri deklaraciji
  - private Tacka centar = new Tacka(0,0);
  - Kompajler automatski dodaje kod za inicijalizaciju na početak konstruktora

C# Metode klase

```
public String ispisiIme(bool malaSlova) {
        if (malaSlova)
                return ime.ToLower();
        else
               return ime;
}
```

Preklapanje metoda (overloading)

- Može da postoji više metoda istog imena, ali sa različitim argumentima

```
public double ispisiIme(bool malaSlova)
{ …
 }
public double ispisiIme()
 { ...
}
```

- Ne može da bude više metoda sa istim imenom i argumentima, ali različitog tipa povratne vrednosti

C# Svojstva (Properties)

- Enkapsulacija
  - program koji koristi klasu ne treba da poznaje detalje implementacije te klase
  - atributi ne trebaju direktno da budu dostupni spolja
- Standardan mehanizam enkapsulacije – get/set metode
- Properties
  - obezbeđuju efikasniji mehanizam enkapsulacije
  - get/set metode koje se koriste kao atributi

```
private double x;
 public double X {
        get { return x; }
```

```
        set { x = value; }
    }
```

- promenljiva value sadrži vrednost koja je prosleđena i koja se postavlja u atribut
- Atribut može biti read-only ili write-only ako se implementira samo get ili set deo koda
- Property je metoda sa specifičnom sintaksom, što znači da može da sadrži proizvoljan kod
- Automatski property – jednostavna sintaksa za slučaj kada se samo postavlja, odnosno preuzima vrednost atributa
- public double X { get ; set; }
- pristup property elementu

```
Tacka t = new Tacka();
t.X = 5.2;
```

Prostori imena (namespaces)

- Tipovi su organizovani u prostore imena zbog
- jedinstvene identifikacije tipova
- organizacije tipova po srodnosti
- Slično kao paketi u Javi, ali .NET prostori imena predstavljaju logičku strukturu koja ne mora biti u skladu sa strukturom fajlova i foldera
- Jedinstvena identifikacija tipa
- Prostor imena + naziv tipa

```
System.Collections.ArrayList l = new System.Collections.ArrayList();
```

- Ovakvo korišćenje punog imena bi smanjilo čitljivost koda, zato se koristi ključna reč using

```
using System.Collections;
ArrayList l = new ArrayList();
```

Nasleđivanje

- Različite klase mogu da dele deo osobina i operacija
- Zbog iskorišćenja koda zajednički atributi i metode se mogu grupisati
- Sve klase u C# implicitno nasleđuju klasu System.Object sa metodama
- ToString() – konvertuje sadržaj objekta u tekst
- Equals(Object) – poredi sadržaj objekta sa drugim objektom
- GetHashCode() – vraća hash code objekta (sadržaj objekta mapiran na numeričku vrednost)
- GetType() – Vraća Type objekat koji opisuje klasu kojoj objekat pripada
- Sintaksa – koristi se karakter :

```
public class Osoba {
        protected String ime;
        protected String prezime;
}
public class Radnik : Osoba {
        protected String radnoMesto;
        public void obracunajPlatu() {…}
}
```

Polimorfizam

- Objekat se može posmatrati kao da je višestrukog tipa (polimorfno)
- Implicitna konverzija – iz nasleđenog tipa u bazni tip

```
Student s1 = new Student();
```

s1.Prosek = 9.3;
Object obj = s1;

- U ovom slučaju promenljiva tipa bazna klasa može da pristupi (bez eksplicitne konverzije) samo članovima definisanim u baznoj klasi
  -- obj.ToString() – ispravno
  -- obj.Prosek – greška pri kompajliranju
- Eksplicitna konverzija – iz baznog tipa u nasleđeni tip
- Student s2 = (Student) obj;
- Moguć izuzetak u toku izvršavanja (InvalidCastException), ako instanca nije odgovarajućeg tipa
- Racunar r = (Racunar) obj; - IZUZETAK! (U toku kompajliranja nije poznat tip instance na koju pokazuje promenljiva obj. U toku izvršavanja desiće se izuzetak, jer promenljiva tipa Racunar ne može pokazivati na instancu tipa Student (na koju pokazuje obj).

Nasleđivanje i konverzija tipova

- Pojava run-time izuzetka pri konverziji, može se sprečiti operatorima:
- is - utvrđuje tip instance. Najpre provera tipa, pa zatim konverzija

```
if (obj is Racunar)
    Racunar r = (Racunar) obj;
```

- as – istovremeno provera i konverzija. Ukoliko konverzija nije moguća, operator vraća vrednost null

```
Racunar r = obj as Racunar;
```

Dynamic binding

- Ključna prednost objektnog programiranja
- Bez dynamic binding-a objektno programiranje bi bilo samo način organizacije koda
- Sama modularizacija koda je i u proceduralnom programiranju moguća kroz odvajanje srodnih promenljivih i funkcija u odvojene celine/fajlove
- Omogućuje nezavisno i naknadno proširivanje funkcionalnosti bez izmene postojećeg koda
- Ako postoji jedna osnovna klasa koju nasleđuje više klasa naslednica
- Dynamic binding predstavlja sposobnost programskog jezika da tretira objekte naslednica zavisno od njihovog tipa
- Programski kod koji radi sa objektima je generički, radi sa osnovnom klasom i ne sadrži kod specifičan za klase naslednice
- U toku izvršavanja programa (run-time), poziva se odgovarajuća metoda klase naslednice zavisno od tipa objekta
- dynamic (late, run-time) binding

```
List<osnovna> l;
l.add(new Naslednica1());
l.add(new Naslednica2());

foreach (Osnovna x in l) {        --> Ovaj deo koda uvek ostaje isti
    x.izvrsi();                   --> Poziva se metoda izvrsi iz klase Naslednica1 ili Naslednica2 zavisno od tipa objekta x
}
```

- Na taj način kod koji radi sa osnovnom klasom ne treba da se menja
- Samo se u klasama naslednicama definiše novo ponašanje

Virtuelne metode i redefinisanje metoda (overriding)

- Metoda pretka čije ponašanje može biti u klasi naslednici izmenjeno (redefinisano) metodom koja ima istu deklaraciju

- To je preduslov da bi dynamic binding funkcionisao
  - Klasa predak treba da ima definisanu metodu
  - Klasa naslednica ima istu tu metodu sa drugačijim ponašanjem
  - U klasi predak metoda mora biti definisana kao virtuelna
- Virtualna metoda se deklariše sa virtual
  - Za razliku od Jave gde su sve metode implicitno virtualne
- Redefinisana metoda se deklariše sa override
  - Ako se ne stavi override, metoda se ne smatra redefinicijom metode pretka nego kao nova metoda sa istom deklaracijom
  - Kompajler prikazuje upozorenje u ovom slučaju
  - Rečju new može se reći kompajleru da je ovo svesno urađeno
  - Virtuelna metoda ne može biti private
  - Metode i redefinicije moraju imati potpuno istu deklaraciju i modifikator pristupa

Apstraktne klase i metode

- Ne može biti instancirana
- Služe kao osnov za nasleđivanje
- Sintaksa
  public abstract class Figura {
  … }
- Može da sadrži apstraktne metode
  - metoda deklarisana bez implementacije
  - klase naslednice zadužene da implementiraju telo metode
  - sintaksa public abstract double getPovrsina();

Interfejsi

- specifikacija funkcionalnosti koje određeni objekat podržava, bez implementacije ovih funkcionalnosti
- implementaciju funkcionalnosti definisana je u klasi koja implementira interfejs
- Klasa može da implementira više interfejsa (ali može da nasledi samo jednu klasu)
- u sintaksnom smislu interfejs je apstraktna klasa koja sadrži samo javne apstraktne metode
- Deklariše se rečju interface
- Ispred metoda nema modifikatora pristupa, jer su uvek javne

Interfejsi - sintaksa

```
interface IPoredjenje {
        int poredi(Object obj);
} …
class Student: IPoredjenje {
       int poredi(Object obj) {
       … }
}
```

- Interfejs ne sme imati atribute, čak ni statičke
- Nema konstruktora
- Nema ugrađenih tipova i klasa unutar interfejsa
- Ne može da nasledi klasu, ali može da implementira drugi interfejs

Statičke klase i članovi klase

- Statički članovi klase

- funkcionalnosti koje nisu vezane za neku konkretnu instancu klase
- može im se pristupati bez kreiranja objekta klase

```
class Math {
        …
        public static double Sqrt(double d) { … }
}
…
double v = Math.Sqrt(2);
```

- u statičkoj metodi moguće je pristupati samo statičkim atributima i pozivati samo statičke metode
- konstanta je statički član koji se ne može menjati public const double PI = 3.14159265358979323846

- Statička klasa
- sadrži samo statičke atribute i metode
- ne može da se instancira

```
public static class Convert
```

Modifikatori pristupa

- Za klasu
- internal – podrazumevani modifikator, pristup klasi moguć iz drugih klasa unutar istog sklopa (eng. assembly)
- public – pristup klasi moguć iz svih drugih klasa
- ugrađene klase mogu imati modifikatore pristupa kao atributi
- Za atribut
- private – podrazumevani modifikator, atribut dostupan samo unutar klase
- protected – atribut dostupan unutar klase i iz klasa naslednica
- internal – atribut dostupan iz klasa unutar istog sklopa (eng. assembly)
- protected internal – atribut dostupan iz klasa unutar istog sklopa i iz klasa naslednica
- public – atribut dostupan iz svih klasa

Zapečaćene (sealed) klase i metode

- Zapečaćena klasa je klasa koju nije moguće naslediti
- Klasa eksplicitno zabranjuje nasleđivanje rečju sealed u deklaraciji

```
sealed class A { …
```

- Zapečaćena metoda je redefinisana metoda koju nije dalje moguće redefinisati

```
public sealed override int m1() { … }
```

- sealed se odnosi samo na redefinisane metode, jer za obične metode izostavljanjem reči virtual moguće je zabraniti njenu redefiniciju

Ugrađene klase

- Klasa definisana unutar neke klase
- Koristi se kada se klasa koristi samo za realizaciju funkcionalnosti klase unutar koje je definisana
- Najčešće spolja nije vidljiva i nije namenjena za korišćenje iz drugih delova koda
- Zato je default modifikator private
- Mogu se staviti i drugi modifikatori

```
class Glavna {
    class Unutrasnja {
        ... }
    ... }
```

Anonimne klase

- Moguće je definisati novu klasu u trenutku kreiranja objekta te klase
  - Ova klasa nema ime
  - Pri kreiranju objekta navode se svojstva klase i vrednosti svojstava u objektu

```
var a = new {Cena = 450.37, Naziv = “TV”};
a.Cena = 500.99;
```

Operatori nad objektima

- Naziv tipa
  - typeof
- Tip objekta
  - is
- Konverzija tipa (cast)
  - as
- Može se vršiti preklapanje operatora

Metode koje proširuju klasu Extension Methods

- Moguće je naknadno proširiti klasu metodama bez izmene izvornog koda klase
- Koristi se za klase preuzete iz postojećih biblioteka
- Time nad objektima klase možemo pozivati naknadno dodate metode
  - Bez Extension Methods, jedina varijanta za uvođenje novih metoda je nasleđivanje postojeće klase, ali se onda svugde mora koristiti klasa naslednica
- Sintaksa
  - Npr. proširujemo klasu String novom metodom koja izbacuje sva prazna mesta iz sredine stringa
  - Uvodimo novu klasu u kojoj ćemo definisati metodu

```
static class StringProsirenje {
    public static string TrimMiddle(this string s) {
        return s.Replace(" ", "");
    }
}
```

```
string s = "Marko Markovic Novi Sad";
Console.WriteLine("Spojeno sve: " + s.TrimMiddle());
```

- Nakon definisanja metode koja proširuje klasu, kompajler tretira ovu metodu isto kao i druge metode koje su definisane u samoj klasi

C# Windows Presentation Foundation – WPF

Windows Presentation Foundation (WPF)

- WPF je tehnologija za razvoj korisničkog interfejsa u .NET aplikacijama
- Pojavila se 2006. godine u okviru .NET Framework 3.0
- Oslanja se na DirectX
- Namenjena i desktop i internet aplikacijama

Struktura

- Razdvojeni prikaz i ponašanje programa
- Prikaz se definiše u XML-baziranom jeziku XAML

```
<Window xmlns="http://schemas.microsoft.com/win
    fx/2006/xaml/presentation“
```

x:Class="TestWpf.MainWindow” Title=“Main" Width="250” Height="100">
<Button Name=“btnLogin">Login</Button>
</Window>

- Ponašanje se definiše u programskom kodu

```
public partial class MainWindow : Window {
    public MainWindow() { I
        nitializeComponent();
    }
}
```

Klasa Application

- Ulazna tačka programa
- Inicijalizuje aplikaciju
- Definiše globalne resurse koji se mogu koristiti iz svih delova programa
- Prikazuje startnu formu

<Application x:Class=“HelloWorld.App"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
StartupUri="MainWindow.xaml">
<Application.Resources>
</Application.Resources>
</Application>

Forme

- Predstavljene klasom System.Windows.Window
- Properties
  - Title, Icon, Width, Height, ResizeMode, WindowState, Background, ...
- Events
  - Activated, Closed, GotFocus, KeyDown, KeyUp, Loaded, MouseDown, MouseEnter, MouseLeave, …

Komponente interfejsa

- Prikaz teksta
- Dugmad
- Polja za unos
- Raspoređivanje elemenata
- Grupisanje elemenata
- Prikaz podataka
- Meniji
- Dijalozi
- Ostali elementi

- System.Windows.Controls prostor imena
- Prikaz teksta
  - TextBlock

-- Može se definisati prikaz teksta u više redova – TextWrapping
<TextBlock Height=“40" Name=“tbOpis"
Text=“Osnovne komponente interfejsa"
TextWrapping="Wrap" />

- Label
  -- Može u sebi da sadrži druge komponente i da prikazuje proizvoljan sadržaj
  <Label Content=“Ime" Height="28"
  Name=“lbIme" />

Komponente interfejsa - Dugme

- Dugme - Button
- Klik na dugme izvršava metodu čiji je naziv postavljen u atributu Click
- Može sadržati druge komponente – npr. može se definisati dugme koje sadrži sliku ili neki multimedijalni sadržaj
- ToolTip – tekst koji dodatno objašnjava ulogu dugmeta - pojavljuje se kada se miš postavi na dugme
  <Button Content=“Prijava" Name=“btnPrijava"
  Height="23" Width="75“
  Click="btnPrijava_Click“/>
- Znak _ ispred slova u imenu označava prečicu za aktiviranje

Komponente interfejsa – TextBox

- TextBox - Polje za unos teksta
- Prikazuje neformatirani tekst
- Atribut Text sadrži trenutni tekst u polju
- Događaj TextChanged za reakciju na promenu teksta
- Poravnanje sadržaja u polju –
  HorizontalContentAlignment, VerticalContentAlignment
- Prikaz scroll bar komponenti unutar polja –
- VerticalScrollBarVisibility
- HorizontalScrollBarVisibility
- Definisanje komponente u XAML fajlu
  <TextBox Height="23" Width="120" Name="tbIme" Text=“Petar“
  TextChanged=“tbIme_TextChanged”/>
- Rad sa komponentom u programskom kodu
  string ime = tbIme.Text;

Komponente interfejsa - RichTextBox

- RichTextBox - Unos i prikaz formatiranog dokumenta koji može da sadrži tekst, paragrafe, slike, tabele, ...
- Sadrži jedan element u sebi tipa FlowDocument
- FlowDocument reprezentuje strukturu i sadržaj dokumenta
  <RichTextBox Height=“500" Width=“200" Name="rtb_Biografija"/>

Komponente interfejsa - PasswordBox

- PasswordBox - Polje za unos šifre
- Korisniku se umesto unesenog teksta prikazuje znak definisan atributom PasswordChar
- Tekst u polju se može programski preuzeti iz atributa Password
  <PasswordBox Height=“23” Width="120" Name=“pbSifra“ PasswordChar="*" />
  string sifra = pbSifra.Password;

Komponente interfejsa - CheckBox

- CheckBox - Polje za izbor opcije
- Izbor definisan atributom IsChecked
- Tekst u polju definisan atributom Content (osim teksta, CheckBox može sadržati proizvoljnu kontrolu)

- Prikaz teksta sa leve ili desne strane polja – FlowDirection (LeftToRight ili RightToLeft)
- Može sadržati tri stanja – IsThreeState
  <CheckBox Content=“Prihvatam uslove" Name="cbUslovi“/>

Komponente interfejsa - ComboBox

- ComboBox - Polje za prikaz padajuće liste
- Sadrži kolekciju objekata bilo kojeg tipa
- Items – objekti u kolekciji
- SelectedIndex – indeks selektovanog elementa
- SelectedItem – referenca na selektovani element
- DisplayMemberPath – određuje koji atribut elementa kolekcije se prikazuje u ComboBox kontroli
- SelectedValue – selektovana vrednost
- SelectedValuePath – određuje atribut elementa kolekcije, čija vrednost se dobija pozivom SelectedValue svojstva
  <ComboBox Name=“cbStudenti" Width="120" SelectedIndex="0">
  Student s1 = new Student(“Petar”, “Petrovic”);
  Student s2 = new Student(“Marko”, “Markovic”);
  cbStudenti.Items.Add(s1);
  cbStudenti.Items.Add(s2);
  cbStudenti.DisplayMemberPath = "Prezime";
  cbStudenti.SelectedValuePath = "Ime";
  Student selStudent = cbStudenti.SelectedItem;
  string selIme = cbStudenti.SelectedValue;

Komponente interfejsa - RadioButton

- RadioButton - Izbor samo jedne od više opcija
- Izbor definisan atributom IsChecked
- GroupName – grupa kojoj RadioButton pripada. Dugmad iz iste grupe se međusobno isključuju pri izboru opcije
- Content – Tekst pored dugmeta
  <RadioButton Content=“Pol - muški" Name=“rbPolMuski“ GroupName="pol" IsChecked="True" />

Komponente interfejsa – ListBox

- ListBox - Prikaz kolekcije elemenata
- Slične funkcionalnosti kao ComboBox
- Može se selektovati više od jednog elementa
- SelectionMode
  - Extended – selektuje se više elemenata držeći taster Shift
  - Multiple – selektuje se više elemenata bez korišćenja tastera Shift
  - Single – selektuje se samo jedan element
- SelectedItems – lista svih selektovanih elemenata
  <ListBox Height="126" Width="254“ Name="lbSpisak“ SelectionMode="Multiple“/>

Komponente interfejsa - Grid

- Grid - Panel za raspoređivanje elemenata
- Elementi se raspoređuju u redove i kolone (inicijalno samo jedna ćelija - jedan red i jedna kolona)
- Definisanje redova
  <Grid><Grid.RowDefinitions>
  <RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />
<RowDefinition Height="*" />
<RowDefinition Height="28" /> </Grid.RowDefinitions>

- Definisanje kolona
  <Grid.ColumnDefinitions>
  <ColumnDefinition Width="1*" />
  <ColumnDefinition Width="3*" /></Grid.ColumnDefinitions>
- Veličina redova/kolona
  - Fiksna vrednost
  - Auto – zavisno od veličine kontrola koje se nalaze u redu/koloni
  - * - proporcionalno raspoređivanje prostora između svih redova/kolona označenih zvezdicom. Npr. u prikazanom primeru, prva kolona će zauzeti jednu četvrtinu, a druga kolona tri četvrtine grida
- Raspoređivanje elemenata u Grid kontroli
  - Grid.Row, Grid.Column, Grid.RowSpan, Grid.ColumnSpan
    <Grid>
    <Grid.RowDefinitions>
    <RowDefinition /> <RowDefinition />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
    <ColumnDefinition /> <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <Label Grid.Row="0" Grid.ColumnSpan="2" Content="Unos podataka o studentu"/>
    <Label Grid.Row="1" Grid.Column="0" Content="Ime:"/>
    <TextBox Grid.Row="1" Grid.Column="1" />

Komponente interfejsa - Canvas

- Canvas - Panel za raspoređivanje elemenata na određene koordinate
- Relativno u odnosu na poziciju Canvas kontrole
  - Canvas.Left – udaljenost od leve ivice Canvas kontrole
  - Canvas.Top – udaljenost od vrha Canvas kontrole
    <Canvas><Label Canvas.Left="5" Canvas.Top="10" Content="Ime"/>
    <TextBox Canvas.Left="50" Canvas.Top="10"/> </Canvas>

Komponente interfejsa - DockPanel

- DockPanel – panel za raspoređivanje elemenata uz ivice ili u centar panela
- DockPanel.Dock – pozicija elementa (Left, Top, Right, Bottom)
- LastChildFill – da li poslednji element popunjava preostali prostor
  <DockPanel LastChildFill="true">
  <TextBox DockPanel.Dock="Top"/> <TextBox DockPanel.Dock="Bottom"/>
  <TextBox />
  </DockPanel>

Komponente interfejsa - StackPanel

- StackPanel – panel za raspoređivanje elemenata jedan ispod drugog ili jedan iza drugog
- Orientation – kako se elementi raspoređuju
  - Horizontal
  - Vertical
    <StackPanel Orientation="Horizontal"> <Label Content="Ime"/>
    <TextBox Name="tbIme" /></StackPanel>

Komponente interfejsa – WrapPanel

- WrapPanel – panel raspoređuje elemente isto kao StackPanel, ali prelazi u novi red/kolonu kada nema više mesta u tekućem redu/koloni
  <WrapPanel Orientation="Horizontal">
  <TextBox Name="textBox1"/><TextBox Name="textBox2" />
  <TextBox Name="textBox3" /><TextBox Name="textBox4" />
  <TextBox Name="textBox5" /></WrapPanel>

Komponente interfejsa - grupisanje

- Komponente koje sadrže hijerarhiju drugih komponenata
  - GroupBox
  - Expander
  - TabControl

Komponente interfejsa - GroupBox

- Header – naslov panela
- Unutar komponente se definiše Grid (ili drugi kontejner) za raspoređivanje elemenata …
  <GroupBox Header="Podaci o studentu" Height="119" Name="gbPodaci" Width="118">
  <Grid>...</Grid> </GroupBox>

Komponente interfejsa - Expander

- GroupBox koji korisnik može da sakriva
- IsExpanded – da li je prikazan
- ExpandDirection – na koju stranu se otvara pri prikazu (Down, Up, Left, Right)
  <Expander Header="Adresa" Height="100" Name="expAdresa" Width="309" IsExpanded="True" ExpandDirection="Down">
  <Grid><Label Content="Adresa" Height="28" Name="lbAdresa" /><TextBox Height="46" Name="tbAdresa" Width="241" /></Grid> </Expander>

Komponente interfejsa - TabControl

- TabControl
  - Roditeljska kontrola za tabove
  - TabStripPlacement - Pozicija trake sa tabovima (Left, Top, Right, Bottom)
- Svaki tab predstavljen sa TabItem
  - IsSelected – da li je Tab selektovan

Komponente interfejsa – DataGrid

- Tabelarni prikaz i unos podataka
- Definisanje kolona
  - Columns – kolekcija kolona tabele
  - Različiti tipovi kolona
    -- TextBoxColumn
    -- CheckBoxColumn
    -- ComboBoxColumn ...
  - Automatsko generisanje kolona
    -- AutoGenerateColumns
- Ručno podešavanje kolona

- Design-time - .XAML
  <DataGrid Name="dgStudenti“ Width=“200” Height=“400”>
  <DataGrid.Columns>
  <DataGridTextColumn Header="Ime" /><DataGridTextColumn Header="Prezime" />
  <DataGrid.Columns></DataGrid>
- Run-time - .cs
  dgStudenti.Columns.Add(new DataGridTextColumn()); dgStudenti.Columns[0].Header = “Ime”;
  dgStudenti.Columns.Add(new DataGridTextColumn()); dgStudenti.Columns[1].Header = “Prezime”;

- Items – podaci u tabeli
- ItemsSource – automatsko preuzimanje podataka iz kolekcije
  dgStudenti.AutoGenerateColumns = true;
  dgStudenti.ItemsSource = lista;
- Mogućnost dodavanja novih podataka i izmene postojećih
  - CanUserAddRows
  - CanUserDeleteRows
- Selekcija
  - SelectionMode – način selekcije, jedan (Single) ili više (Extended) redova
  - SelectionUnit – element selekcije, jedna ćelija (Cell) ili ceo red (FullRow)
  - SelectedItem, SelectedValue, SelectedIndex – preuzimanje podataka o selektovanom elementu
- Manipulacija prikazom
  - CanUserReorderColumns - izmena redosleda kolona
  - CanUserResizeColumns - izmena širine kolona
  - CanUserResizeRows – izmena visine redova
  - CanUserSortColumns - sortiranje podataka u koloni
- Grupisanje
  - Prikaz u tabeli može biti grupisan po određenom kriterijumu, ako je grupisanje definisano nad kolekcijom čiji se podaci prikazuju u tabeli

Komponente interfejsa - ListView

- Prikaz podataka u listi
- Slične funkcionalnosti kao DataGrid
- ListView nije namenjen za izmenu podataka
  <ListView Height=“300“ Width="100" Name="lvStudenti" >
  <ListViewItem Content="test1" /><ListViewItem Content="test2" /></ListView>

Komponente interfejsa - TreeView

- Prikaz u vidu stabla za hijerarhijski organizovane podatke
- TreeViewItem
  - jedna stavka u prikazu
  - može sadržati druge stavke
  <TreeView Name="tvFolderi"><TreeViewItem Header="Program files" IsExpanded="True">
  <TreeViewItem Header=“Adobe" /><TreeViewItem Header=“Office" />
  </TreeViewItem><TreeViewItem Header="Windows" /></TreeView>

Komponente interfejsa - Menu

- Reprezentuje padajući meni
- MenuItem
  - stavka menija

- ▫ može sadržati podstavke
- ▫ Header – naslov stavke
- Click – događaj, klik mišem na opciju menija
- Separator – razdvaja stavke menija
  <Menu Height="23" Width="200" Name="mainMenu"><MenuItem Header="File">
  <MenuItem Header="New" Click="New_Click" /><MenuItem Header="Open"
  Click="Open_Click"/></MenuItem><Separator/><MenuItem Header="Edit" Click="Edit_Click"/></Menu>

Komponente interfejsa - ContextMenu

- Padajući meni koji se otvara klikom na desni taster miša na određenoj komponenti
- MenuItem - stavka menija
- ContextMenu se definiše u komponenti za koju je vezan
  <GroupBox Header="Podaci o studentu" Name="gbPod"> <GroupBox.ContextMenu> <ContextMenu>
  <MenuItem Header="Detalji..." Click="Detalji_Click"/>
  <MenuItem Header="Postavi sliku..." Click="Slika_Click">
  </ContextMenu> </GroupBox.ContextMenu> </GroupBox>

Komponente interfejsa - ToolBar

- ToolBarTray – kontejner za smeštanje ToolBar komponenti
- ToolBar - grupa kontrola (najčešće srodne funkcionalnosti) na panelu sa alatima
  <ToolBarTray Height="26" Name="tbtAlati" Width="200" ><ToolBar Height="26" Name="tbFile" Width="55"
  ><Button Name="btn_New" ><Image Source="Images/new.gif"/></Button>
  <Button Name="btn_Open" ><Image Source="Images/open.gif"/></Button>
  </ToolBar></ToolBarTray>

Komponente interfejsa - MessageBox

- Jednostavan panel za prikaz obaveštenja i izbor opcije klikom na dugme
  string tekst = "Da li ste sigurni?";
  string naslov = "Potvrda";
  MessageBoxButton dugme = MessageBoxButton.YesNoCancel;
  MessageBoxImage ikona = MessageBoxImage.Warning;
  MessageBoxResult rez = MessageBox.Show(tekst, naslov, dugme, ikona); s
  witch (rez) { case MessageBoxResult.Yes: ... }

Komponente interfejsa - OpenFileDialog

- Dijalog za izbor fajla
  OpenFileDialog dlg = new OpenFileDialog();
  dlg.FileName = "Studenti"; // Default ime fajla
  dlg.DefaultExt = ".txt"; // Default ekstenzija
  //filter – tip fajlova koji se prikazuju
  dlg.Filter = "Text documents (.txt)|*.txt";
  bool? result = dlg.ShowDialog();//prikaz
  if (result == true) { string ime = dlg.FileName;
  //izabrani fajl //obrada ... }

Komponente interfejsa - SaveFileDialog

- Dijalog za snimanje fajla
  SaveFileDialog dlg = new SaveFileDialog();

dlg.FileName = “Studenti"; // Default ime fajla
dlg.DefaultExt = ".txt"; // Default ekstenzija
//filter – tip fajlova koji se prikazuju dlg.Filter = "Text documents (.txt)|*.txt";
bool? result = dlg.ShowDialog();//prikaz
if (result == true) { //ime fajla definisano u dijalogu string ime = dlg.FileName; //obrada ... }

Komponente interfejsa - PrintDialog

- Dijalog za prikaz opcija za štampanje
  PrintDialog dlg = new PrintDialog();
  dlg.PageRangeSelection = PageRangeSelection.AllPages;
  dlg.UserPageRangeEnabled = true;
  bool? result = dlg.ShowDialog(); //prikaz
  If (result == true) { // Stampanje dokumenta //... }

Komponente interfejsa - Image

- Komponenta za prikaz slike
- Source
  - fajl sa slikom koja se prikazuje
  - može se referencirati po
    -- apsolutnoj putanji
    -- relativnoj putanji
    -- iz resursa aplikacije
    <Image Height="150" Width="200" Name="imgLogo"Source="/Images/ftn.gif" />

Komponente interfejsa - MediaElement

- Prikaz audio ili video sadržaja
- Source – putanja do fajla sa sadržajem
- Position – pri reprodukciji sadržaja, vreme proteklo od početka audio/video sadržaja
- SpeedRatio – ubrzavanje/usporavanje pri reprodukciji
- Play() – pokretanje reprodukcije
- Stop(), Pause() – zaustavljanje reprodukcije
  <MediaElement Height=“400" Width=“300" Name=“mePrezentacija" Source=@“C:\Temp\prezentacija.avi”/>

Komponente interfejsa - Calendar

- Prikaz i izbor datuma
- SelectedDate – selektovani datum
- SelectionMode – način izbora datuma
  - SingleDate, SingleRange, MultipleRange, None
- DisplayMode – način prikaza kalendara
  - Month, Year, Decade
    <Calendar Height="170" Width="170" Name="clDatumRodj" DisplayMode="Month" />

Komponente interfejsa - DatePicker

- Izbor i unos datuma
- Slične funkcionalnosti kao Calendar
- Unos datuma se vrši ukucavanjem brojeva datuma
- Izbor datuma se vrši iz ugrađene Calendar kontrole

- Language – lokalna podešavanja formata datuma
  <DatePicker Height="25" Width="120" Name="dtpDatumRodjenja" Language="sr"/>

C# WPF – Povezivanje kontrola sa podacima

Veza kontrole sa podacima

- Iz objektnog modela potrebno je prekopirati podatke u kontrole u kojima se podaci prikazuju
- Nakon izmene u kontrolama, potrebno je osvežiti objektni model
- Data Binding - WPF mehanizam koji automatski vrši ove operacije
- Generalno, Data Binding obezbeđuje automatsko sinhronizovanje vrednosti određenog objekta sa njegovim izvorom podataka

Data Binding

- Klasa **System.Windows.Data.Binding**
- Veza izvora podataka sa ciljnim podatkom
- Za prikaz imena studenta (definisanog u objektu klase Student) u TextBox komponenti
  - Objekat klase Student je izvor veze
  - Ime studenta je svojstvo (Property)
  - TextBox komponenta je cilj veze
  - Text svojstvo u TextBox komponenti je zavisno svojstvo (Dependency Property)

Smer povezivanja

- **Binding.Mode**
- Jednosmerno (OneWay) – izmena izvornog podatka automatski menja ciljni podatak, ali obrnuto ne važi
- Dvosmerno (TwoWay) – izmene nad izvornim podatkom menjaju ciljni podatak i obrnuto
- Jednosmerno ka izvornom podatku (OneWayToSource) – izmena ciljnog podatka automatski menja izvorni podatak, ali ne i obrnuto
- Samo jedanput (OneTime) – pri inicijalizaciji, ciljni podatak dobija vrednost izvornog podatka, ali se kasnije promene izvornog podatka ne reflektuju na vrednost ciljnog podatka

Interfejs INotifyPropertyChanged

- Obezbeđuje notifikaciju u trenutku kada se desi izmena objekta
- Objekat koji je izvor povezivanja, mora da implementira ovaj interfejs

Trenutak povezivanja

- **Binding.UpdateSourceTrigger**
- Svojstvo definiše u kom trenutku se podaci iz jednog objekta kopiraju u drugi
- Najčešće se kopiraju pri izmeni svojstva (**UpdateSourceTrigger** ima vrednost **PropertyChanged**)
- Može se vezati za druge događaje (npr. TextBox je napravljena tako da podrazumevano prebacuje podatke nakon što izgubi fokus)

DataBinding – Implementacija

- Prikaz imena studenta u TextBox komponenti
- TextBox koji prikazuje ime studenta **– Path** definiše izvorno svojstvo koje se kopira u ciljni objekat
  - XAML fajl
    <TextBox Name="tbImeStudenta" Text="{Binding Path=Ime}"/>
- Definisanje izvora podataka za TextBox putem svojstva **DataContext**

- Cs fajl
  Student s1 = new Student();
  s1.Ime = "Goran";
  s1.Prezime = "Savic";
  tbImeStudenta.DataContext = s1;
- Ako je izvor podataka drugi element u aplikaciji, može se koristiti svojstvo ElementName
- Lista sadrži objekte klase Student, a TextBox prikazuje prezime selektovanog studenta
  <TextBox Name="tbPrezimeStudenta" Text="{Binding ElementName=lvStudenti,
  Path=SelectedValue.Prezime}"/>
- Path se ne mora specificirati ako je izvorni podatak ceo izvorni objekat
  lvStudenti.ItemsSource = listaStudenata;
- Moguće je kompletno povezivanje izvršiti programski u toku izvršavanja programa
- Prikaz imena studenta u TextBox komponenti
  Binding b = new Binding(); //source property
  b.Path = new PropertyPath("Ime"); //target and dependency property
  tbImeStudenta.SetBinding(TextBox.TextProperty, b); //binding source
  tbImeStudenta.DataContext = s1;

Povezivanje sa kolekcijom

- **ItemsControl** – kontrola koja se može koristiti za prikaz kolekcije elemenata
- Naslednice klase ItemsControl: **ListView, TreeView, DataGrid**, ...
- Svojstvo **ItemsSource** za definisanje izvora podataka
- Kolekcija mora da implementira **INotifyCollectionChanged** interfejs
  ObservableCollection studenti = new ObservableCollection();
  lvStudenti.ItemsSource = studenti;

Manipulacija prikazom kolekcije

- Standardni zahtevi su sortiranje, filtriranje i grupisanje podataka u kolekciji
- Ove operacije realizuju se kreiranjem pogleda na kolekciju
- Pogled omogućuje različite prikaze podataka u kolekciji bez izmene sadržaja kolekcije

Kreiranje pogleda

- Korišćenjem **ICollectionView**
  ICollectionView view = CollectionViewSource. GetDefaultView(listaStudenata);
  lvStudenti.ItemsSource = view;
- Korišćenjem **CollectionViewSource**
  CollectionViewSource view = new CollectionViewSource();
  view.Source = listaStudenata;
  lvStudenti.ItemsSource = view.View;

Sortiranje

- Pogled sadrži spisak objekata tipa **SortDescription**
- **SortDescription**
  - opis sortiranja po jednom atributu kolekcije
  - sadrži naziv svojstva i smer sortiranja
- Može se istovremeno sortirati po više podataka
  view.SortDescriptions.Add(new SortDescription("Prezime",ListSortDirection.Ascending));

Filtriranje

- Korišćenjem **ICollectionView**
  - Predikat koji definiše da li će objekat biti prikazan

```
public bool Uslov(object s) {
Student student = s as Student;
 return student.Ime.Length > 0 && student.Ime[0] == 'A';
}
```

  - Postavljanje filtera

```
view.Filter = new Predicate
```

- Korišćenjem **CollectionViewSource**
  - Definišemo događaj

```
cvs.Filter += new FilterEventHandler(PocetnoA);
```

  - Definišemo obrađivač događaja

```
private void PocetnoA(object sender, FilterEventArgs e) {
Student s = e.Item as Student;
 if (s != null) e.Accepted = s.Ime.Length > 0 && s.Ime[0] == 'A';
}
```

Grupisanje

- Organizacija podataka u logičke grupe po određenom kriterijumu
- Grupe studenata koji se isto prezivaju

```
PropertyGroupDescription groupDescription = new PropertyGroupDescription();
groupDescription.PropertyName = "Prezime";
view.GroupDescriptions.Add(groupDescription);
```

Trenutni element u pogledu

- **ICollectionView.CurrentItem**
- Povezivanje vrednosti u komponenti sa trenutnim elementom u pogledu

```
ICollectionView view = CollectionViewSource. GetDefaultView(listaStudenata);
 lvStudenti.ItemsSource = view;
lvStudenti.IsSynchronizedWithCurrentItem = true;                //!!!

Binding b1 = new Binding();
 b1.Path = new PropertyPath("Prezime");
tbStudentRunTime.SetBinding(TextBox.TextProperty, b1);
 tbStudentRunTime.DataContext = view
```

C# Događaji

Primer

- U toku izvršavanja određuje se vrednost promenljive koja se ispisuje

```
class Test {
public string S {get; set;}
public void Ispis() {
      Console.WriteLine(S);
}
}
Test t = new Test();
 t.S = "Petar";
t.Ispis();
```

Delegati - Primer

- U toku izvršavanja određuje se koja metoda će obaviti ispisivanje
- Definisane su dve metode za ispis, u trenutku kompajliranja se ne zna koja metoda treba da se poziva

```
private void IspisMalimSlovima(string s) {
        Console.WriteLine(s.ToLower());
}
private void IspisVelikimSlovima(string s) {
        Console.WriteLine(s.ToUpper());
}
```

- Delegat je kao referenca na metodu
- Svojstvo OI "pokazuje" na metodu koja će izvršiti ispis
  - zaglavlje metode mora biti u skladu sa delegatom ObradjivacIspisa
- U toku izvršavanja se postavlja vrednost promenljive OI

```
class Test {
public delegate void ObradjivacIspisa(string tekst);
public ObradjivacIspisa OI { get; set; }
public void Ispis(String s) {
        if (OI != null)
                OI(s);
        }
}
Test t = new Test();
t.OI = IspisVelikimSlovima;
t.Ispis("Tekst");
```

Događaji

- Mehanizam kojim objekat obaveštava korisnika objekta da je izvršena određena operacija nad objektom
- Implementiraju se korišćenjem delegata
  - Korišćenjem ključne reči **event** obezbeđuje se enkapsulacija delegata - korisnik objekta može da reaguje na događaj, ali ne i da sam izaziva događaj
- U trenutku kada se događaj desi, klasa obezbeđuje poziv metode, a korisnik klase određuje koja se konkretno metoda poziva i obrađuje događaj

Događaji - Primer

- Objekat obaveštava korisnika da je došlo do promene teksta
- Pri promeni teksta, pozvaće se metoda koju određuje korisnik objekta

```
class Test {
    private string s;
     public string S {
          get { return s; }
          set {
             s = value;
            if (PromenaTeksta != null)
                 PromenaTeksta(this);
          }
        }
public delegate void ObradjivacPromeneTeksta(Test test);
```

```
public event ObradjivacPromeneTeksta PromenaTeksta;
}
```

```
Test t = new Test();
t.PromenaTeksta += ObradaPromene;
t.S = "Petar";          //nakon ovog reda pozvace se metoda ObradaPromene
private void ObradaPromene(Test t) {
        Console.WriteLine( "Izvrsena promena. Novi tekst: " + t.S);
}
```

C# metode

Sintaksa

- Definisanje metode
  povratni_tip naziv_metode(
  tip_prvog_parametra naziv_prvog_parametra,
  tip_drugog_parametra naziv_drugog_parametra, …)
  {
  //telo metode
  }
  void promeni(int x)
  {
  x = 5;
  }
- Metoda može biti bez parametara
- Ako povratni tip nije void, metoda mora sadržati **return** izraz
- Poziv metode
  naziv_promenljive = naziv_objekta.naziv_metode(vrednost_prvog_parametra, vrednost_drugog_parametra, …)

Prenos po vrednosti i referenci

- Pri prenosu parametara u metodu, kopira se vrednost parametra
  - kod vrednosnih tipova, vrednost je sama instanca
    -- u metodi se koristi kopija instance
    -- posledica: metoda vrši izmene nad kopijom instance, original u pozivaocu ostaje nepromenjen
  - kod adresnih tipova, vrednost je referenca na instancu
    -- u metodi se koristi kopija reference, ali i ova kopija pokazuje na istu instancu kao originalna
    -- posledica: metoda vrši izmene nad instancom na koju referenca pokazuje. Sve promene izvršene nad instancom u metodi, vidljive su i u pozivaocu

Prenos vrednosnog tipa po adresi

- **ref** ispred parametra i u deklaraciji i pri pozivu metode
  void promeni(ref int x) {
  x = 5;
  }
  int a = 10;
  promeni(ref a);
- Definisanjem parametra kao izlaznog navođenjem out ispred parametra i u deklaraciji i pri pozivu metode
  void promeni(out int x) {

```
        x = 5;
    }
    int a;
promeni(out a);
```

- **ref** – parametar mora biti inicijalizovan pre prenosa
- **out** – može se poslati neinicijalizovan parametar. Metoda je dužna da postavi vrednost parametra

Metode sa promenljivim brojem parametara

- Metoda prima niz parametara označen ključnom rečju params
- Ako su svi parametri istog tipa

```
void ispis(params int[] brojevi) {
      foreach (int i in brojevi)
        Console.WriteLine(i);
}
ispis(2, 5, 7);
```

- Generička varijanta za parametre različitog tipa

```
void ispis(params object[] stavke) {
      foreach (object s in stavke)
        Console.WriteLine(s.ToString());
    }
ispis(new Računar(), new Automobil(), new Student());
```

C# izuzeci

Pojam i sintaksa

- Obrada grešaka nastalih u toku izvršavanja programa
- Sintaksno gledano, izuzetak je instanca klase **System.Exception** ili neke od naslednica ove klase
- Atributi klase System.Exception
  - **Message** – opis izuzetka
  - **StackTrace** – trag izvršavanja programskog koda koji je doveo do izuzetka

Obrada izuzetka

```
try {
    double rez = a / b;
} catch (DivideByZeroException e) {
     Console.WriteLine(e.StackTrace);
  }
```

- Može biti više **catch** blokova – svaki blok obrađuje različit tip izuzetka
- Na kraju može da stoji **finally** blok
  - izvršava se nakon obrade izuzetka
  - izvršava se ako se ne desi izuzetak
  - **Ne izvršava se** ako se izuzetak desio, a nije obrađen

Izazivanje izuzetka

- Ključna reč **throw** nad instancom klase **System.Exception** ili naslednicom ove klase
- Izazivanje izuzetka završava metodu – dalji kod se ne izvršava

- Izuzetak se šalje pozivaocu metode, koja obrađuje izuzetak ili šalje dalje svom pozivaocu i tako redom
  if (obj == null)
  throw new ArgumentNullException(“Neinicijalizovan objekat”);

C# Životni ciklus objekta

Kreiranje objekta

- Kreiranje objekta **Racunar r = new Racunar();**
  1. CLR alocira jedan deo heap memorije za objekat
  - Nije pod kontrolom programera
  2. Inicijalizuje se objekat
  - Pod kontrolom programera kroz kod u konstruktoru

Uništavanje objekta

- Garbage Collector
  - deo CLR koji se povremeno aktivira i uništava objekte koji se ne koriste
  - Ne može programer da inicira uništavanje objekta
- Uništavanje se sastoji od
  1. Izvršavanja destruktora
  - Nekad je potrebno izvršiti programski kod nakon uništavanja objekta da bi sistem nastavio ispravan rad
  - Programer piše destruktor - ne možemo biti sigurni da će se kod izvršiti jer to zavisi od toga da li će biti potrebe da se aktivira Garbage collector
  2. Dealociranja memorije
  - CLR oslobađa memoriju koja je bila zauzeta od strane objekta

Destruktor

- Nasleđe C++ jezika gde se oslobađanjem memorije bavio programer
- Najčešće je nepotreban obzirom da CLR automatski uništava objekat
  - Koristan je nekad zbog optimizacije da bi se efikasnije uništio objekat
  - Ili da bi se zatvorila veza ka bazi ili fajlu (mada ima i boljih načina da se ovo uradi)

Destruktor sintaksa

```
class FileProcessor
{
    FileStream file = null;
    public FileProcessor(string fileName)
    {
     this.file = File.OpenRead(fileName);
     }
     ~FileProcessor()
       {
          this.file.Close();
       }
  }
```

Ograničenja za destruktor

- Može se definisati samo za adresne tipove
- Ne može se direktno pozvati iz koda i zato ne može
  - da ima modifikator pristupa (npr. **public**)

- pa prima parametre (npr. **~FileProcessor(int x)**)

Implementacija destruktora

- Kompajler automatski prevodi destruktor u metodu Finalize nasleđenu od klase Object

```
protected override void Finalize() {
    try {    //programski kod iz destruktora }
    finally { base.Finalize(); }
```

- Ne možemo napisati svoju redefiniciju metode Finalize
- Ne možemo direktno pozvati metodu Finalize

Garbage Collector - aktiviranje

- GC se aktivira samo kada ima potrebe
- Kada količina heap memorija postane premala
- Dakle, ne aktivira se odmah kad se objekat više ne upotrebljava
- GC uništava samo objekte na koje ne postoji nijedna aktivna referenca
- Eksplicitni poziv Garbage collectora
- System.GC.Collect()
- Ne preporučuje se ovakav eksplicitni poziv

Garbage Collector - implementacija

- Izvršava se kao posebna nit
- odvojen tok izvršavanja koji se izvršava konkurentno sa glavnim tokom izvršavanja
- Aktivira se periodično
- Kada se aktivira, ostale niti u aplikaciji su blokirane jer GC premešta objekte u memoriji
- Održava mapu dostupnih objekata
- Svi koji nisu u ovoj mapi smatraju se nedostupnim i uništavaju se
- Pre toga se pozivaju njihovi destruktori ako postoje

Disposal metoda

- Gde objekat treba da obavi završne operacije?
- Npr. zatvaranje veze ka fajlu : **reader.Close();**
- U destruktoru je nepouzdano
  - Ne znamo da li će se i kada pozvati
- Na kraju metode nije exception-safe
  - Ako se u toku metode desi izuzetak kod se neće izvršiti
- U finally bloku je pouzdano, ali komplikuje kod
- Standardan i najbolji način je korišćenjem using bloka

```
using (TextReader reader = new StreamReader(filename)) {
    string line;
    while ((line = reader.ReadLine()) != null) {
        Console.WriteLine(line);
    } }
```

- Objekat će biti uništen kada se završi using blok

IDisposable interfejs

- U using izrazu može da se koristi samo objekat koji implementira IDisposable interfejs
- Interfejs ima samo jednu metodu : **void Dispose();**
- Na kraju using bloka automatski će biti pozvana Dispose metoda

- U ovoj metodi je potrebno napisati završne operacije koje objekat treba da obavi pre uništavanja
  - Npr. u TextReader klasi je u ovoj metodi zatvorena veza ka fajlu

ADO .NET

Uvod

- ADO.NET predstavlja skup klasa koje obezbeđuju servise za pristup podacima
- Klase su univerzalno dizajnirane, tako da omoguće rad sa različitim izvorima podataka
- Dve grupe servisa
  - Preuzimanje podataka iz izvora podataka
  - Pristup podacima

Preuzimanje podataka

- Data Provider
  - Povezivanje na izvor podataka
  - Izvršavanje upita
  - Preuzimanje podataka
- Specijalizovani Data Provider za svaki izvor podataka
- Ugrađeni Data Provider
  - Za Microsoft SQL Server (Prostor imena **System.Data.Sql**)
  - Za OLE DB pristup
  - Za ODBC pristup

Data Provider komponente

- Connection
- Command
- DataReader
- DataAdapter

Uspostavljanje veze – klasa Connection

- Connection
  - Uspostavljanje i upravljanje vezom sa izvorom podataka
  - Podržava upravljanje transakcijama
  - DbConnection – klasa predak za sve tipove konekcija
  - Različiti tipovi konekcija za različite izvore podataka
    - SqlConnection – za Microsoft SQL Server
    - OleDbConnection – za pristup izvoru podataka preko Microsoft OLE DB API
    - OdbcConnection – za pristup izvoru podataka preko ODBC API
    - OracleConnection – za pristup Oracle DBMS

Uspostavljanje veze - sintaksa

```
using (SqlConnection conn = new SqlConnection(connectionString)) {
    conn.Open(); // rad sa bazom podataka
  }
```

- **Using** izraz obezbeđuje automatsko zatvaranje konekcije pri uništavanju objekta (poziva se **Dispose** metoda, objekat mora da implementira **IDisposable** interfejs)

Uspostavljanje veze – Connection String

- Connection String – tekst koji specificira izvor podataka i parametre povezivanja
- Sintaksa (nazivi parametara) je različita za svaki tip izvora podataka
- Generalna sintaksa
  - parametar1=vrednost1;parametar2=vrednost2;
- Primer
  - conn.ConnectionString = @" Data Source = .\SQLExpress;
    Integrated Security = true;
    Initial Catalog = ComputerShop";

Uspostavljanje veze – Connection pooling

- Uspostavljanje veze troši vreme i resurse
- U praksi se u aplikaciji često vrši otvaranje/zatvaranje iste veze
- Connection pooling – mehanizam za optimizaciju korišćenja resursa pri povezivanju sa jednim izvorom podataka
- Automatski se inicijalizuje određen broj veza ka jednom izvoru podataka
- Kada zatraži otvaranje veze, aplikacija dobija na korišćenje jednu od veza iz pool-a, a pri zatvaranju konekcije vraća vezu u pool
- U ADO.NET Connection pooling mehanizam je automatski uključen

Connection pooling - Primer

```
using (SqlConnection conn = new SqlConnection( "Data Source = .\SQLExpress;
Integrated Security = true;")) {
        conn.Open(); // kreira se pool
}
 using (SqlConnection conn = new SqlConnection( " Data Source = .\SQLExpress“
Integrated Security = true;)) {
       conn.Open(); // posto je isti Connection String,  uzima se jedna konekcija iz ranije  kreiranog pool-a
}
```

Command

- Command
  - Izvršavanje upita nad izvorom podataka
  - Moguće je poslati i preuzeti parametre
  - Može se izvršavati u okviru transakcije
  - DbCommand – klasa predak za sve tipove komandi. Postoje različiti tipovi komandi za različite izvora podataka

Kreiranje komande

- Prva varijanta
  - Instanciranje komande i postavljanje konekcije
    SqlCommand comm = new SqlCommand();
    comm.Connection = conn;
- Druga varijanta
  - Odgovarajući tip komande se dobija iz konekcije
    SqlCommand comm = conn.CreateCommand();
- Postavljanje upita koji komanda izvršava
  comm.CommandText = "SELECT * FROM STUDENT";

Izvršavanje komande

- Pozivom različitih metoda komande, zavisno od tipa podatka koji je rezultat izvršavanja:
  - ExecuteReader - Vraća **DataReader** objekat; Koristi se kada je rezultat upita kolekcija podataka **(SELECT * FROM STUDENT)**
  - ExecuteScalar - Vraća jedan podatak **(SELECT IME FROM STUDENT WHERE ID=1)**
  - ExecuteNonQuery – Za upite koji ne vraćaju podatke **(UPDATE STUDENT SET IME='PETAR' WHERE ID=1)**
  - ExecuteXMLReader – Vraća **XMLReader** objekat; Podržano samo za SqlCommand klasu

Tipovi komandi

- Tip se definiše svojstvom **CommandType** u komandi
- Podržani tipovi
  - Text
    - SQL upit koji treba biti izvršen nad izvorom podataka
  - StoredProcedure
    - Izvršavanje uskladištene procedure na serveru. CommandText je u tom slučaju naziv procedure.
  - TableDirect
    - Preuzimanje svih podataka iz jedne tabele. CommandText je u tom slučaju naziv tabele.

Parametrizovane komande

- Komanda može da sadrži parametre upita
- Primer
  - **(SELECT IME FROM STUDENT WHERE ID=@ID_PRM)**
  - **@ID_PRM** je parametar koji se postavlja za komandu
  - Postavljanje parametra

```
String id_studenta = "1";
SqlParameter paramId = new SqlParameter("@ID_PRM", id_studenta);
comm.Parameters.Add(paramId);
```

- Tipovi parametara
  - Input (ulazni - default), Output (izlazni), InputOutput (ponaša se i kao ulazni i kao izlazni), ReturnValue (sadrži povratnu vrednost komande – npr. kod poziva uskladištene procedure)
- Parametrizovane komande predstavljaju zaštitu od SQLInjection napada
- Primer SQL-Injection napada:
  - Pri formiranju upita se ne koriste parametri u komandi:
    - **Comm.CommandText = "SELECT * FROM STUDENT WHERE brojIndeksa = " + tbBrIndeksa.Text;**
  - U polje "Broj indeksa" u aplikaciji korisnik unese tekst: 123; DROP TABLE STUDENT
  - Pošto promenljiva **tbBrIndeksa.Text** sadrži uneseni SQL upit, taj SQL upit će biti izvršen i tabela sa spiskom studenata obrisana

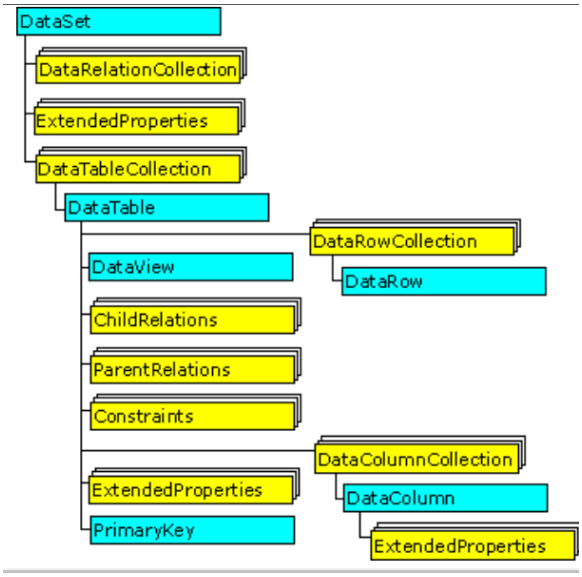
DataReader

- Objektna struktura koja sadrži preuzete redove iz izvora podataka
- Više tipova zavisno od izvora podataka
  - SqlDataReader, OleDbDataReader, OdbcDataReader, ...
- Popunjavanje objekta
  - **SqlDataReader reader=comm.ExecuteReader();**
- Preuzimanje podataka
  - **reader.Read()** – čitač se pozicionira na naredni red

- **reader.GetString(0)** – preuzimanje string vrednosti iz prve kolone reda
- Nakon korišćenja, potrebno je zatvoriti DataReader
- **reader.Close();**

Pristup podacima

- DataSet - memorijska reprezentacija podataka preuzetih iz izvora podataka



DataSet

- Struktura DataSet objekta odgovara relacionoj bazi podataka
- DataSet sadrži tabele
  - DataTable klasa
  - Svojstvo Tables sadrži kolekciju DataTable objekata
  - Indekser za preuzimanje jedne tabele
    - ds.Tables[0]
    - ds.Tables[“Student”]
  - Tabela sadrži redove
  - DataRow klasa predstavlja jedan red
  - Svojstvo Rows klase DataTable sadrži kolekciju redova
  - Indekser za preuzimanje jednog reda
    - ds.Tables[0].Rows[3]
  - Tabela sadrži kolone
  - DataColumn klasa predstavlja jednu kolonu
  - Svojstvo Columns klase DataTable sadrži kolekciju kolona
  - Indekser za preuzimanje jedne kolone
    - ds.Tables[0].Columns[3]
    - ds.Tables[0].Columns[“Br_Indeksa”]
- Red sadrži vrednosti (ćelije u redu)
  - Svojstvo ItemArray sadrži kolekciju vrednosti
  - Indekser za preuzimanje vrednosti iz ćelije
    - Object o = row[2];
    - Object o = row[“Br_Indeksa”];

DataSet relacije

- Između tabela mogu se definisati relacije
  - Relacije omogućuju poštovanje referencijalnog integriteta među podacima u DataSet objektu
  - DataRelation klasa predstavlja relaciju između dve tabele
    - ParentTable – prva tabela u vezi
    - ChildTable – druga tabela u vezi
    - ParentColumns – kolone prve tabele koje formiraju relaciju
    - ChildColumns – kolone prve tabele koje formiraju relaciju
  - Svojstvo Relations klase DataSet sadrži kolekciju relacija
  - Indeksiranje veze
    - ds.Relations[0]
    - ds.Relations[“Veza_Student_Predmet”]

DataSet – pristup podacima preko relacije

- Kada je formirana relacija, mogu se za određeni red jedne tabele preuzeti podaci iz druge tabele koji su povezani sa ovim redom
- DataRow klasa sadrži metodu GetChildRows koja vraća kolekciju redova iz druge tabele
- Primer
  - Preuzimanje predmeta određenog studenta

```
DataRow student = ds.Tables[0].Rows[0];
DataRow[] studentoviPredmeti = student.GetChildRows(“Veza_Student_Predmet”);
```

DataAdapter

- Kopiranje podataka iz izvora podataka u DataSet i obrnuto
- Popunjavanje DataSet objekta

Metoda **Fill** izvršava komandu predstavljenu svojstvom **SelectCommand**

```
DataSet ds = new DataSet();
SqlDataAdapter da = new SqlDataAdapter();
comm.CommandText = @"SELECT * FROM STUDENT";
da.SelectCommand = comm; da.Fill(ds,"Student");
```

DataAdapter – primary-key constraint

- Pri popunjavanju DataSet objekta automatski se kreira DataTable objekat, ali se ne kreiraju ograničenja (constraints) definisana nad tabelom u izvoru podataka
- Za kreiranje primary-key constraint (u kojoj koloni se skladište primarni ključevi), dve varijante:
  - Metoda **FillSchema**

```
da.FillSchema(ds, SchemaType.Source, "Student");
```

  - Svojstvo MissingSchemaActions

```
da.MissingSchemaAction = MissingSchemaAction.AddWithKey;
```

DataAdapter – popunjavanje iz više tabela

- DataSet je moguće popuniti putem različitih DataAdapter objekata od kojih svaki preuzima podatke iz jedne tabele

```
daStudenti.Fill(ds, "Student");
daPredmeti.Fill(ds, "Predmet“);
```

- Između dve tabele u DataSet-u je tada moguće definisati relaciju koja predstavlja spoljni ključ u izvoru podataka

```
ds.Relations.Add(“VezaStudentPredmet",
ds.Tables[“Student"].Columns[“StudentID"],
ds.Tables[“Predmet"].Columns[“PredmetID"]);
```

DataAdapter – ažuriranje izvora podataka

- Slanje izmenjenih podataka iz memorijske reprezentacije (DataSet objekta) u izvor podataka
- Metoda Update klase DataAdapter
  - Vrši slanje izmenjenih podataka
  - Izmenjeni podaci mogu biti poslani kao
    - Ceo DataSet
    - Jedna tabela (DataTable objekat)
    - Niz izmenjenih redova (DataRow objekata)
- Najpre se analiziraju izmene izvršene nad DataSet objektom
- Svaki red ima podatak o tome da li je izmenjen
  - U klasi DataRow svojstvo RowState (enumeracija DataRowState)
  - Mogući statusi:
    - Added – novi red dodan
    - Deleted – red obrisan
    - Detached – red kreiran, ali ne pripada nijednoj tabeli
    - Modified – red izmenjen
    - Unchanged – red neizmenjen
- Zavisno od statusa reda, izvršava se jedna od komandi
- U DataAdapter objektu se definišu komande za svaku akciju:
  - Svojstvo InsertCommand – ova komanda se izvršava za nove redove
  - Svojstvo DeleteCommand – ova komanda se izvršava za obrisane redove
  - Svojstvo UpdateCommand – ova komanda se izvršava za izmenjene redove
- Primer

```
SqlCommand delCom = conn.CreateCommand();
delCom.CommandText = @"DELETE FROM STUDENT WHERE ID=@ID";
SqlParameter idParam = delCom.Parameters.Add("@ID",SqlDbType.Int); idParam.SourceColumn = "ID";
da.DeleteCommand = delCom; ds.Tables[0].Rows[0].Delete();
da.Update(ds,“Student");
```

Upravljanje transakcijama

- Klasa SqlTransaction
  - BeginTransaction – pokretanje transakcije
  - Commit – potvrda transakcije
  - RollBack – poništavanje transakcije
- Kreiranje transakcije

```
SqlTransaction t = connection.BeginTransaction();
```

- Izvršavanje komande u transakciji

```
command.Transaction = t;
```

WPF Resursi

WPF Resursi

- Deljeni objekti koji se mogu koristiti u različitim delovima aplikacije
- Primer
  - Konverter objekat koji se može koristiti bilo gde unutar **Window** elementa

```
<Window.Resources>
<local:ValidatorConverter x:Key="converter" />
</Window.Resources>    …
```

<TextBlock Text="{Binding ElementName=tbEmail,
Path=(Validation.Errors), Converter={StaticResource ResourceKey=converter}}" />

Statički resursi

- Vrednost statičkih resursa je poznata u početku izvršavanja aplikacije
- Ne menjaju se pri izvršavanju aplikacije
- Referenciranje resursa u XAML fajlu:
  - Prvi način
    <object property="{StaticResource key}" .../>
  - Drugi način
    <object> <object.property> <StaticResource ResourceKey="key" .../> </object.property> </object>

Dinamički resursi

- Vrednost dinamičkih resursa se postavlja tek u toku izvršavanja aplikacije
- Primer – u toku izvršavanja programa postavlja se boja dugmeta
  <Window.Resources><SolidColorBrush Color="LightBlue" x:Key="background" /></Window.Resources>
  <StackPanel Name="panel"><Button Name="btnTest" Content="Test" Background="{DynamicResource background}"/>
  - Cs fajl:
    void R_Loaded(object sender, RoutedEventArgs e){ panel.Resources["background"] = Brushes.Yellow;

WPF Validacija

Validacija

- Provera ispravnosti podataka pri unosu
- Ukoliko unesena vrednost nije u skladu sa određenim kriterijumom, korisnik u grafičkom interfejsu dobija obaveštenje da je vrednost neispravna

- WPF omogućuje različite tehnike validacije:
  - Validacija u slučaju izuzetka
  - Validacija putem validacionih pravila
  - Validacija korišćenjem **IDataErrorInfo** interfejsa
- Sve tehnike validacije se koriste u sklopu Data Binding mehanizma
- Trenutak kada se validacija vrši: Binding.UpdateSourceTrigger

Validacija u slučaju izuzetka

- Najjednostavniji oblik validacije
- Ako se pri prebacivanju podatka iz izvornog objekta u ciljni pojavi izuzetak, korisnik dobija obaveštenje o neispravnoj vrednosti
- Primer
  - unos godine upisa studenta (u klasi **Student** godina upisa je **int**)
  - ako se unese tekst koji nije moguće konvertovati u broj, desiće se izuzetak, što ukazuje na nevalidnu vrednost
    <TextBox Name="tbGodinaUpisa" Text="{Binding Path=GodinaUpisa, ValidatesOnExceptions=True}"/>

Validacija putem validacionih pravila

- Koristi se u slučaju kada neispravan podatak ne izaziva izuzetak, već je potrebno u skladu sa određenim kriterijumom utvrditi da li je podatak ispravan

- Primer
  - Unos E-mail adrese studenta
  - E-mail je u klasi **Student** tipa **string**, pa bilo koja unesena vrednost ne izaziva izuzetak
  - Vrednost je validna ukoliko je u skladu sa formatom e-mail adrese
- Kreira se klasa naslednica klase **ValidationRule**
- U ovoj klasi se definiše programski kod koji utvrđuje ispravnost podatka
- Rezultat validacije je tipa **ValidationResult**
- **ValidationResult** sadrži svojstva
  - ErrorContent – sadrži detalje o grešci
  - IsValid – podatak da li je validirani podatak ispravan

```
public class EMailValidationRule: ValidationRule {
 public override ValidationResult Validate(
object value, System.Globalization.CultureInfo cultureInfo) {
 //utvrdjivanje da li je objekat value validan } }
```

- Pri definisanju ulazne komponente definiše se klasa koja vrši validaciju

```
<TextBox Name="tbEmail">
    <TextBox.Text>
        <Binding Path="Email">
            <Binding.ValidationRules>
                <local:EMailValidationRule/>
            </Binding.ValidationRules>
        </Binding>
    </TextBox.Text>
</TextBox>
```

Validacija putem IDataErrorInfo interfejsa

- Koristi se kada validnost podatka zavisi od vrednosti drugih podataka
- Primer
  - ako je za studenta unesen podatak o datumu odbrane diplomskog rada, tada je obavezno uneti i naslov teme rada
  - Ovo nije moguće realizovati putem validacionog pravila, jer pravilo samostalno validira samo jedan objekat
- Klasa implementira ovaj interfejs i definiše programski kod za validaciju svojih atributa

```
class Student: IDataErrorInfo
```

- Svojstvo **Error** definiše grešku na nivou celog objekta (npr. za prikaz u tabeli kao indikaciju da je ceo red neispravan)

```
public string Error { get { return "Neispravni podaci o studentu"; } }
```

- Indekser za validaciju konkretnog atributa klase

```
public string this[string propertyName] {
    get {
        switch (propertyName) {
            case “Email”:
               bool valid = EMailValidacija();
               if (!valid)
                    return “Neispravan e-mail”; ...
```

- Ulazna kontrola automatski poziva validaciju definisanu u klasi koja implementira **IDataErrorInfo interfejs**

```
<TextBox Name="tbEmail">
     <Binding Path="Email" ValidatesOnDataErrors="True"></Binding>
</TextBox>
```

Prikaz greške pri validaciji

- Default – crveni okvir oko kontrole
- Prilagođeni prikaz
  - U posebnoj labeli poruka o grešci – direktan prikaz poruke o grešci
    <TextBlock Text="{Binding ElementName=tbEmail,Path=(Validation.Errors).CurrentItem.ErrorContent}" />
  - U posebnoj labeli poruka o grešci – konverzija greške preko konvertera
  - Konverter
    class ValidatorConverter:IValueConverter {
    public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture) {
    //konverzija objekta value //(tipa ReadOnlyCollection) //u tekst za prikaz }
  - U posebnoj labeli poruka o grešci – konverzija greške preko konvertera
  - Definisanje i poziv konvertera
    <Window.Resources><local:ValidatorConverter x:Key="conv" /></Window.Resources>
    <TextBlock Text="{Binding ElementName=tbEmail, Path=(Validation.Errors), Converter={StaticResource ResourceKey=conv}}“ />
  - Definisanjem šablona za prikaz greške pri validaciji

```
<ControlTemplate x:Key="TextBoxErrorTemplate">
    <DockPanel LastChildFill="True">
      <TextBlock DockPanel.Dock="Right" Foreground="Orange" FontSize="12pt">
        !!!
      </TextBlock>
     <Border BorderBrush="Green" BorderThickness="1">
     <AdornedElementPlaceholder /></Border>
    </DockPanel>
</ControlTemplate>

<TextBox Validation.ErrorTemplate="{StaticResource ResourceKey= TextBoxErrorTemplate}" Name="tbEmail" >
<TextBox.Text>
      <Binding Path="Email">
        <Binding.ValidationRules>
          <local:EMailValidationRule/>
        </Binding.ValidationRules>
      </Binding>
  </TextBox.Text>
</TextBox>
```

WPF Stilovi

WPF Stilovi

- Tehnologija za dizajniranje izgleda komponenti grafičkog interfejsa u WPF aplikacijama
- Obezbeđuju razdvajanje poslovne logike od izgleda aplikacije
- Stil je nezavisno definisan skup grafičkih osobina koji se može primeniti na različite grafičke komponente
- Mogu se definisati u okviru bilo kojeg elementa izvedenog od **FrameworkElement ili FrameworkContentElement**
- Najčešće se definišu kao resurs u okviru **Resources** elementa

Primena stila na komponentu

- **Tag Style**

- **TargetType** – tip komponente na koju se stil primenjuje
- **Setter** – postavljanje vrednosti za određeno svojstvo komponente
- **Key**
  - naziv stila
  - ako nije definisan **Key**, stil se primenjuje na sve komponente
  - Kada **Key** nije definisan, stil implicitno dobija naziv u obliku x:Type vrednost_TargetType_atributa
- Primer – primena stila na sve TextBlock komponente

```
<Window.Resources>
        <Style TargetType="TextBlock">
            <Setter Property="FontFamily" Value="Comic Sans MS"/>
             <Setter Property="FontSize" Value="14"/>
        </Style>
</Window.Resources>
<TextBlock Name="lbIme" Text="Ime"/>
```

- Primer – primena stila na TextBlock komponente za koje je eksplicitno naznačen takav stil

```
<Window.Resources>
      <Style x:Key="Comic" TargetType="TextBlock">
          <Setter Property="FontFamily" Value="Comic Sans MS"/>
          <Setter Property="FontSize" Value="14"/>
       </Style>
</Window.Resources>
<TextBlock Name="lbIme" Text="Ime" Style="{StaticResource ResourceKey=Comic}"/>
```

Proširenje stila

- Stil baziran na postojećem stilu
- **BasedOn** – naziv stila na kojem je novi stil baziran

```
<Style TargetType="TextBlock" BasedOn="{StaticResource {x:Type TextBlock}}" x:Key="Title">
      <Setter Property="FontSize" Value="18"/>
       <Setter Property="FontWeight" Value="Bold"/>
</Style>
<TextBlock Name="lbTitle" Text="Podaci o studentu" Style="{StaticResource Title}" />
```

Šablon podataka

- DataTemplate – šablon koji definiše kako se prikazuju podaci
- Primer definisanja izgleda podataka u ListBox komponenti

```
<DataTemplate x:Key="StudentTemplate">
      <Border BorderBrush="Blue" BorderThickness="1">
          <TextBlock Text="{Binding Path=Prezime}"/>
     </Border>
</DataTemplate>
<ListBox Name="lbStudenti“ ItemTemplate="{StaticResource ResourceKey=StudentTemplate}"/>
```

Šablon kontrole

- ControlTemplate – definiše strukturu prikaza grafičkih elemenata u kontroli
- Primer – prikaz dugmeta kao elipse

```
<ControlTemplate x:Key="ElBtn" TargetType="Button">
        <Grid><Ellipse Fill="Yellow" Stroke="Black"/>
              <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/></Grid>
```

```
</ControlTemplate>
<Button Content="Test" Template="{StaticResource ResourceKey=ElBtn}" />
```

Okidači (Triggers)

- Dinamička promena izgleda grafičke komponente u toku izvršavanja programa
- Okidači se mogu se definisati u okviru elemenata **Style, DataTemplate i ControlTemplate**
- Primer – promena boje tekst polja kada je polje selektovano

```
<Style TargetType="TextBox">
    <Style.Triggers>
      <Trigger Property="IsFocused" Value="True">
        <Setter Property="Background" Value="Yellow"/>
</Trigger> </Style.Triggers> </Style>
```

Language-Integrated Query LINQ

Uvod

- LINQ je jezik za upite nad podacima
- Integrisan je u .NET kao deo sintakse jezika
- LINQ univerzalnom sintaksom omogućuje upite nad različitim izvorima podataka:
  - kolekcijom u memoriji
  - relacionom bazom
  - XML dokumentom
  - tokom podataka
  - ...

LINQ upit

- LINQ upit
  1. Definisanje izvora podataka
  2. Kreiranje upita
  3. Izvršavanje upita
- Primer

```
//definisanje izvor podataka
int[] nums = new int[] {2, 0, 1, 6};
//kreiranje upita
var res = from a in nums where a < 3 orderby a select a;
//izvršavanje upita
foreach (int i in res) Console.WriteLine(i);
```

- Izvor podataka – "queryable type" - bilo koji tip koji implementira interfejs **IEnumerable<T> i**li od njega izvedeni interfejs **IQueryable<T>**
- Definisanje upita – koje podatke želimo iz izvora podataka i kako da ti podaci budu organizovani
- Upit sadrži delove:
  - **from** – izvor podataka
  - **where** – filtriranje podataka
  - **group** – grupisanje podataka
  - **join** – veze između podataka koje nisu eksplicitno modelovane u izvoru podataka
  - **select** – tip elemenata koji su rezultat upita
- Upit se izvršava odloženo
  - pri korišćenju promenljive koja predstavlja rezultat upita

- svako korišćenje promenljive inicira izvršavanje novog upita – uvek se dobijaju aktuelni podaci iz izvora podataka

Povratna vrednost upita

- Povratna vrednost je **IEnumerable <T> ili IQueryable <T>**
- **Primer**
  int[] nums = new int[] {2, 0, 1, 6};
  IEnumerable res = from a in nums where a < 3 orderby a select a;
- promenljiva **res** će nakon izvršavanja upita sadržati sekvencu vrednosti tipa **int**
- Kompajler može automatski da odredi tip korišćenjem ključne reči **var**
  var res = from a in nums where a < 3 orderby a select a;

Struktura LINQ upita

- from
  - sadrži izvor podataka i "range" promenljivu
  - "range" promenljiva – redom ukazuje na svaki element u izvoru podataka. Koristi se za referenciranje elemenata u izvoru podataka u drugim delovima upita
- where
  - sadrži logički izraz
  - rezultat upita sadrži samo one elemente iz izvora podataka za koje je definisani logički izraz istinit
- select
  - specificira objekte koji su rezultat upita (npr. element kolekcije definisane u **from** delu, jedan atribut elementa, podskup atributa elementa, ili neka nova promenljiva nastala kao rezultat izračunavanja u upitu)
  - Primer (preuzimanje jednog atributa elementa)
    var rez = from s in studenti select s.Ime;
    foreach (string ime in rez) Console.WriteLine(ime);
- orderby
  - sadrži elemente po kojima se kolekcija sortira i smer sortiranja – **ascending ili descending**
  - Primer
    var res = from a in nums where a < 3 orderby a descending select a;
- group
  - specificira koji element se grupiše po kojem kriterijumu
  - ako se u ostatku upita vrše operacije nad grupom, grupa se može imenovati
  - Primer //studenti je ranije kreiran objekat tipa // List<Student>
    var rez = from s in studenti group s by s.Prezime into g select g;
    foreach (IGrouping<string, Student> g in rez) { Console.WriteLine(g.Key);
    foreach(Student s in g) Console.WriteLine("\t" + s.Ime); }
- join
  - navodi se kolekcija sa kojom se vrši spajanje i "range" promenljiva za kolekciju
  - Korišćenjem ključne reči on definiše se uslov spajanja kolekcija
  - Primer //studenti i gradovi su ranije kreirani // objekti tipa List<Student> i List<Grad>
    var rez = from s in studenti join g in gradovi on s.Grad equals g.Naziv select new {Ime = s.Ime,Prezime=s.Prezime, PttGrada = g.PttBroj};
    foreach (var r in rez) Console.WriteLine(r.Ime + " " + r.Prezime + " " + r.PttGrada);

LINQ za SQL

Uvod

- LINQ to SQL – .NET komponenta koja koristi mehanizme LINQ jezika za rad sa podacima smeštenim u relacionoj bazi podataka
- Dostupna od .NET verzije 3.5
- Obezbeđuje mapiranje objektnog na relacioni model (O-R mapiranje)
- Podržava samo Microsoft SQL Server
- Kao alternativa za LINQ to SQL razvijen je ADO .NET Entity Framework koji pruža širi skup funkcionalnosti i mogućnost rada sa različitim SUBP

Mapiranje objektnog modela na relacioni model

- Mapiranje se vrši anotiranjem elemenata objektnog modela (klasa i atributa)
- Mapiranje klase na tabelu u bazi podataka
  - Anotacija **Table**

  [Table(Name = "STUDENT")] public class Student { ...
- Mapiranje atributa klase na kolonu u bazi podataka
  - Anotacija **Column**

  public class Student { ...
  [Column(IsPrimaryKey = true, Name = "ID", CanBeNull = false)]
  public int Id { get; set; } [Column (Name="GRAD_ID")]
  public int GradId { get; set; }
- Mapiranje veze N:1
  - Definiše se atribut tipa **EntityRef** koji predstavlja referencu na drugu klasu (veza asocijacije)
  - Anotacija **Association** definiše atribute preko kojih se veza ostvaruje

  public class Student { ...
  private EntityRef <grad>; [Association(Storage = "grad", ThisKey =“GradId", OtherKey = "Id", IsForeignKey=true)]
  public Grad Grad { get { return this.grad.Entity; } set { this.grad.Entity = value; } }
- Mapiranje veze 1:N
  - Definiše se atribut tipa **EntitySet** koji predstavlja listu od N objekata (veza asocijacije)
  - Anotacija **Association** definiše atribute preko kojih se veza ostvaruje

  public class Grad { ... private EntitySet <Student> studenti = new EntitySet<Student>();
  [Association(Storage=“studenti", OtherKey=“GradId", ThisKey="Id")]
  public EntitySet<Student> Studenti { get { return this.studenti; } set { this.studenti.Assign(value); } }

Veza sa bazom podataka

- Klasa **DataContext**
  - Povezivanje na bazu podataka
  - Preuzimanje podataka iz baze
  - Slanje izmenjenih podataka u bazu

  DataContext dc = new DataContext(@"Integrated Security=true; InitialCatalog=Fakultet; Data Source=.\SQLExpress");

Preuzimanje podataka

- Korišćenjem **DataContext** klase
  - Klasa **Table** - programska reprezentacija tabele iz baze podataka Table studenti=dc.GetTable();
- LINQ upitom

  var res = from s in dc.GetTable() select s;
- Direktnim SQL upitom

  var res = dc.ExecuteQuery("SELECT * FROM STUDENT");
  foreach (Student s in res) Console.WriteLine(s.Ime);

Dodavanje sloga u tabelu u bazi podataka

- Kreira se novi objekat klase koja u objektnom modelu reprezentuje tabelu u bazi podataka
- Objekat se ubacuje u odgovarajuću tabelu DataContext objekta
- Izmena se šalje u bazu podataka

```
Student s = new Student("Petar", "Petrovic");
studenti.InsertOnSubmit(s); dc.SubmitChanges();
```

Izmena sloga u bazi podataka

- Preuzima se slog koji je potrebno promeniti iz baze podataka
- Vrši se izmena nad objektom koji reprezentuje slog
- Izmena se šalje u bazu podataka

```
IEnumerable<Student> res = from s in studenti where s.Id == 23 select s;
Student student = res.ElementAt(0); student.Ime = "Marko"; dc.SubmitChanges();
```

Brisanje sloga iz baze podataka

- Preuzima se slog koji je potrebno izbrisati iz baze podataka
- Objekat se izbacuje iz odgovarajuće tabele DataContext objekta
- Izmena se šalje u bazu podataka

```
IEnumerable <Student> res = from s in studenti where s.Id == 23 select s;
Student student = res.ElementAt(0); studenti.DeleteOnSubmit(student); dc.SubmitChanges();
```

Slanje izmena u bazu podataka

- Sve izmene se vrše nad objektima u memoriji, dok god se podaci eksplicitno ne pošalju u bazu podataka pozivom metode **SubmitChanges** klase **DataContext**
- U okviru metode utvrđuju se razlike između podataka u memoriji i podataka u bazi podataka
- Na osnovu utvrđenih razlika redom se formiraju SQL upiti koji izvršavaju odgovarajuće operacije nad bazom podataka

Upravljanje transakcijama

- Implicitno
  - Ako nije drugačije naznačeno, pri svakom pozivu metode **SubmitChanges** kreira se nova transakcija u okviru koje se obavljaju sve izmene koje metoda vrši (redom se svi SQL upiti izvršavaju u okviru ove transakcije)
- Eksplicitno
  - Moguće je eksplicitno kreirati novu transakciju (objekat klase **System.Data.Common.DbTransaction**)
  - Svaka **SubmitChanges** metoda se tada izvršava u okviru ove transakcije