Univerzitet u Istočnom Sarajevu

Elektrotehnički fakultet Istočno Sarajevo

**etf+grb**

**Paralelni računarski sistemi**

Izrada WPF aplikacije za praćenje statistike NBA lige

**Seminarski rad**

|  |  |
| --- | --- |
| Predmetni profesor: | Studenti: |
| Prof. dr Nikola Davidović | Anđela Andrijašević  Nedeljko Ilić |

Sadržaj

[1. Istorijski pregled 3](#_Toc89978511)

[2. Programski jezik C# 4](#_Toc89978512)

[3. Osnove programiranja u C# 6](#_Toc89978513)

[3.1. Tipovi podataka, deklaracija i definicija promjenljive 6](#_Toc89978514)

[3.2. Klasa, objekat 10](#_Toc89978515)

[4. .NET Framework 13](#_Toc89978516)

[4.1. Objektno - orijentisano programiranje (Klase, objekti i metode) 14](#_Toc89978517)

[4.2. Ključne riječi u C# 14](#_Toc89978518)

[4.3. Kreiranje konzolnih aplikacija u Microsoft Visual C# 2010 Professional 14](#_Toc89978519)

[5. Razvoj desktop aplikacija u .NET Framework-u 17](#_Toc89978520)

[Kreiranje novog WPF projekta 17](#_Toc89978521)

[6. Event-driven programiranje 20](#_Toc89978522)

[7. Višenitno programiranje u C# 20](#_Toc89978523)

[Tipovi niti 25](#_Toc89978524)

[Foreground Thread 25](#_Toc89978525)

[Background Thread 26](#_Toc89978526)

[Kreiranje niti 28](#_Toc89978527)

[Glavna nit 28](#_Toc89978528)

[Stanja i životni vijek niti 29](#_Toc89978529)

[Klasa *Thread* 30](#_Toc89978530)

[Prekidanje niti 31](#_Toc89978531)

[8. Aplikacija 32](#_Toc89978532)

[Main Window 32](#_Toc89978533)

[Prozor „utakmica“ 33](#_Toc89978534)

[Detalji o igraču 34](#_Toc89978535)

[9. Testiranje i paralelizam 35](#_Toc89978536)

[Intel Pentium P6000 1.87GHz 35](#_Toc89978537)

[Intel Core i3 2310M 2.1GHz 37](#_Toc89978538)

[Intel Celeron N3060 1.6Ghz 39](#_Toc89978539)

[10. Zaključak 41](#_Toc89978540)

[Literatura 42](#_Toc89978541)

[11. Popis slika 42](#_Toc89978542)

[12. Popis grafika 43](#_Toc89978543)

[13. Popis tabela 43](#_Toc89978544)

# Istorijski pregled

Programski jezik C spada u proceduralne programske jezike koji je razvijen u ranim [70-](http://hr.wikipedia.org/wiki/1970-ih) [im](http://hr.wikipedia.org/wiki/1970-ih) godinama [20. vijeka](http://hr.wikipedia.org/wiki/20._stolje%C4%87e). Autor ovog programskog jezika je [Dennis Ritchie](http://hr.wikipedia.org/wiki/Dennis_Ritchie), no značajan doprinos nastanku C-a dali su [Ken Thompson](http://hr.wikipedia.org/wiki/Ken_Thompson) kao autor programskog jezika B i [Martin](http://hr.wikipedia.org/w/index.php?title=Martin_Richards&action=edit&redlink=1) [Richards](http://hr.wikipedia.org/w/index.php?title=Martin_Richards&action=edit&redlink=1), autor programskog jezika [BCPL](http://hr.wikipedia.org/w/index.php?title=BCPL&action=edit&redlink=1). Dennis Ritchie je stvorio ovaj programski jezik za rješavanje praktičnih problema kodiranja [sistemskih programa](http://hr.wikipedia.org/w/index.php?title=Sistemski_program&action=edit&redlink=1) i jezgra [operativnog](http://hr.wikipedia.org/wiki/Operacijski_sustav) [sistema](http://hr.wikipedia.org/wiki/Operacijski_sustav) [UNIX](http://hr.wikipedia.org/wiki/UNIX), koji je praktički u cijelosti napisan u C-u.

Programski jezik C dosta se mijenjao tokom [godina](http://hr.wikipedia.org/wiki/Godina) te je u više navrata neformalno i formalno standardizovan. Prva važnija verzija poznata je pod nazivom "K&R C", što je [engl](http://hr.wikipedia.org/wiki/Engleski_jezik). skraćenica [prezimena](http://hr.wikipedia.org/wiki/Prezime) dva autora najpoznatijeg C priručnika "The C Programming Language", a to su [Brian Kernighan](http://hr.wikipedia.org/w/index.php?title=Brian_Kernighan&action=edit&redlink=1) i Dennis Ritchie. Prvo izdanje te vrlo sažete i precizno pisane [knjige](http://hr.wikipedia.org/wiki/Knjiga) koje datira iz [1978](http://hr.wikipedia.org/wiki/1978). godine ujedno je [de facto](http://hr.wikipedia.org/wiki/De_facto) standardizovalo jezik u 70-ima. Drugo izdanje iz [1988](http://hr.wikipedia.org/wiki/1988). godine opisuje "ANSI C", standard koji je [1983](http://hr.wikipedia.org/wiki/1983). godine definisao [američki](http://hr.wikipedia.org/wiki/SAD) nacionalni institut za standardizaciju, a koji je i danas najbolje podržan. Donedavno je standard bio ISO/IEC standard skraćeno poznat kao "C99", no krajem 2011. usvojen je ISO/IEC 9899:2011, poznat kao "C11", za koji su kompajleri još u razvoju.

Kao jedan od najvažnijih jezika u istoriji komercijalne [računarske](http://hr.wikipedia.org/wiki/Ra%C4%8Dunalo) industrije, C je do danas ostao jedini programski jezik prilagođen za sve računarske platforme, od malih sistema pa do mrežnih superračunara. Programi napisani u njemu vrlo su bliski načinu rada [hardvera](http://hr.wikipedia.org/wiki/Hardver) te u principu zahtijevaju od programera dobro razumijevanje rada [procesora](http://hr.wikipedia.org/wiki/Procesor), [memorije](http://hr.wikipedia.org/wiki/Memorija_(ra%C4%8Dunala)), [ulazno-](http://hr.wikipedia.org/w/index.php?title=Ulazno-izlazni_skolpovi&action=edit&redlink=1) [izlaznih sklopova](http://hr.wikipedia.org/w/index.php?title=Ulazno-izlazni_skolpovi&action=edit&redlink=1) itd. No, rad s registrima procesora i adresiranje memorije apstrahovani su pomoću koncepta varijabli i pokazivača što uz eksplicitne kontrolne strukture i funkcije znatno olakšava programiranje u odnosu na direktno programiranje u mašinskim jezicima.

Tokom [1980-ih](http://hr.wikipedia.org/wiki/1980-ih), [Bjarne Stroustrup](http://hr.wikipedia.org/wiki/Bjarne_Stroustrup) zajedno s drugim istraživačima u Bell Labs proširuje C dodavajući sposobnosti objektno orijentisanog programiranja, a naziv ovog novog programskog jezika je [C++](http://hr.wikipedia.org/wiki/C_Plus_Plus). Nažalost, ta je 100%-na kompatibilnost ujedno i razlog što su problemi koje programiranje u C-u nosi sa sobom naslijeđeni u C++-u. Efikasno i sigurno programiranje u C-u vrlo je zahtjevna vještina koja traži višegodišnje iskustvo pa je stoga C jezik koji se ne preporučuje početnicima, posebno ako im programiranje nije primarni posao.

Mnogobrojni problemi vezani prije svega za upravljanje memorijom koje programer mora sam eksplicitno kodirati razlog su da je danas većina novih korisničkih aplikacija napisana u nekom modernijem jeziku koji ima ugrađeno automatsko upravljanje memorijom (engl. garbage collection), ne dopušta direktan rad s memorijom pomoću pokazivača te ima podršku za upravljanje kodom odnosno njegovom okolinom za vrijeme njegova izvođenja. Danas postoji relativno rijetka potreba za pisanjem novih korisničkih aplikacija direktno u C-u, pa čak i u vrlo malim sistemima kao što su [mobilni telefoni](http://hr.wikipedia.org/wiki/Mobilni_telefon). Glavno područje njegove upotrebe su sistemski programi na strani servera (engl. servers), programi prevodioci (engl. compilers) i jezgra operativnih sistema (engl. operating system kernels), gdje je potreba za najvećom mogućom brzinom izvođenja, efikasnom kontrolom resursa i direktnom kontrolom hardvera od primarne važnosti.

C je jezik opšte namjene, što znači da se u njemu može napraviti apsolutno sve: od rješavanja [zadataka](http://hr.wikipedia.org/w/index.php?title=Zadatak&action=edit&redlink=1), do pisanja drajvera, [operativnih](http://hr.wikipedia.org/wiki/Operacijski_sustav) sistema, [tekst procesora](http://hr.wikipedia.org/wiki/Tekst_procesor) ili [igara](http://hr.wikipedia.org/wiki/Ra%C4%8Dunalna_igra). C, kao jezik, ni u čemu ne ograničava. Omogućuje i uključivanje naredbi pisanih asemblerski, zbog čega je zajedno s mogućnošću direktnog pristupa pojedinim [bitovima](http://hr.wikipedia.org/wiki/Bit), [bajtovima](http://hr.wikipedia.org/wiki/Bajt) ili cijelim blokovima memorije, pogodan za pisanje sistemskog [softvera](http://hr.wikipedia.org/wiki/Softver). Zbog tih karakteristika C je među popularnijim [programskim jezicima](http://hr.wikipedia.org/wiki/Programski_jezik) i koriste ga mnogi [programeri](http://hr.wikipedia.org/wiki/Programer). Rezultat toga je postojanje velikog broja prevodioca za C i alata te stalno dostupne pomoći na [internetu](http://hr.wikipedia.org/wiki/Internet). Programi napisani u C-u su prenosivi (mogu se prevoditi i izvršavati na različitim porodicama računara uz minimalne ili nikakve ispravke) i obično su vrlo brzi. Postoje mnogi [prevodioci](http://hr.wikipedia.org/wiki/Jezi%C4%8Dni_prevoditelj) za jezik C, a jedan od najšire korištenih je [GNU C Compiler](http://hr.wikipedia.org/wiki/GNU_C_Compiler).

# Programski jezik C#

C# je zreo programski jezik koji omogućava izradu aplikacija za Web sa svom potrebnom funkcionalnošću. Osim što pruža programski okvir za razvoj softvera, omogućava i podršku za pristupanje brojnim bazama podataka. On predstavlja jedan od jezika koji služe za izradu aplikacija koje mogu raditi pod .NET okruženjem. Kako predstavlja evoluciju jezika C i C++, kao takav je kreiran uz korišćenje svih prednosti ostalih jezika i otklanjanje njihovih mana. Sam razvoj aplikacije je jednostavniji u C# jeziku, a sam jezik je izuzetno moćan. S vremena na vreme njegov kod je mnogo razumljiviji u odnosu na druge jezike, ali je nešto robusniji i jednostavnije je otkloniti greške u njemu. Sam jezik nema ograničenja u pogledu toga kakve se aplikacije mogu napraviti u njemu, on koristi okruženje i samim tim nema ograničenja u vezi sa mogućim aplikacijama. Za programiranje u C# korišten je Visual Studio .NET, koji je mnogo olakšao razvoj same aplikacije. Visual Studio .NET pored toga što sadrži dizajnerske alatke za aplikacije, alatke za prikazivanje i navigaciju elemenata projekta, omogućava jednostavniju distribuciju i prilagođavanje projekta klijentima i obezbeđuje napredne tehnike otklanjanja grešaka pri razvoju projekta.

Programski jezik C# (C sharp) je proizvod kompanije Microsoft i nastao je kao odgovor na nedostatke postojećih jezika kao što su C, C++ i Visual Basic, u isto vrijeme kombinujući njihove dobre strane.

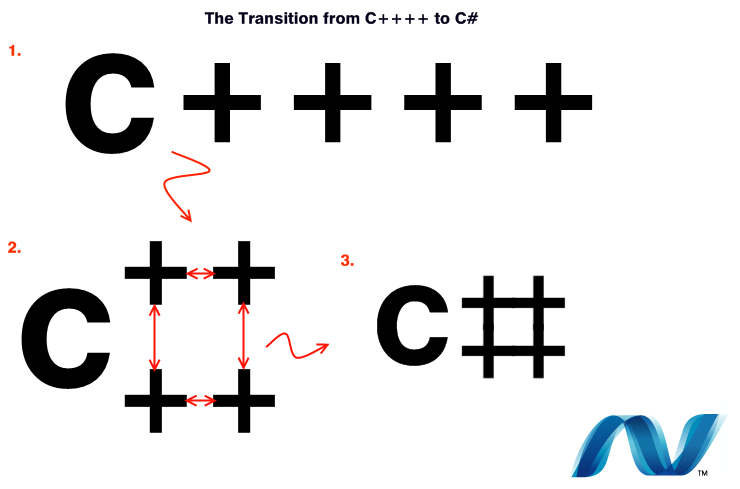
C# je potpuno objektno-orijentisani programski jezik, što znači da svi elementi unutar njega predstavljaju objekt. Objekt predstavlja strukturu koja sadrži elemente podataka, kao i metode i njihove međusobne interakcije.

Kao primjer jednostavnog C# programa ćemo navesti sljedeći kod:

*Using System; Class Primjer{*

*statis void Main()*

*{*



*Console.WriteLine("Pozdrav!");*

*}*

*}*

Nakon izvođenja ovog programskog koda, na ekranu se ispisuje poruka "Pozdrav!".

*Using System*; je kompajlerska naredba koja mu govori da aplikacija koristi skup elemenata pod nazivom *System*. Na ovaj način sve funkcije i klase unutar prostora (namespace) *System* postaju dostupne unutar aplikacije. Na primjer, klasi *Console* u našem primjeru pristupa se direktno iako se ona nalazi unutar klase System.

Slika 1:Tranzicija od C do C#

U suprotnom bismo joj morali pristupati na sljedeći način: *System.Console*;

Nadalje definišemo vlastitu klasu pod nazivom Primjer. Za ovo koristimo ključnu riječ *Class*, te vitičaste zagrade koje idu nakon imena klase, i definišu opseg klase. Drugim riječima, svi objekti koje definišemo unutar ovih zagrada pripadaće toj klasi.

Svaki program mora imati tačku u kojoj kreće sa izvršavanjem, i to je u C# aplikaciji funkcija *Main()*. U našem primjeru ta funkcija definisana je pomoću dvije ključne riječi: *static* i *void*.

Za kraj ostaje objasniti naredbu za ispis teksta korištenu u primjeru.

Ta naredba imena WriteLine sadržana je u klasi Console. Svim sastavnim elementima neke klase/objekta pristupamo na način da napišemo ime klase/objekta te nakon njega stavimo tačku i napišemo ime elementa (varijable, funkcije...) kojem pristupamo.

Funkcija WriteLine prima jedan argument, i to upravo tekst koji želimo ispisati. Konačno, naredba izgleda: Console.WriteLine("Pozdrav!");

Još je bitno napomenuti da u C# jeziku svaka linija završava tačka-zarezom (eng. semicolon)

# Osnove programiranja u C#

Programski jezik C# je nastao 2002. godine u „radionici“ čuvene kompanije Microsoft kao proizvod želje da se napravi objektno orijentisani programski jezik (poput Jave) koji omogućava relativno lako kreiranje aplikacija u grafičkom korisničkom okruženju (poput programskog jezika Borland Delphi). Ovaj programski jezik je jezik opšte primjene i namijenjen je izradi aplikacija za Microsoft .NET platformu.

Sintaksne konstrukcije programskog jezika C# su izgrađene na uobičajenoj azbuci (velika i mala slova abecede, cifre, specijalni znaci). Kao i u svakom drugom programskom jeziku postoje takozvane službene ili rezervisane riječi koje imaju svoje unaprijed definisano značenje i ne mogu se koristiti za imenovanje bilo kakvih podataka od strane programera. Ovdje navodimo skup rezervisanih riječi raspoređenih u leksikografskom redoslijedu:

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

Ovaj dio seminarskog rada govori o tipovima podataka kojima C# manipuliše kao i o konstrukciji osnovnih sintaksnih i semantičkih konstrukcija ovog programskog jezika.

# Tipovi podataka, deklaracija i definicija promjenljive

Svakom aplikacijom obrađujemo podatke. Svi podaci čuvaju se u memoriji računara u obliku nizova bistabilnih elemenata (0 i 1), bez obzira da li podatak predstavlja broj, riječ, sliku ili zvuk. Različite vrste podataka registruju se na različite načine. Takođe, nad različitim vrstama podataka su dozvoljene različite operacije. U zavisnosti od vrste podatka, mora se odvojiti određeni prostor za njegovo registrovanje i obezbediti izvršavanje određenih akcija

nad tim podatkom. Računaru moramo dati informaciju koja vrsta podatka je registrovana da bi ga na pravi način tumačio i obrađivao.

Potrebne informacije saopštavamo računaru tipom podataka. Tipom podataka definisan je:

o način registrovanja podataka u memoriji;

o skup mogućih vrijednosti tih podataka;

o skup mogućih akcija nad podacima.

Promjenljiva je ime (identifikator) memorijske lokacije u kojoj aplikacija čuva vrijednost. Identifikator (ime) koji programer definiše gradi se na osnovu sledećeg pravila: Identifikator je niz slova (velikih i malih) engleske abecede, donje crte (\_) i cifara. Ime mora da počne slovom ili donjom crtom.

Prema tome, primjeri identifikatora su *nazivPredmeta, ucenik1, Rezultat, radni\_dan, \_x, ...* dok

*nazivPredmeta, godisnje-doba, 1dan, ...*

nisu identifikatori.

Važno je napomenuti da C# (poput programskog jezika C) razlikuje velika i mala slova tako da identifikatori GodisnjeDoba i godisnjeDoba nisu isti.

U programskom jeziku C# promjenljive su podijeljene u dvije kategorije: vrijednosne i referentne promjenljive. Memorijska lokacija, imenovana promjenljivom vrijednosnog tipa, direktno sadrži vrijednost, dok lokacija imenovana promjenljivom referentnog tipa sadrži adresu lokacije u dinamičkoj memoriji u kojoj se čuva vrijednost. Za podatke osnovnih tipova (tj. sistemskih struktura) u programskom jeziku C# (int, double, char, bool…) ime promjenljive predstavlja vrijednost a za sistemske objekte i klase koje definiše programer, ime promjenljive predstavlja adresu lokacije u dinamičkoj memoriji na kojoj se ti objekti kreiraju i gdje postoje, pa im se pristupa indirektno. Referentna promjenljiva čuva memorijsku adresu podatka, tj. referencira pravu vrijednost.

Imenovanje podataka u aplikaciji i definisanje tipa podataka kojem pripadaju postižemo deklaracijom promjenljivih na sledeći način:

*<ime tipa><ime promjenljive1>, …, <ime promjenljiveN> ;*

Deklaracija promjenljive tipa vrijednosti u memoriji odvaja prostor potreban za registraciju podataka zadatog tipa, a deklaracija promjenljive referentnog tipa odvaja se prostor za registrovanje adrese memorijske lokacije na kojoj će se nalaziti objekat.

*int x,y; char p,q;*

*Ucenik ucenik1,ucenik2;*

Samom deklaracijom ne dodjeljuje se vrijednost promjenljivoj vrijednosnog tipa. Ako želimo da dodijelimo vrijednost promjenljivoj (da definišemo promjenljivu) koristimo operator = (operator dodjele) na sledeći način:

*<promenljiva>=<izraz>;*

Tip vrijednosti izraza mora se poklapati sa tipom promjenljive (ili se implicitnom konverzijom može konvertovati u tip promjenljive).

*int x,y; x=5; y=!145+x;*

*char p; p=’A’;*

Deklaracijom promjenljive referentnog tipa ne zauzima se memorija za vrijednost već samo za njenu referencu. Da bismo napravili prostor za vrijednost, upotrebljavamo izraz oblika

*new <ime referentnog tipa>(<lista parametara>)*

Korišćenjem operatora new odvaja se u dinamičkoj memoriji prostor za registrovanje vrijednosti tipa <ime referentnog tipa >. Ovaj operator vraća adresu dodjeljenog prostora koju onda možemo dodijeliti referentnoj promjenljivoj.

Primjer:

*mojRefTip x;*

*x= new mojRefTip(); Ucenik a;*

*a=new Ucenik(“Pera”, “Peric”);*

Promjenljivoj referentnog tipa x, odnosno a, dodjeljujemo adrese memorijskog prostora odvojenog korišćenjem operatora new. Pozivima metoda mojRefTip() i Ucenik(“Pera”, “Peric”), u prostoru odvojenom korišćenjem operatora new, postavljaju se odgovarajuće vrijednosti. Možemo zaključiti da se uvođenje promjenljive odvija u dva koraka:

o deklaracija promjenljive

Promjenljiva vrijednosnog tipa se pri deklaraciji ne inicijalizuje ukoliko to nije

eksplicitno navedeno: promjenljiva referentnog tipa se pri deklaraciji inicijalizuje

nulom;

o inicijalizacija promjenljive

Za promjenljive vrijednosnog tipa eksplicitno navodimo vrijednost ; za promjenljive

referentnog tipa operator new odvaja u dinamičkoj memoriji prostor za vrijednost

odgovarajućeg tipa.

U programskom jeziku C# je data mogućnost objedinjavanja ova dva koraka, što vrlo često koristimo.

*int x=32, y;*

*mojTip z= new mojTip();*

*Ucenik a=new Ucenik(“Pera”, “Peric”);*

Osnovni podaci referentnog tipa u programskom jeziku C# su klase a vrijednosnog strukture i enumeracijski tip. Promjenljiva vrijednosnog tipa direktno sadrži podatak, dok promjenljiva referentnog tipa sadrži referencu na vrijednost koja je smještena u posebnom memorijskom prostoru. Svaka promjenljiva vrijednosnog tipa sadrži sopstvenu kopiju podatka, pa operacije nad jednom promjenljivom nemaju efekta na drugu promjenljivu.

Dvije referentne promjenljive mogu sadržati adresu iste vrijednosti, pa operacije nad jednom referentnom promjenljivom mogu uticati na vrijednost na koju upućuje druga promjenljiva.

# Klasa, objekat

Do klasa dolazimo polazeći od pojedinačnih objekata. Posmatranjem objekata uočavamo njihova zajednička svojstva koja zovemo atributima.

Takođe, objektima možemo pridružiti iste „akcije“ koje u okviru klase nazivamo metodama klase. Njima opisujemo funkcionalnosti objekata klase. Klasu definišemo navođenjem rezervisane riječi class iza koje slijedi identifikator klase (ime klase). Posle imena, ako formiramo klasu koja nasljeđuje neku, prethodno definisanu klasu, navodimo ’:’ a zatim slijedi ime klase iz koje je izvedena. Zatim u vitičastim zagradama {} definišemo članove klase (atributi i metode).

class <imeKlase> :<imePrethodnoDefinisaneKlase>

*{*

*opis / definicija članova klase*

*}*

Na primjer, klasu učenika iz uvodnog primjera možemo definisati na sljedeći način:

*Class Ucenik*

*{*

*String ime, prezime;*

*DateTime datumRodjenja;*

*int razred;*

*int SkolskaGodina;*

*char odjeljenje;*

*int [] ocjene;*

*double prosjek()*

*{*

*…*

*}*

*void uci(Lekcija X)*

*{*

*…*

*}*

*}*

Atributima opisujemo određenu osobinu objekta (ime, prezime, datumRodjenja, razred, odjeljenje). Najčešće, različiti objekti iste klase imaju različite vrijednosti atributa. Često kažemo da vrijednosti atributa definišu stanje objekta. Pri opisu atributa moramo navesti tip kome taj atribut pripada (cjelobrojni, realni, znakovni, Form, Button) i ime atributa. Pri tome, navedeni tip je prethodno definisan (ugrađen u sistem ili definisan od strane programera).

*int a,b; //cjelobrojni atributi a i b*

*Button dugme;// atribut dugme, objekat ugrađene klase Button*

U programskom jeziku C#, po navođenju “//“, pišemo komentare napisanog koda. Počev od “//“ pa do kraja linije napisani komentar se pri prevođenju programa ignoriše. Ukoliko pišemo opširniji komentar, koji obuhvata više redova, od ostalog dijela koda odvajamo ga na početku sa “/\*“ a na kraju sa *“\*/“.*

Pristup atributima objekta u programskom jeziku C# realizujemo navođenjem imena objekta, znaka tačka (‘.’), pa imena atributa. Na primer, ako je X objekat klase Ucenik, atributu razred pristupamo sa X.razred. Metodom opisujemo ponašanje objekta u određenoj situaciji i pod određenim uslovima (npr. metod uci), ali i određujemo nove vrijednosti na osnovu osobina koje objekat posjeduje (npr. metod prosjek). Na taj način opisujemo funkcionalnost objekta.

Metoda klase je imenovani blok naredbi koji se sastoji iz zaglavlja i tijela metode. U zaglavlju navodimo povratni tip (ako metoda ne vraća vrijednost navodimo rezervisanu riječ void), zatim ime metode za kojim slijedi u malim zagradama spisak parametara metode. Za svaki parametar navodi se tip kome taj parametar pripada kao i ime parametra. Poslije zaglavlja u vitičastim zagradama navodimo tijelo metode koje se sastoji iz odgovarajućih naredbi programskog jezika C#.

*<povratni tip><ime metode>(<lista parametara>)*

*{ <tijelo metoda> }*

Svi atributi konkretnog objekta dostupni su i vidljivi svim metodama tog objekta i traju dok traje objekat. Za razliku od njih, promjenljive deklarisane u okviru neke od metoda (lokalne promjenljive), nastaju izvršavanjem te metode, vidljive su samo u okviru njega, i gase se završetkom te metode. Zato, ako je potrebno da koristimo vrijednost u okviru više metoda jedne klase (ili u više izvršavanja jedne metode), možemo je definisati kao atribut klase.

Neke od metoda se pokreću kao reakcija na događaje koji nastaju tokom izvršavanja aplikacije. Za takve metode kažemo da obrađuju događaje(engl. event handler).

Kada u toku izvršavanja aplikacije pritisnemo dugme za izlazak iz programa u naslovnoj liniji prozora u kome se izvršava aplikacija, automatski se pokreće metoda Closing u okviru koje možemo programirati akcije koje će spriječiti izlazak iz programa ukoliko uslovi za to nisu ispunjeni. Vrlo često u toku izvršavanja Windows aplikacija korisnik ima mogućnost da klikom na dugme usmjeri dalje izvršavanje aplikacije. Tok izvršavanja aplikacije nakon klika na dugme programer definiše metodom Click. Poziv metode objekta u programskom jeziku C# realizujemo na sledeći način:

*imeObjekta.imeMetode(lista vrijednosti parametara)*

Na primer, ako je x objekat klase Ucenik, metodu prosjek pozivamo

*x.prosjek()*

*a metoda uci x.uci(L)*

gde je L objekat klase Lekcija.

Svaka klasa sadrži metode koji omogućavaju kreiranje objekta, a može sadržati i metodu koja

„uništava“ objekat (destruktor). Metode koje kreiraju objekte zovemo konstruktorima. Konstruktori su sastavni dio svake klase i nose njeno ime. Pozivom konstruktora objekat

počinje svoj život. Klasa može imati jedan ili više konstruktora koji se razlikuju po listi parametara. Lista parametara najčešće sadrži vrijednosti kojima inicijalizujemo svojstva objekta, a može biti i prazna. Konstruktor neke klase pozivamo pri definisanju objekta iste klase, korišćenjem operatora new.

Objekat A klase Ucenik možemo definisati izrazom

*A=new Ucenik("Pera","Peric");*

Jedan od osnovnih koncepata objektno orijentisanog programiranja jeste kontrolisanje pristupa članovima objekta (atributima i metodama). Za svaki član klase (atribute i metode) moramo definisati nivo pristupa (vidljivosti, dostupnosti, zaštite). Na taj način definišemo koliko je taj član klase otvoren prema „spoljašnjem“ svijetu.

U programskom jeziku C# postoji više nivoa pristupa a mi ćemo, u početku, koristiti privatni (private) i javni (public ) pristup. Neki članovi klase mogu da se koriste interno, samo u metodama klase, i metode drugih klasa im ne mogu pristupiti. Za takve članove klase kažemo da su privatni i pri njihovom definisanju navodimo rezervisanu riječ private. Privatnim članovima klase mogu pristupati samo metode te klase i to je najviši nivo zaštite.

Pri opisu javnih članova klase navodimo rezervisanu riječ public . Takvi članovi klase su dostupni svijetu van klase i pristup tim članovima je potpuno slobodan. Podrazumijevani pristup članovima klase je privatni, što znači da su članovi klase za koje nije naveden nivo pristupa, privatni članovi. Uobičajeno je da su atributi privatni, a metode javni članovi klase.

Ukoliko postoji potreba da se izvan klase pristupa privatnim atributima, potrebno je definisati javne metode ili javno svojstvo (engl. property) u okviru klase koji služe za komunikaciju sa privatnim atributom, odnosno omogućava postavljanje vrijednosti ili čitanje vrijednosti atributa. Pri definisanju članova klase, posle nivoa pristupa, možemo navesti i službenu riječ static kojom definišemo da odgovarajući član postoji i da se koristi i bez kreiranja konkretnih instanci klase.

Takvim članovima pristupamo navođenjem imena klase umjesto imena objekta. Bez obzira na broj instanci klase koja sadrži atribut static, postoji tačno jedna kopija tog atributa u aplikaciji. U navedenom primjeru klase Ucenik, član SkolskaGodina možemo definisati kao static jer je vrijednost ista za sve učenike, i u tom slučaju tom atributu pristupamo Ucenik.SkolskaGodina.

# .NET Framework

.NET Framework je revulucionarna platofrma koju je kreirao Microsoft za razvoj aplikacija. To je takva platforma na kojoj mogu raditi različite vrste aplikacija (Windows aplikacije, Web aplikacije, Web servisi...). .NET Framework je dizajniran tako da može koristiti bilo koji programski jezik kao što su C++, Visual Basic, Jscript, čak i ranije verzije Cobol-a. Takođe je moguće da se kombinuju pojedini dijelovi koda iz različitih programskih jezika. Npr. kod napisan u C# se može kombinovati sa kodom iz Visual Basic-a.

.NET Framework sadrži veliku biblioteku koda koji može da koristi klijentski jezik (kao što je C#) koristeći tehnike objektno-orijentisanog programiranja. Ova biblioteka se sastoji od različitih modula, pri čemu je moguće koristiti pojedine dijelove u zavisnosti od potrebe. Tako npr. neki moduli sadrže blokove za kreiranje windows aplikacija, drugi dio za mrežno programiranje, a neki drugi za Web razvoj. Neki moduli su podijeljeni u neke specifičnije podmodule.

# Objektno - orijentisano programiranje (Klase, objekti i metode)

***Objekat*** je primjerak (instanca) klase. ***Klasa*** predstavlja skup podataka i instrukcija koje se izvode na nekim podacima. Klasa je skup pravila koji određuju kako će se formirati neki objekat. Svaka klasa ima svoje članove i to su podaci i kod. Podaci koji se definišu u klasi obično se nazivaju polja (fields). Ovde se koriste termini kao što su funkcija (***metoda***) koja predstavlja kod programa koji se izvršava nad podacima. Metode su ustvari podprogrami i oni mogu imati svojstva, događaje i konstruktore. Metode se izvršavaju nad poljima koja su definisana u klasi.

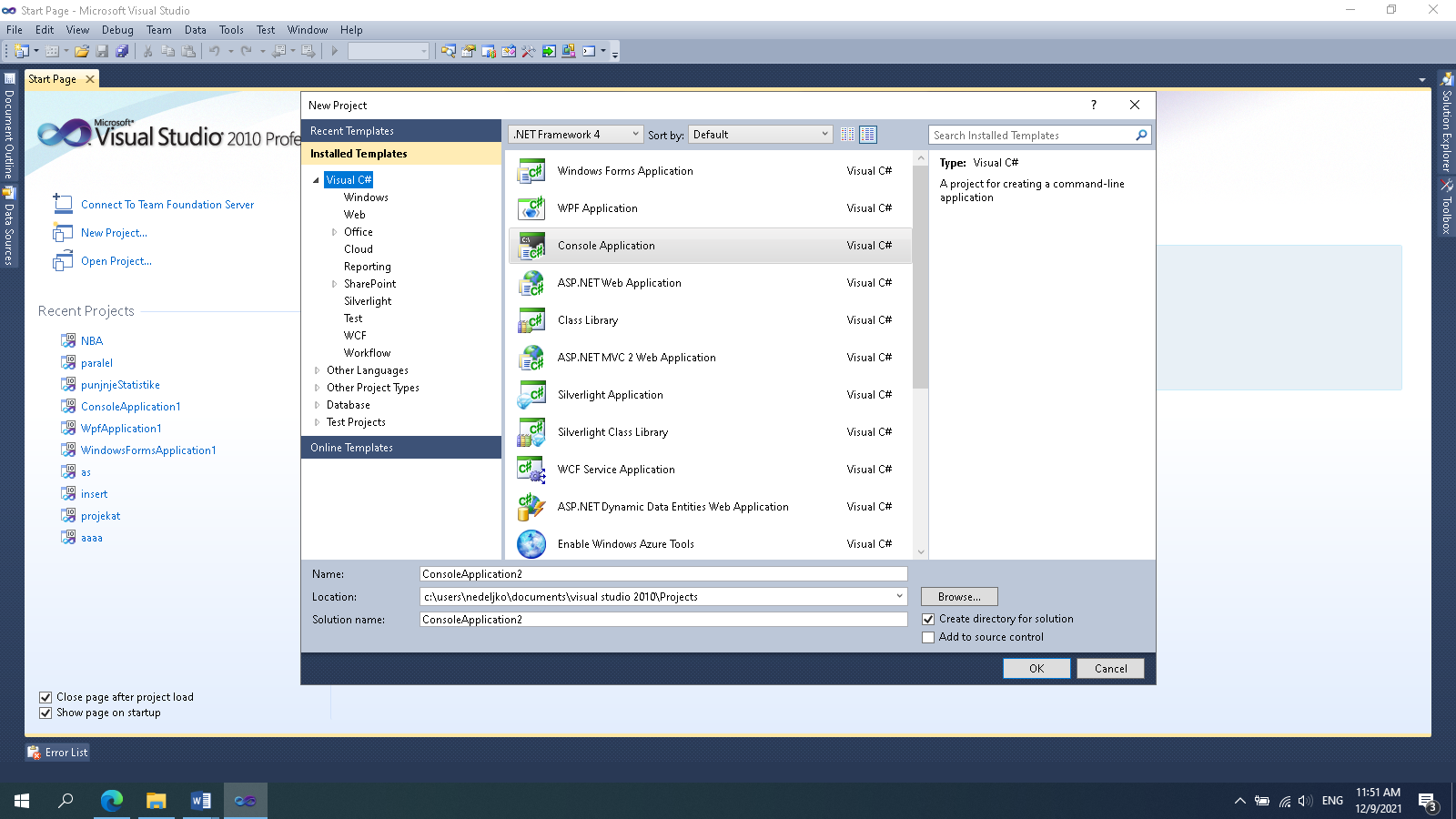
# Ključne riječi u C#

|  |  |  |  |
| --- | --- | --- | --- |
| [abstract](http://msdn.microsoft.com/en-us/library/sf985hc5.aspx) | [as](http://msdn.microsoft.com/en-us/library/cscsdfbt.aspx) | [base](http://msdn.microsoft.com/en-us/library/hfw7t1ce.aspx) | [bool](http://msdn.microsoft.com/en-us/library/c8f5xwh7.aspx) |
| [break](http://msdn.microsoft.com/en-us/library/adbctzc4.aspx) | [byte](http://msdn.microsoft.com/en-us/library/5bdb6693.aspx) | [case](http://msdn.microsoft.com/en-us/library/06tc147t.aspx) | [catch](http://msdn.microsoft.com/en-us/library/0yd65esw.aspx) |
| [char](http://msdn.microsoft.com/en-us/library/x9h8tsay.aspx) | [checked](http://msdn.microsoft.com/en-us/library/74b4xzyw.aspx) | [class](http://msdn.microsoft.com/en-us/library/0b0thckt.aspx) | [const](http://msdn.microsoft.com/en-us/library/e6w8fe1b.aspx) |
| [continue](http://msdn.microsoft.com/en-us/library/923ahwt1.aspx) | [decimal](http://msdn.microsoft.com/en-us/library/364x0z75.aspx) | [default](http://msdn.microsoft.com/en-us/library/25tdedf5.aspx) | [delegate](http://msdn.microsoft.com/en-us/library/900fyy8e.aspx) |
| [do](http://msdn.microsoft.com/en-us/library/370s1zax.aspx) | [double](http://msdn.microsoft.com/en-us/library/678hzkk9.aspx) | [else](http://msdn.microsoft.com/en-us/library/5011f09h.aspx) | [enum](http://msdn.microsoft.com/en-us/library/sbbt4032.aspx) |
| [event](http://msdn.microsoft.com/en-us/library/8627sbea.aspx) | [explicit](http://msdn.microsoft.com/en-us/library/xhbhezf4.aspx) | [extern](http://msdn.microsoft.com/en-us/library/e59b22c5.aspx) | [false](http://msdn.microsoft.com/en-us/library/67bxt5ee.aspx) |
| [finally](http://msdn.microsoft.com/en-us/library/zwc8s4fz.aspx) | [fixed](http://msdn.microsoft.com/en-us/library/f58wzh21.aspx) | [float](http://msdn.microsoft.com/en-us/library/b1e65aza.aspx) | [for](http://msdn.microsoft.com/en-us/library/ch45axte.aspx) |
| [foreach](http://msdn.microsoft.com/en-us/library/ttw7t8t6.aspx) | [goto](http://msdn.microsoft.com/en-us/library/13940fs2.aspx) | [if](http://msdn.microsoft.com/en-us/library/5011f09h.aspx) | [implicit](http://msdn.microsoft.com/en-us/library/z5z9kes2.aspx) |
| [in](http://msdn.microsoft.com/en-us/library/ttw7t8t6.aspx) | [in (generic modifier)](http://msdn.microsoft.com/en-us/library/dd469484.aspx) | [int](http://msdn.microsoft.com/en-us/library/5kzh1b5w.aspx) | [interface](http://msdn.microsoft.com/en-us/library/87d83y5b.aspx) |
| [internal](http://msdn.microsoft.com/en-us/library/7c5ka91b.aspx) | [is](http://msdn.microsoft.com/en-us/library/scekt9xw.aspx) | [lock](http://msdn.microsoft.com/en-us/library/c5kehkcz.aspx) | [long](http://msdn.microsoft.com/en-us/library/ctetwysk.aspx) |
| [namespace](http://msdn.microsoft.com/en-us/library/z2kcy19k.aspx) | [new](http://msdn.microsoft.com/en-us/library/51y09td4.aspx) | [null](http://msdn.microsoft.com/en-us/library/edakx9da.aspx) | [object](http://msdn.microsoft.com/en-us/library/9kkx3h3c.aspx) |
| [operator](http://msdn.microsoft.com/en-us/library/s53ehcz3.aspx) | [out](http://msdn.microsoft.com/en-us/library/t3c3bfhx.aspx) | [out (generic modifier)](http://msdn.microsoft.com/en-us/library/dd469487.aspx) | [override](http://msdn.microsoft.com/en-us/library/ebca9ah3.aspx) |
| [params](http://msdn.microsoft.com/en-us/library/w5zay9db.aspx) | [private](http://msdn.microsoft.com/en-us/library/st6sy9xe.aspx) | [protected](http://msdn.microsoft.com/en-us/library/bcd5672a.aspx) | [public](http://msdn.microsoft.com/en-us/library/yzh058ae.aspx) |
| [readonly](http://msdn.microsoft.com/en-us/library/acdd6hb7.aspx) | [ref](http://msdn.microsoft.com/en-us/library/14akc2c7.aspx) | [return](http://msdn.microsoft.com/en-us/library/1h3swy84.aspx) | [sbyte](http://msdn.microsoft.com/en-us/library/d86he86x.aspx) |
| [sealed](http://msdn.microsoft.com/en-us/library/88c54tsw.aspx) | [short](http://msdn.microsoft.com/en-us/library/ybs77ex4.aspx) | [sizeof](http://msdn.microsoft.com/en-us/library/eahchzkf.aspx) | [stackalloc](http://msdn.microsoft.com/en-us/library/cx9s2sy4.aspx) |
| [static](http://msdn.microsoft.com/en-us/library/98f28cdx.aspx) | [string](http://msdn.microsoft.com/en-us/library/362314fe.aspx) | [struct](http://msdn.microsoft.com/en-us/library/ah19swz4.aspx) | [switch](http://msdn.microsoft.com/en-us/library/06tc147t.aspx) |
| [this](http://msdn.microsoft.com/en-us/library/dk1507sz.aspx) | [throw](http://msdn.microsoft.com/en-us/library/1ah5wsex.aspx) | [true](http://msdn.microsoft.com/en-us/library/eahhcxk2.aspx) | [try](http://msdn.microsoft.com/en-us/library/0yd65esw.aspx) |
| [typeof](http://msdn.microsoft.com/en-us/library/58918ffs.aspx) | [uint](http://msdn.microsoft.com/en-us/library/x0sksh43.aspx) | [ulong](http://msdn.microsoft.com/en-us/library/t98873t4.aspx) | [unchecked](http://msdn.microsoft.com/en-us/library/a569z7k8.aspx) |
| [unsafe](http://msdn.microsoft.com/en-us/library/chfa2zb8.aspx) | [ushort](http://msdn.microsoft.com/en-us/library/cbf1574z.aspx) | [using](http://msdn.microsoft.com/en-us/library/zhdeatwt.aspx) | [virtual](http://msdn.microsoft.com/en-us/library/9fkccyh4.aspx) |
| [void](http://msdn.microsoft.com/en-us/library/yah0tteb.aspx) | [volatile](http://msdn.microsoft.com/en-us/library/x13ttww7.aspx) | [while](http://msdn.microsoft.com/en-us/library/2aeyhxcd.aspx) |  |

Tabela 1: Ključne riječi u C#

# Kreiranje konzolnih aplikacija u Microsoft Visual C# 2010 Professional

Na slici 2 je prikazana startna strana, na kojoj se mogu odabrati ponuđene mogućnosti za formiranje novog projekta izborom opcije *New Project* ili otvaranje već postojećeg projekta izborom opcije *Open Project*.



Slika 2: Izgled startnog prozora Microsoft Visual C# 2010 Express

Pri kreiranju konzolne aplikacije izabere se opcija sa padajućeg menija File→New Project. Zatim se u okviru panela Templates izabere Console Application. Potom se dobija sledeće:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace ConsoleApplication2

{

class Program

{

static void Main(string[] args)

{

}

}

}

Da bi aplikacija bilo šta radila potrebno je dodati bar jednu liniju koda ispod glavne funkcije static void Main(string[] args).

***Primjer 1. Ispisati tekst na ekranu.***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace ConsoleApplication2

{

class Primjer

{

static void Main(string[] args)

{

Console.WriteLine("Jednostavan C# program.");

Console.ReadKey();

}

}

}

namespace predstavlja .NET način da se kod i njegov sadržaj jedinstveno identifikuju. Takođe se koristi za kategorizaciju elemenata u .NET Framework-u. namespace je deklarisan za kod aplikacije koja se u ovom slučaju naziva ConsoleApplication1.

using System;-predstavlja korištenje System namespace (odnosi se na biblioteku klasa .NET Framework, čiju biblioteku koristi C#). Za veće projekte može se kreirati sopstveni namespace.

class Primjer{ -definicija klase Primer. Početak klase počinje sa otvorenom vitičastom zagradom {, a završava se sa }.

static void Main()-počinje glavna metoda, gde se izvršava program. Ova metoda počinje sa static, što znači da je statički i void - ne vraća vrijednost.

Console.WriteLine("Jednostavan C# program."); -WriteLine() je funkcija koja kao rezultat vraća string koji joj je proslijeđen. Console- podržava konzolni I/O. U kombinaciji se kompajleru govori da je WriteLine() članica Console klase.

Izrazi u C# se zvršavaju sa tačka-zarezom (;).

Console.ReadKey(); - funkcija za unos nekog znaka sa tastature (za prekid rada programa).

Pri izvršavanju aplikacije pojaviće se u komand propmptu tekstualna poruka:

*Jednostavan C# program.*

# Razvoj desktop aplikacija u .NET Framework-u

Karakteristika *Windows* aplikacija je bogat grafički korisnički interfejs (engl. *GUI-Graphical User Interface*). *Windows* aplikacije se sastoje od jedne ili više formi na kojima se nalaze kontrole (na primjer, dugmad, labele (natpisi), combobox, listbox, edit polja, itd.). Svaka forma i kontrola ima sposobnost prihvatanja odgovarajućih događaja koje proizvodi korisnik (npr. klik lijevim tasterom miša ili samo klik, dupli lijevi klik (dupli klik lijevim tasterom miša), desni klik, prevlačenje miša preko kontrole itd.) ili pak sam sistem tako da se korišćenje jedne *Windows* aplikacije svodi na komunikaciju sa korisnikom, prepoznavanjem akcija korisnika i generisanjem odgovarajućeg odziva.

Sve akcije se jednim imenom nazivaju događajima (engl. *events*) tako da se ovakav način programiranja sreće i pod nazivom programiranje upravljano događajima. Ovaj vid programiranja se zbog ove karakteristike odlikuje i drugačijim pristupom u razvoju aplikacije u odnosu na npr. proceduralno programiranje. Naime, najprije se generiše izgled korisničkog interfejsa, a zatim se određuju događaji za pojedine forme i kontrole na koje će sistem da reaguje tj. da da neki odziv. Na kraju se za svaki događaj piše odgovarajuća funkcija kojom se definiše odziv aplikacije na neku akciju korisnika.

Pošto se u prvi plan stavlja interfejs tj. forme i kontrole koje su više manje standardizovane po pitanju funkcionalnosti, savremena okruženja za razvoj *Windows* aplikacija nude unaprijed definisane klase kojim su opisane forme i kontrole.

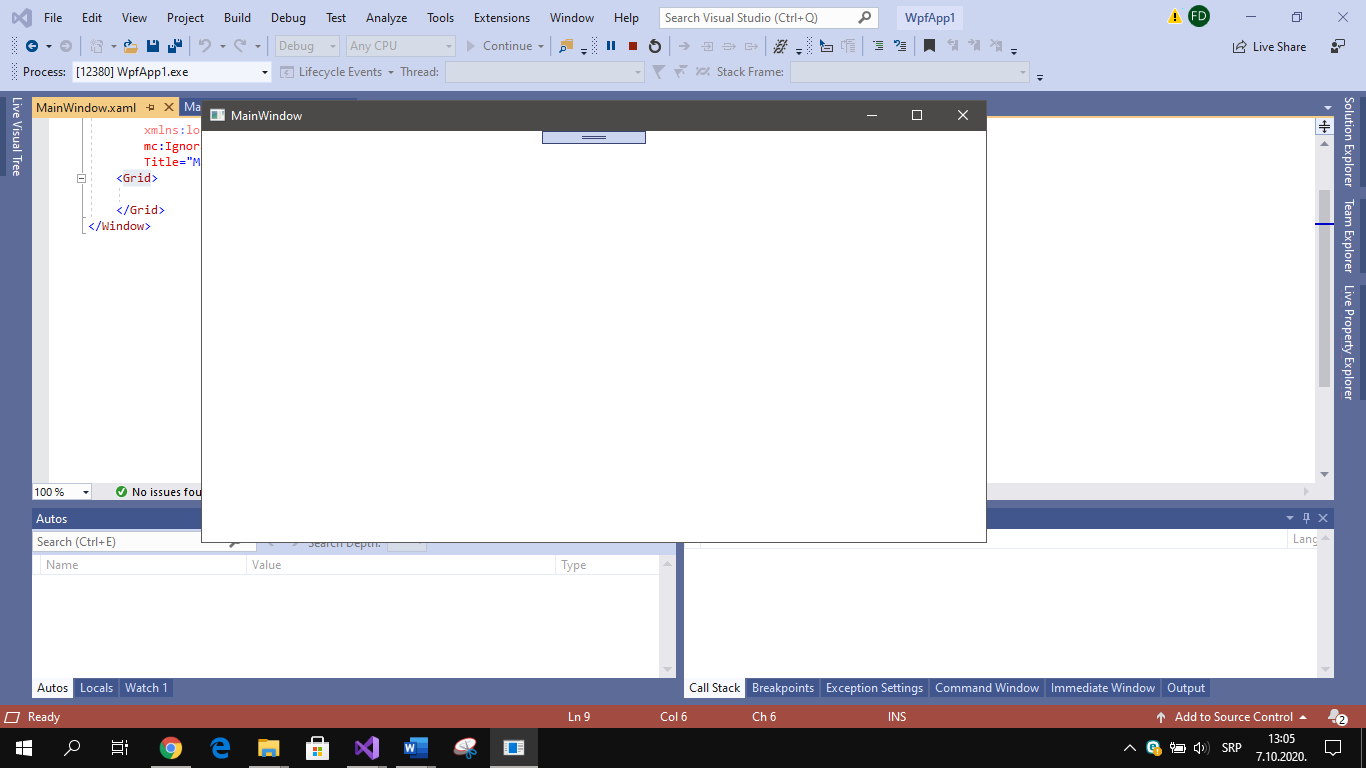
Veoma često se ovakav razvoj Windows aplikacija naziva i vizuelnim programiranjem. Takva situacija je i sa okruženjima *Visual Studio* koja nude bogat skup unaprijed definisanih biblioteka klasa.

Kako je broj tih biblioteka odnosno klasa veliki objasnićemo samo one najčešće korišćene. Ako se ima u vidu da je princip koji je korišćen u realizaciji ovih biblioteka isti upoznavanje sa svojstvima ostalih ne bi trebao da predstavlja veći problem.

## Kreiranje novog WPF projekta

Kreiranje projekta jedne *Windows* aplikacije je slično kreiranju projekta konzolne aplikacije, s tom razlikom da se pri izboru vrste aplikacije odabere ”*Windows Presentation Foundation*”.

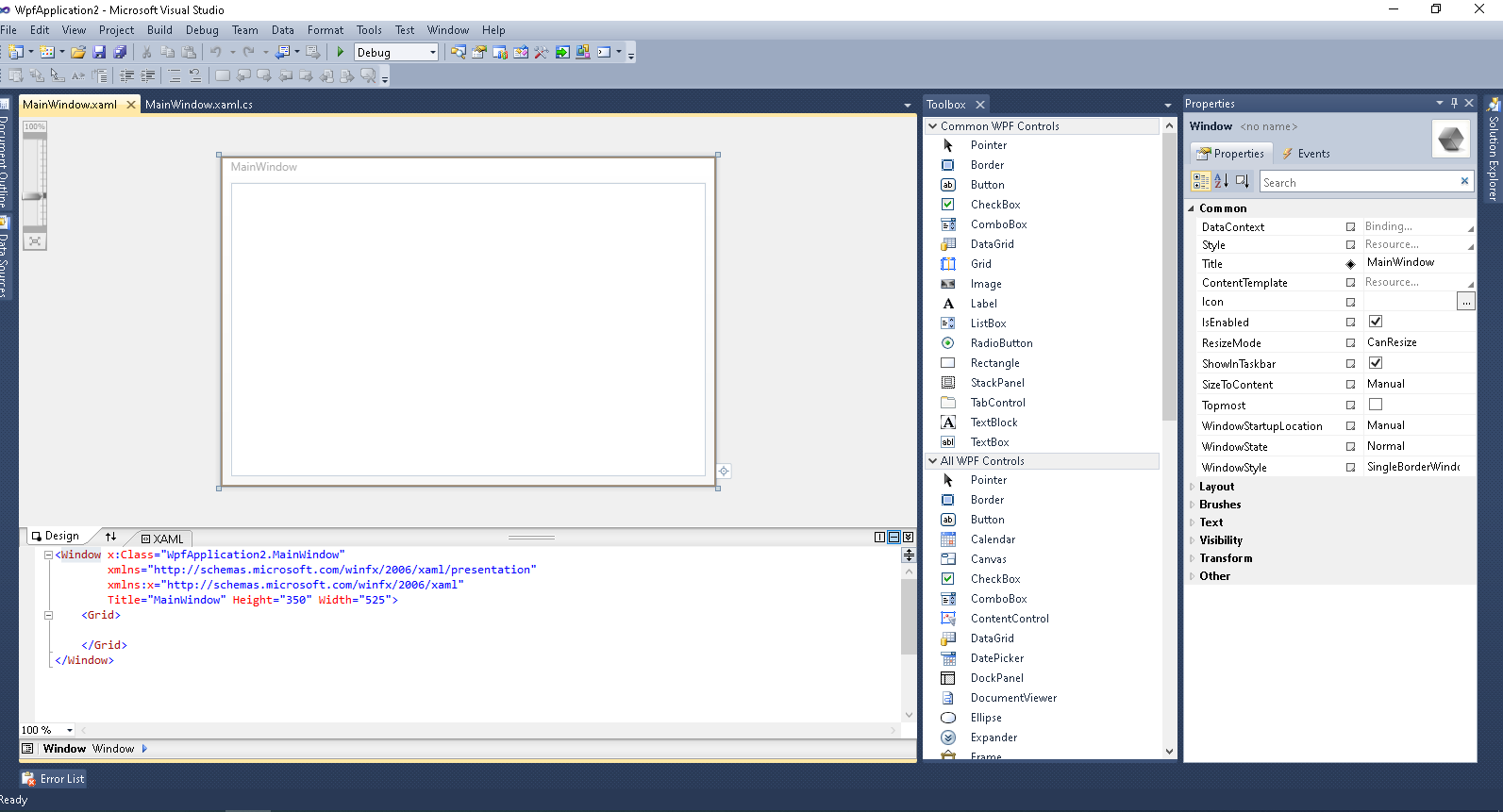
Po kreiranju novog projekta može se uočiti da se za razliku od slučaja kada smo kreirali konzolnu aplikaciju u centralnom dijelu pojavila prazna forma (slika 2) koja predstavlja početnu formu.



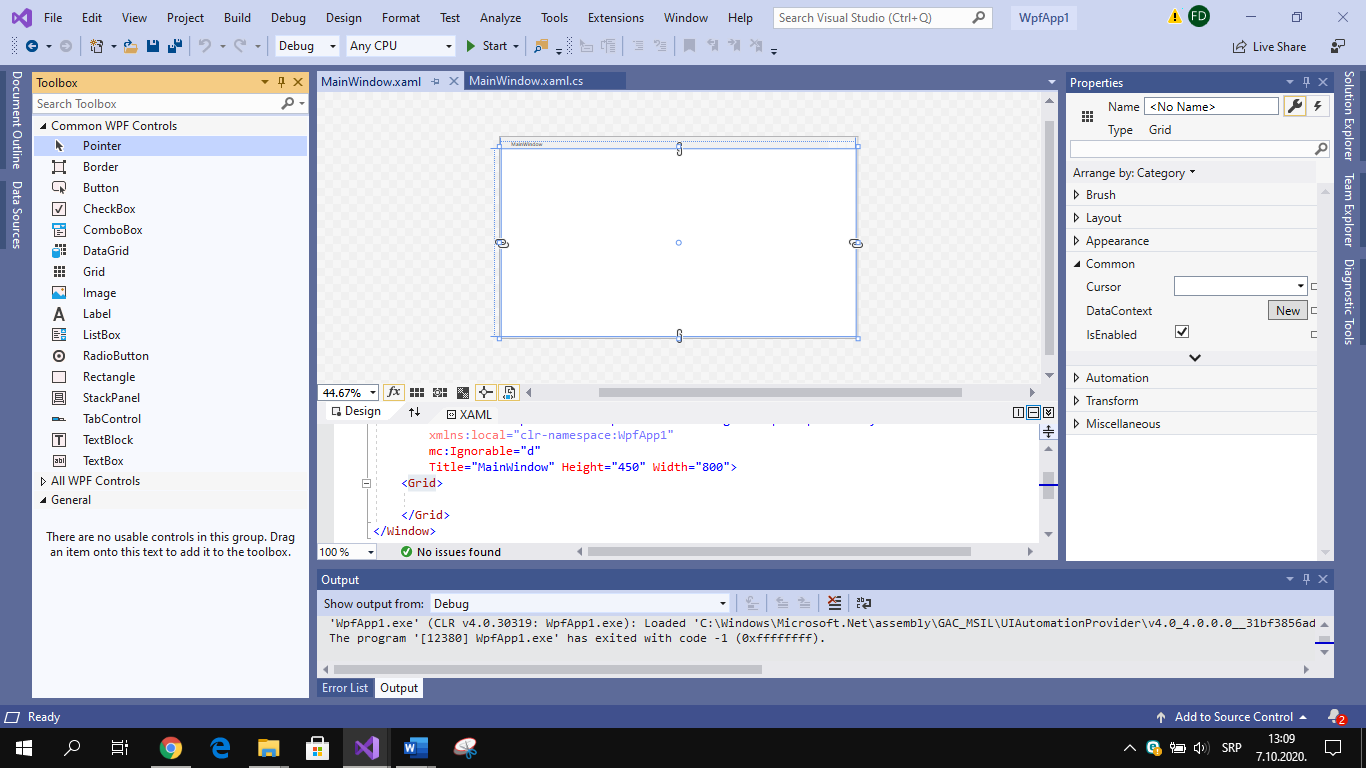
Slika 3: Prazna forma

Pokretanjem novog *Windows* projekta samo okruženje je odradilo sav posao umjesto nas i kreiralo glavnu formu aplikacije koja predstavlja početnu tačku tj. osnovu u razvoju bilo koje *Windows* aplikacije. Znači, našu buduću aplikaciju ćemo da razvijamo tako što ćemo na osnovnu formu da dodamo neke elemente interfejsa a potom i da ispišemo kod za odgovarajuće događaje forme odnosno dodatih elemenata interfejsa (kontrola).

Dodavanje elemenata interfejsa tj. kontrola (npr. dugme, labela, itd.) na formu vrši se prevlačenjem iz *Toolboxa* (paleta sa alatkama) u kome se nalaze kontrole grupisane u više grupa (paleta).



Slika 4: Razvojno okruženje za Windows aplikacije



Slika 5: Toolbox sa osnovnim WPF kontrolama

# Event-driven programiranje

Sada se u prvi plan stavlja izgled interfejsa tako da je prva etapa u kreiranju neke *Windows* aplikacije zapravo kreiranje izgleda interfejsa aplikacije.

Nakon toga dolazi druga etapa gdje se za svaki dogadjaj (npr. klik lijevim dugmetom miša, dupli klik lijevim dugmetom miša, klik desnim dugmetom miša, pritisak na tab i slično) koji se vezuje za kontrole interfejsa formiranog u prvoj etapi generiše kod funkcija koje se vezuju za događaj, tj. koje se izvršavaju kada se registruje događaj.

Znači programiranje se zapravo svodi na pisanje koda funkcija koje treba da se aktiviraju kada se desi neki događaj. Zato se ovaj pristup programiranju naziva i programiranje vođeno ili upravljano događajima.

Na prvi pogled reklo bi se da se sada programira lokalno jer programiramo funkcionalnosti koje treba da se dese kao odgovor na neki događaj (najčešće akciju korisnika programa). Ovo je u principu mnogo jednostavniji pristup, jer se globalno planiranje svodi na lokalno kreiranje funkcionalnosti.

Korišćenje aplikacije je niz akcija korisnika koje očekuju adekvatan odziv aplikacije. Stoga se i samo kreiranje aplikacije svodi na: kreiranje vizuelnog izgleda aplikacije tj. grafičkog korisničkog interfejsa upotrebom opštepoznatih *Windows* kontrola i pisanja odgovarajućeg koda kojim se opisuje željeno ponašanje tj. odgovor aplikacije na pojedine akcije korisnika. Oba ova koraka su izuzetno važna i veoma često se oni u toku realizacije aplikacije prepliću tj. izgenerišu se neke forme sa nekim kontrolama, a zatim se ispiše odgovarajući kod za *evente* kontrola, a nakon toga isti postupak ponovi za neke nove forme i kontrole itd. tj. inkrementalno i iterativno.

Dobra strana je što se od same osnovne forme pa nadalje uvijek ima neka verzija aplikacije sa odgovarajućim korisničkim interfejsom. Poseban akcenat se na ovaj način stavlja na projektovanje tj. dizajn interfejsa jer on određuje u krajnjoj mjeri kompletan način funkcionisanja cijele aplikacije.

Znači, grubo se može reći da se razvoj jedne *Windows* aplikacije odvija tako što se najprije isprojektuje interfejs koji obuhvata i popis svih događaja i akcija koje treba da slijede te događaje. Nakon toga se u drugoj fazi piše kod za funkcije koje se pozivaju po aktiviranju događaja. Događaji se po pravilu vezuju za elemente grafičkog korisničkog interfejsa aplikacije.

# Višenitno programiranje u C#

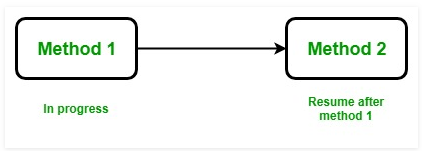
*Multitasking* je istovremeno izvršavanje više zadataka ili procesa u određenom vremenskom intervalu. Windows operativni sistem je primjer *multitasking*-a jer je u stanju da pokreće više od jednog procesa u isto vrijeme kao što je pokretanje Google Chrome-a, Notepad-a, VLC player-a itd. u isto vrijeme.

Operativni sistem koristi termin poznat kao proces za izvršavanje svih ovih aplikacija u isto vrijeme. Proces je dio operativnog sistema koji je odgovoran za izvršavanje aplikacije. Svaki program koji se izvršava na vašem sistemu je proces i za pokretanje koda unutar aplikacije proces koristi termin poznat kao nit.

Nit je lagan proces, ili drugim riječima, nit je jedinica koja izvršava kod u okviru programa. Dakle, svaki program ima logiku i nit koja je odgovorna za izvršavanje ove logike.

Svaki program po *defaultu* nosi jednu nit za izvršavanje logike programa i nit je poznata kao Glavna nit, tako da je svaki program ili aplikacija po defaultu jedan navojni model. Ovaj jednostruki model ima manu. Jedna nit pokreće sav proces prisutan u programu u sinhronizaciji, znači jedan za drugim. Dakle, drugi proces čeka dok prvi proces ne dovrši svoje izvršenje, troši više vremena u obradi.

Na primjer, imamo klasu nazvanu Geek i ova klasa sadrži dvije različite metode, tj. *metod1*, *metod2*. Sada je glavna nit odgovorna za izvršavanje svih ovih metoda, tako da glavna nit izvršava sve ove metode jednu po jednu.



Slika 6: Izvršavanje programa koji koristi samo glavnu nit

Primjer:

// C# program kojim se ilistruje koncept single thread modela

using System;

using System.Threading;

public class Geek

{

//statička metoda 1

public static void method1()

{

//Ispis brojeva od 0 do 10

for(int i=0; i<=10; i++)

{

Console.Writeline(“Method1 is {0}” i);

//Kada je vrijednost brojača i jednaka 5 onda program prelazi u stanje

// mirovanja na 6 sekundi i nakon 6 sekundi nastavlja sa radom

if(i==5)

{

Thread.Sleep(6000);

}

}

}

//statička metoda 2

public static void method2()

{

//Ispis brojeva od 0 do 10

for(int j=0; j<=10; j++)

{

Console.Writeline(“Method2 is {0}” j);

}

}

}

//Driver klasa

public class GFC {

//Glavna metoda

static public void Main()

{

//Poziv statičkih metoda

Geek.method1();

Geek.method2();

}

}

Ispis na konzolu:

Method1 is: 0

Method1 is: 1

Method1 is: 2

Method1 is: 3

Method1 is: 4

Method1 is: 5

Method1 is: 6

Method1 is: 7

Method1 is: 8

Method1 is: 9

Method1 is: 10

Method2 is: 0

Method2 is: 1

Method2 is: 2

Method2 is: 3

Method2 is: 4

Method2 is: 5

Method2 is: 6

Method2 is: 7

Method2 is: 8

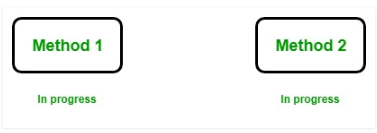
Method2 is: 9

Method2 is: 10

Ovdje se prije svega izvršava *metod1*. U *metod1*, petlja počinje od 0 kada je vrijednost i jednaka 5, tada metoda prelazi u stanje mirovanja 6 sekundi i nakon 6 sekundi nastavlja proces i ispisuje preostalu vrijednost dok se *metod2* ne nalazi u stanju čekanja. *metod2* započinje sa radom kada *metod1* završi zadati zadatak. Da bi se prevazišao nedostatak modela s jednim navojem uveden je *multithreading*.

*Multithreading* je proces koji sadrži više niti unutar jednog procesa. Ovdje svaka nit izvodi različite aktivnosti.

Na primjer, imamo klasu i ovaj poziv sadrži dvije različite metode, a sada se korištenjem *multithreading*-a svaka metoda izvodi zasebnom niti. Dakle, glavna prednost *multithreading*-a je da radi istovremeno, znači da se istovremeno izvršava više zadataka. Takođe, prednost je i maksimizacija upotrebe CPU-a jer *multithreading* radi na konceptu vremenskog dijeljenja znači da svaka nit uzima svoje vrijeme za izvršenje i ne utiče na izvršenje druge niti. Ovaj vremenski interval je datoperativnim sistemom.



Slika 7: Izvršavanje programa koji koristi višenitnost

Primjer:

// C# program za ilustraciju koncepta single threaded modela

using System;

using System.Threading;

public class GFG

{

//statička metoda 1

public static void method1()

{

//Ispisivanje brojeva od 0 do 10

for(int i=0; i<=10; i++)

{

Console.Writeline(“Method1 is {0}” i);

//Kada je vrijednost brojača i jednaka 5 onda program prelazi u stanje

// mirovanja na 6 sekundi i nakon 6 sekundi nastavlja sa radom

if(i==5)

{

Thread.Sleep(6000);

}

}

}

//statička metoda 2

public static void method2()

{

//Ispisivanje brojeva od 0 do 10

for(int j=0; j<=10; j++)

{

Console.Writeline(“Method2 is {0}” j);

}

}

//Glavna metoda

static public void Main()

{

Thread thr1 = new Thread(method1);

Thread thr2 = new Thread(method2);

thr1.Start();

thr2.Start();

}

}

Ispis na konzolu:

Method1 is: 0

Method1 is: 1

Method1 is: 2

Method1 is: 3

Method2 is: 0

Method2 is: 1

Method2 is: 2

Method2 is: 3

Method2 is: 4

Method2 is: 5

Method2 is: 6

Method2 is: 7

Method2 is: 8

Method2 is: 9

Method2 is: 10

Method1 is: 4

Method1 is: 5

Method1 is: 6

Method1 is: 7

Method1 is: 8

Method1 is: 9

Method1 is: 10

Ovdje kreiramo i inicijalizujemo dvije niti, tj. *thr1* i *thr2* koristeći *Thread* klasu. Sada koristite *thr1.Start ();* i *thr2.Start ();* počinjemo izvršavanje obje niti. Sada obje niti rade istovremeno i obrada *thr2* ne zavisi od obrade *thr1* kao kod modela sa jednim navojem.

**Napomena**: Izlaz može varirati zbog prebacivanja konteksta.

Prednosti Multithreading-a:

* Istovremeno izvršava više procesa
* Maksimizuje korištenje resursa CPU-a
* Dijeljenje vremena između više procesa.

## Tipovi niti

*Multithreading* je najkorisnija karakteristika C# koja omogućava istovremeno programiranje dva ili više dijelova programa za maksimizaciju korištenja CPU-a.

Svaki dio programa se zove *Thread*. Drugim riječima, niti su lagani procesi unutar procesa.

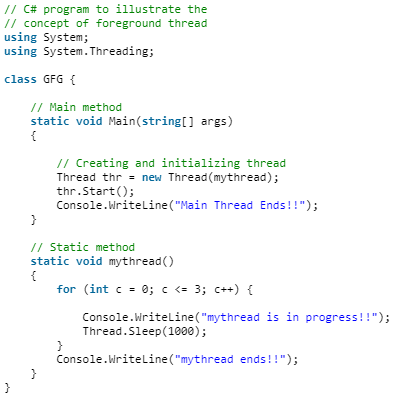
C # podržava dva tipa niti:

* Foreground Thread
* Background Thread

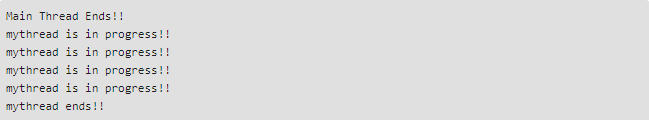
### Foreground Thread

Nit koja nastavlja da radi kako bi dovršila svoj rad, čak i ako glavni tok napusti svoj proces, ovaj tip niti poznat je kao prednji plan. Niti u prvom planu ne zanima da li je glavna nit živa ili ne, završava se samo kada završi dodijeljeni posao. Ili drugim riječima, životni vijek niti ne zavisi o glavnoj niti.

Primjer:



Ispis na konzoli:



U gornjem primjeru, *thr* *thread* se izvodi nakon završetka glavne niti. Dakle, životni vijek trajanja gazećeg sloja ne zavisi od životnog vijeka glavne niti.

*Thr* *thread* završava svoj proces tek kada dovrši svoj dodijeljeni zadatak.

### Background Thread

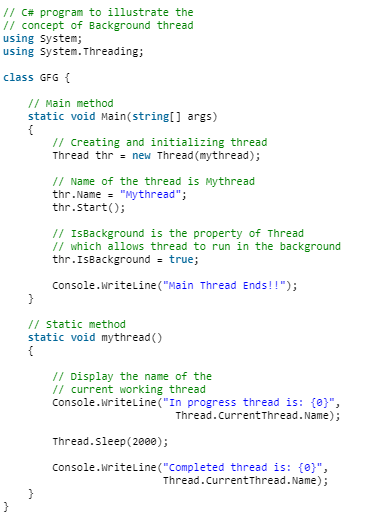
Nit koja napušta svoj proces kada *Main* metoda napusti svoj proces, ovi tipovi niti su poznati kao pozadinske niti ili drugim riječima, život pozadinske niti zavisi o životu glavne niti. Ako glavna nit završi svoj proces, onda pozadinska nit završava svoj proces.

**Napomena:**

Ako želite koristiti pozadinsku nit u vašem programu, tada postavite vrijednost svojstva *IsBackground* niti na true.

U narednom primjeru, svojstvo *IsBackground* klase *Thread* se koristi za postavljanje *thr* niti kao pozadinske niti tako što je vrijednost **IsBackground = true**. Ako postavite vrijednost **IsBackground = false**, tada se zadana nit ponaša kao prednji plan. Sada se proces *thr* niti završava kada se završi proces glavne niti.

Primjer:



Ispis na konzoli:



## Kreiranje niti

U C#, *multithreading* sistem je izgrađen na klasi *Thread*, koja inkapsulira izvršavanje niti. Ova klasa sadrži nekoliko metoda i svojstava koja pomažu u upravljanju i kreiranju niti i ova klasa je definisana u *System.Threading namespace*. *System.Threading namespace* pruža klase i interfejse koji se koriste u programiranju sa više niti.

Koraci za kreiranje niti u C # programu:

* Prije svega importujte *System.Threading* imenski prostor, on igra važnu ulogu u kreiranju niti u vašem programu jer ne morate svaki put pisati potpuno kvalifikovano ime klase.

using System;

using System.Threading;

* Sada kreirajte i inicijalizujte objekat niti u glavnoj metodi.

public static void main()

{

Thread thr=new Thread(job1);

}

Takođe možete koristiti *ThreadStart* konstruktor za inicijalizaciju nove instance.

public static void main()

{

Thread thr=new Thread(new ThreadStart(job1));

}

* Sada možete pozvati vaš objekat niti.

public static void main()

{

Thread thr=new Thread(job1);

thr.Start();

}

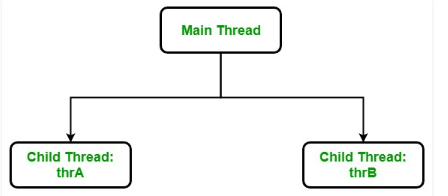
## Glavna nit

C # pruža ugrađenu podršku za višenitno programiranje. Program s više navoja sadrži dva ili više dijelova koji se mogu izvoditi istovremeno. Svaki dio takvog programa naziva se nit, a svaki thread definiše poseban put izvršenja.

Kada se program C# pokrene, jedna nit počinje odmah. Ovo se obično naziva glavna nit našeg programa.

To je nit ispod koje će biti kreirane druge „dijete" (eng. *child*) teme.

Često, to mora biti posljednja nit koja će završiti izvršavanje jer izvodi različite akcije isključivanja.



Slika 8: Glavna nit i njeni potomci

## Stanja i životni vijek niti

Nit u C# u bilo kojem trenutku postoji u bilo kojem od sljedećih stanja. Nit se u svakom trenutku nalazi samo u jednom od prikazanih stanja:

* Unstarted
* Runnable
* Running
* Not Runnable
* Dead

Životni ciklus niti:

* **Unstarted state** - Kada je kreirana instanca klase *Thread*, ona je u nestabilnom stanju, što znači da nit još nije počela da se izvodi kada je nit u ovom stanju. Drugim riječima, metoda *Start()* se ne zove.
* **Runnable State** - Nit koja je spremna za pokretanje premješta se u stanje izvršenja. U ovom stanju, nit može zapravo biti pokrenuta ili može biti spremna za pokretanje u bilo kojem trenutku vremena. To je odgovornost raspoređivača niti da daje temu, vrijeme za izvođenje. Drugim riječima, zove se metoda *Start ().*
* **Running State** - nit koja se izvodi. Drugim riječima, nit dobija procesor.
* **Not Runnable State** - nit koja nije izvršna
* **Dead State** - Kada nit dovrši svoj zadatak, nit ulazi u dead, završava, prekida stanje.

## Klasa *Thread*

U C # - u, *multithreading* sistem je izgrađen na klasi *Thread*, koja inkapsulira izvršavanje niti. Ova klasa sadrži nekoliko metoda i svojstava koja pomažu u upravljanju i kreiranju niti i ova klasa je definisana u *System.Threading* imenskom prostoru.

Karakteristike klase Thread:

* Klasa *Thread* se koristi za kreiranje niti.
* Pomoću *Thread* klase možete kreirati foreground i pozadinu.
* *Thread* *class* vam omogućava da postavite prioritet niti.
* Takođe vam daje trenutno stanje niti.
* On daje referencu trenutno izvršavane niti.
* To je zapečaćena klasa, tako da se ne može naslijediti.

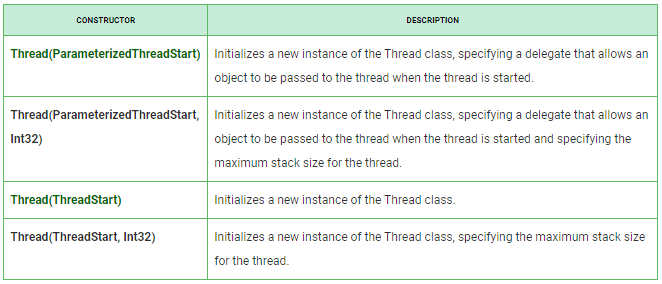


Tabela 2: Konstruktor klase Thread

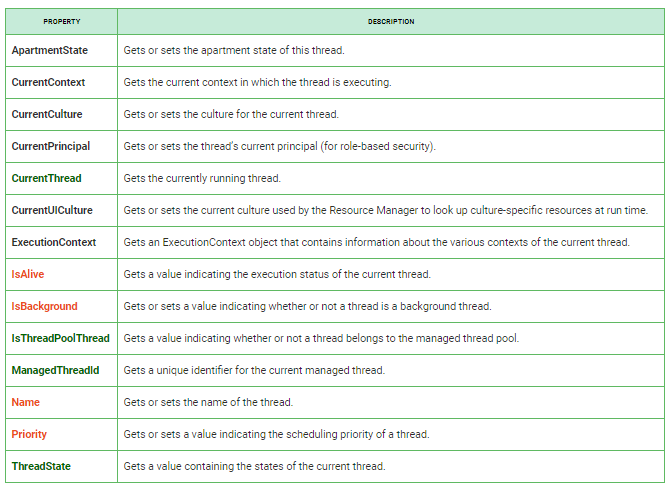


Tabela 3: Propertiji klase Thread

## Prekidanje niti

U C #, nit se može prekinuti pomoću *Abort()* metode. *Abort()* baca *ThreadAbortException* na nit u kojoj se poziva. Zbog ovog izuzetka nit je prekinuta. Postoje dvije metode u popisu preopterećenja metode *Thread.Abort*:

* *Abort()*
* *Abort(Object)*

*Abort()* - Ovaj metod podiže *ThreadAbortException* u niti na kojoj se poziva, da započne proces završavanja niti. Uopšteno, ovaj metod se koristi za okončanje niti.

Sintaksa:

public void Abort();

*Abort(Object)* - Ovaj metod podiže *ThreadAbortException* u niti na kojoj se poziva, da započne proces završavanja niti dok istovremeno daje informacije o izuzetku završetka niti. Uopšteno, ovaj metod se koristi za okončanje niti.

Sintaksa:

public void Abort(object information);

# Aplikacija

` Zadata aplikacija je izrađena korištenjem C# programskog jezika, te WPF framework-a kao GUI (engl. *Graphic User Interface* ). Osnovni cilj aplikacije jeste obrada statističkih podataka sportskih dešavanja, u našem konkretnom slučaju aplikacija prikazuje podatke utakmica odigranih u NBA ligi. Podaci su skladišteni na localhostu korištenjem XAMPP softvera za pokretanje MySQL servera.

U bazi podataka nalaze se sledeće tabele:

1. igrac

Sadrži sve relevantne podatke o svim igračima NBA lige.

1. statistika\_igraca

Sadrži sve relevantne podatke potrebne za vođenje osnovne statistike na utakmicama

(broj poena, broj skokova, broj faulova,broj blokada...).

1. tim

Sadrži sve relevantne podatke o svim timovima NBA lige.

1. utakmica

Spisak svih odigranih utakmica.

Statistički podaci uneseni u tabelu „statistika\_igraca“ uneseni su nasumično korištenjem klase *Random*.

SQL skripta baze podataka dostavljena je sa izvornim kodom kao prateća dokumentacija.

Funkcionalne cijeline aplikacije:

## Main Window

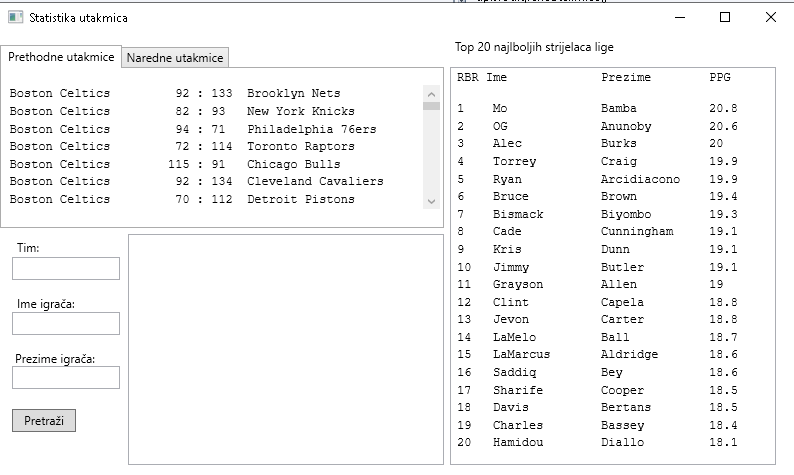
Main Window predstavlja prvu formu prikazanu korisniku nakon pokretanja aplikacije.

U gornjem dijelu prozora prikazane su odigrane utakmice sa rezultatima, kao i utakmice koje tek trebaju da se odigraju, u posebnim karticama.

U okviru dijela sa odigranim utakmicama korisniku je dostupan pregled detaljne statistike odabrane utakmice. Statistika se prikazuje u novom prozoru, a odabir se vrši klikom na utakmicu.

Na desnoj strani prozora prikazan je spisak dvadeset najboljih strijelaca lige, odnosno dvadeset igrača sa najvećim prosjekom poena po utakmici. Za izdvajanje i sortiranje ovih igrača korišteni su nizovi radi boljeg prikaza rada višenitnog izvršenja.

Donji dio prozora zauzima forma za pretragu igrača, dugme za pokretanje pretrage i prostor za prikaz pronađenih rezultata.

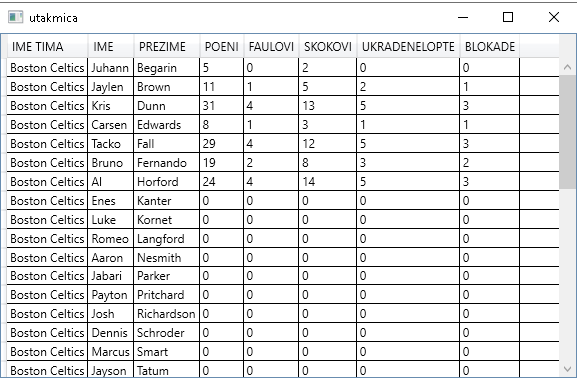


Slika 9: Izgled osnovnog prozora

## Prozor „utakmica“

Prozor utakmica korisniku se prikazuje nakon selekcije odigrane utakmice. U ovom

prozoru dostupni su podaci o svim igračima oba tima koji su učestvovali u utakmici, kao i svi statistički podaci dostupni u bazi podataka.

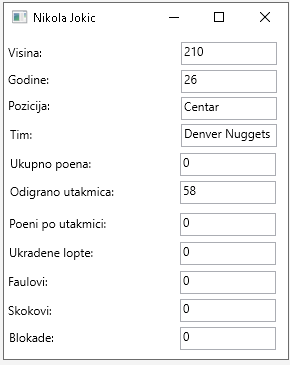


Slika 10: Prikaz detalja utakmice

## Detalji o igraču

Za prikaz podataka o igraču potrebno je da korisnik pronađe igrača u polju za pretragu i

odabere ga klikom miša. Nakon toga se otvara novi prozor u kojem su prikazani podaci o igraču ( visina, godine, pozicija... ) i njegovi rezultati u proteklim utakmicama.



Slika 11: Prikaz detalja o igraču

# Testiranje i paralelizam

Vrijeme izvršenja svake od navedenih cjelina je zasebno izmjereno radi utvrđivanja koji dijelovi koda zahtjevaju više vremena za obradu, kako bismo te dijelove mogli pokušati izvršiti paralelno radi smanjenja utrošenog vremena u konačnici. Mjerenja vremena su izvršena pri sekvencijalnom i paralelnom izvršenju programa. Rezultati su tabelarno prikazani na bazi deset mjerenja, razvrstani po dijelovima aplikacije čija izvršenja su mjerena. Vrijednosti u tabelama su izražene u milisekundama. Takođe mjerenja su izvršena na više različitih procesora. Na sledećim graficima prikazana je razlika između paralelnog i sekvencijalnog pokretanja aplikacije. Na vertikalnoj osi nalaze se vrijednosti u milisekundama, dok je na horizontalnoj naveden redni broj pokretanja. Za svaki procesor posebno su prikazani grafici.

Nakon izmjerenih vremena izvršenja, paralelno izvršenje je dodato na mjestu sortiranja niza najboljih strijelaca lige i u dijelu koda koji je zadužen za prikaz odigranih utakmica.

## Intel Pentium P6000 1.87GHz

Broj fizičkih jezgri: 2

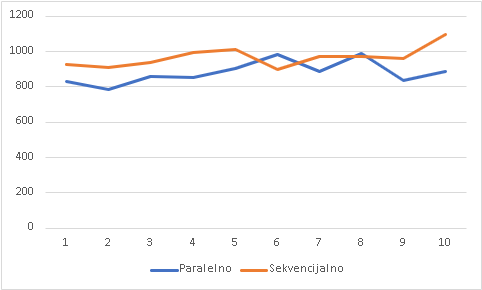
Broj logičkih jezgri: 2

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Sekvencijalno izvršenje | | | | | | | | | | | |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | prosjek |
| Otrvaranje Main Window-a | 1096 | 926 | 911 | 939 | 993 | 1009 | 898 | 971 | 969 | 963 | 967,5 |
| Pretraga igrača i timova | 4 | 2 | 12 | 2 | 2 | 2 | 2 | 4 | 2 | 2 | 3,4 |
| Prikaz detalja o igraču | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0,0 |
| Prikaz statistike | 33 | 27 | 53 | 27 | 26 | 32 | 29 | 36 | 93 | 30 | 38,6 |
| Vrijeme sortiranja niza igrač a | 165 | 162 | 174 | 175 | 196 | 217 | 168 | 185 | 177 | 181 | 180,0 |

Tabela 4: Sekvencijalno izvršenje na Intel Pentium P6000 1.87GHz

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Paralelno izvršenje | | | | | | | | | | | |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | prosjek |
| Otrvaranje Main Window-a | 832 | 785 | 861 | 854 | 903 | 985 | 884 | 987 | 835 | 886 | |  | | --- | | 881,2 | |
| Pretraga igrača i timova | 4 | 2 | 12 | 2 | 2 | 2 | 2 | 4 | 2 | 2 | 3,4 |
| Prikaz detalja o igraču | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0,0 |
| Prikaz statistike | 33 | 27 | 53 | 27 | 26 | 32 | 29 | 36 | 93 | 30 | 38,6 |
| Vrijeme sortiranja niza igrač a | 43 | 76 | 46 | 44 | 69 | 59 | 63 | 55 | 47 | 49 | 55,1 |

Tabela 5: Paralelno izvršenje na Intel Pentium P6000 1.87GHz



Grafik 1: Razlika između sekvencijalnog i paralelnog izvršenja na Intel Pentium P6000 1.87GHz

## Intel Core i3 2310M 2.1GHz

Broj fizičkih jezgri: 2

Broj logičkih jezgri: 4

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Sekvencijalno izvršenje | | | | | | | | | | | |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | prosjek |
| Otrvaranje Main Window-a | 783 | 679 | 697 | 722 | 652 | 707 | 670 | 678 | 706 | 676 | 697,0 |
| Pretraga igrača i timova | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1,0 |
| Prikaz detalja o igraču | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0,0 |
| Prikaz statistike | 22 | 16 | 17 | 16 | 17 | 17 | 16 | 17 | 16 | 16 | 17,0 |
| Vrijeme sortiranja niza igrač a | 148 | 158 | 159 | 155 | 150 | 150 | 155 | 153 | 170 | 150 | 154.8 |

Tabela 6: Sekvencijalno izvršenje na Intel Core i3 2310M 2.1GHz

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Paralelno izvršenje | | | | | | | | | | | |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | prosjek |
| Otrvaranje Main Window-a | 702 | 708 | 704 | 729 | 697 | 708 | 700 | 706 | 708 | 700 | 706.2 |
| Pretraga igrača i timova | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1,0 |
| Prikaz detalja o igraču | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0,0 |
| Prikaz statistike | 22 | 16 | 17 | 16 | 17 | 17 | 16 | 17 | 16 | 16 | 17,0 |
| Vrijeme sortiranja niza igrač a | 127 | 118 | 126 | 118 | 122 | 131 | 124 | 133 | 127 | 137 | 126.3 |

Tabela 7:Paralelno izvršenje na Intel Core i3 2310M 2.1GHz

Grafik 2: Razlika između sekvencijalnog i paralelnog izvršenja na Intel Core i3 2310M 2.1GHz

## Intel Celeron N3060 1.6Ghz

Broj fizičkih jezgri: 2

Broj logičkih jezgri: 2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Sekvencijalno izvršenje | | | | | | | | | | | | | |
|  | 1 | 2 | | 3 | 4 | 5 | 6 | 7 | 8 | 9 | | 10 | prosjek |
| Otrvaranje Main Window-a | 1610 | | 1967 | 1787 | 1776 | 1782 | 1870 | 1798 | 1813 | | 1790 | 1822 | 1801.5 |
| Pretraga igrača i timova | 3 | 3 | | 3 | 3 | 4 | 4 | 4 | 3 | 4 | | 4 | 3.5 |
| Prikaz detalja o igraču | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 0 | 0,0 |
| Prikaz statistike | 30 | 34 | | 35 | 29 | 28 | 31 | 29 | 30 | 30 | | 30 | 30.6 |
| Vrijeme sortiranja niza igrač a | 562 | 558 | | 565 | 558 | 609 | 557 | 590 | 563 | 555 | | 561 | 567.8 |

Tabela 8: Sekvencijalno izvršenje na Intel Celeron N3060 1.6GHz

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Paralelno izvršenje | | | | | | | | | | | | | |
|  | 1 | 2 | | 3 | 4 | 5 | 6 | 7 | 8 | 9 | | 10 | prosjek |
| Otrvaranje Main Window-a | 1697 | | 1678 | 1539 | 1541 | 1449 | 1475 | 1503 | 1478 | | 1521 | 1468 | 1534.9 |
| Pretraga igrača i timova | 3 | 3 | | 3 | 3 | 4 | 4 | 4 | 3 | 4 | | 4 | 3.5 |
| Prikaz detalja o igraču | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 0 | 0,0 |
| Prikaz statistike | 30 | 34 | | 35 | 29 | 28 | 31 | 29 | 30 | 30 | | 30 | 30.6 |
| Vrijeme sortiranja niza igrač a | 260 | 176 | | 194 | 187 | 181 | 198 | 195 | 183 | 175 | | 185 | 193.4 |

Tabela 9:Paralelno izvršenje na Intel Celeron N3060 1.6GHz

Grafik 3: Razlika između sekvencijalnog i paralelnog izvršenja na Intel Celeron N3060 1.6GHz

# Zaključak

Iz navedenih tabela i grafika uočljive su razlike u vremenu izvršenja koda udijelovima koji su parealelizovani. Promjene koje se dešavaju uslijed pokretanja višenitnog izvršenja nemaju isti efekat na svaki procesor. Dakle poboljšanje/pogoršanje nekog vremena izvršenja nije konstantno za sve mašine. Paralelno izvršenje je uvedeno na dijelovima koda za koje je utvrđeno da oduzimaju najviše vremena. U ovoj aplikaciji može se uočiti da paralelizacijom dijela koda „upis odigrane utakmice“ ne daje željena poboljšanja, naprotiv u prosjeku, paralelno izvršenje zahtjeva više vremena od sekvencijalnog. Ovo je razumljivo iz razloga sto ne možemo radnju dodavanja sadržaja na neku UI komponentu (u našem slučaju list box) vršiti simultano. Dok jedna nit pristupa nekoj komponenti, druga nit je blokirana i čeka priliku za pristup komponenti, bez obzira što su podaci ranije spremni za upis. Drugi dio koda prepoznat kao pogodan za izvršenje paralelizacije jeste sortiranje nizova za potrebe prikaza dvadeset najboljih igrača lige. Na ovome mjestu niz koji treba da bude sortiran je podijeljen na dva dijela. Ovime je postignuto simultano sortitanje dvije polovine niza pa smo samim tim uočili i poboljšanja u vremenu izvršenja toga dijela koda, ali i cijele aplikacije. U konačnici zaključak je da paralelno izvršenje ne predstavlja uvijek dobru odluku, potrebno je prethodno utvrditi da li postoji mogućnost paralelnog izvršenja i u kojoj mjeri ti koraci donose uštedu vremena.

# Literatura

1. *A Tour of the C# language.* (2021, 1 28).
2. *Albahari, J.* (2006-2014). *Threading in C#.*
3. *System.Threading Namespace.* (n.d.).
4. *Thread in Operating System.* (n.d.).
5. *Using threads and threading.* (2018, 8 8).
6. *„Objektno orjentisano programiranjeu jeziku C#“, Dragan Milićev, Beograd 1996.*
7. *„Tehnike vizuelnog programiranja u C#“, Zoran Ćirović, Ivan Dunđerski, Viša elektrotehnička škola, Beograd, 2006.*
8. *„Programski jezik C#“, Mario Bošnjak, Univerzitet u Zagrebu*

# Linkovi

<https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>

<http://www.albahari.com/threading/>

https://docs.microsoft.com/en-us/dotnet/api/system.threading?view=net-5.0/

https://www.geeksforgeeks.org/thread-in-operating-system/

<https://docs.microsoft.com/en-us/dotnet/standard/threading/using-threads-and-threading>

# Popis slika

[Slika 1:Tranzicija od C do C# 5](file:///C:\Users\andja\Downloads\nova-verzija.docx#_Toc89978022)

[Slika 2: Izgled startnog prozora Microsoft Visual C# 2010 Express 15](#_Toc89978023)

[Slika 3: Prazna forma 18](#_Toc89978024)

[Slika 4: Razvojno okruženje za Windows aplikacije 18](#_Toc89978025)

[Slika 5: Toolbox sa osnovnim WPF kontrolama 19](#_Toc89978026)

[Slika 6: Izvršavanje programa koji koristi samo glavnu nit 21](#_Toc89978027)

[Slika 7: Izvršavanje programa koji koristi višenitnost 23](#_Toc89978028)

[Slika 8: Glavna nit i njeni potomci 29](#_Toc89978029)

[Slika 9: Izgled osnovnog prozora 33](#_Toc89978030)

[Slika 10: Prikaz detalja utakmice 34](#_Toc89978031)

[Slika 11: Prikaz detalja o igraču 35](#_Toc89978032)

# Popis grafika

[Grafik 1: Razlika između sekvencijalnog i paralelnog izvršenja na Intel Pentium P6000 1.87GHz 37](#_Toc89978079)

[Grafik 2: Razlika između sekvencijalnog i paralelnog izvršenja na Intel Core i3 2310M 2.1GHz 38](#_Toc89978080)

[Grafik 3: Razlika između sekvencijalnog i paralelnog izvršenja na Intel Celeron N3060 1.6GHz **Error! Bookmark not defined.**](#_Toc89978081)

# Popis tabela

[Tabela 1: Ključne riječi u C# 14](#_Toc89978111)

[Tabela 2: Konstruktor klase Thread 30](#_Toc89978112)

[Tabela 3: Propertiji klase Thread 31](#_Toc89978113)

[Tabela 4: Sekvencijalno izvršenje na Intel Pentium P6000 1.87GHz 36](#_Toc89978114)

[Tabela 5: Paralelno izvršenje na Intel Pentium P6000 1.87GHz 37](#_Toc89978115)

[Tabela 6: Sekvencijalno izvršenje na Intel Core i3 2310M 2.1GHz 38](#_Toc89978116)

[Tabela 7:Paralelno izvršenje na Intel Core i3 2310M 2.1GHz 38](#_Toc89978117)

[Tabela 8: Sekvencijalno izvršenje na Intel Celeron N3060 1.6GHz 39](#_Toc89978118)

[Tabela 9:Paralelno izvršenje na Intel Celeron N3060 1.6GHz 40](#_Toc89978119)