C# 03. Generičke metode i klase, izuzeci, ulaz-izlaz, rad sa fajl sistemom

Prof. dr Suzana Stojković

Mr Martin Jovanović

Dipl. inž. Ivica Marković

Dipl. inž. Teodora Đorđević

Sadržaj

- Generičke metode
- Generičke klase
- Klase List i Dictionary
- *Nullable* tipovi
- Izuzeci
- Ulaz/Izlaz
- Interfejs IDisposable i using ključna reč
- Rad sa fajl sistemom

Generičke metode

- Moguće je implementirati metodu koja će prihvatati proizvoljni tip argumenata
  - Implementacija:
```
public void Zameni<T>(ref T a, ref T b) {
    T pom = a;
    a = b;
    b = pom;
}
```
  - Poziv metode:
```
…
int x = 1;
int y = 2;
Zameni<int>(ref x, ref y);
…
```

Generičke metode – where ograničenja

- Moguće je nametnuti neka ograničenja koja generički parametar tipa T mora da ispuni
- Sintaksa za navođenje ograničenja je where <naziv_tipa> : <ograničenja>
  - void Sortiraj<T>(T[] niz) where T : IComparable // tip T mora da implementira interfejs IComparable
  - void Sortiraj<T>(T[] niz) where T : KompleksniBroj // T mora da bude tipa kompleksni broj ili nekog njegovog izvedenog tipa
  - void Sortiraj<T>(T[] niz) where T : class // tip T mora da bude klasa
  - void Sortiraj<T>(T[] niz) where T : struct // tip T mora da bude struktura
  - void Sortiraj<T>(T[] niz) where T : new() // tip T mora da ima konstruktor bez argumenata

Generičke metode – where ograničenja

- Metoda može da ima i više od jednog generičkog parametra
  - void MojaMetoda<T, V> (T arg1, V arg2) { … }
- Moguće je kombinovati više ograničenja za isti tip
  - void Sortiraj<T>(T[] niz) where T : class, IComparable, new()
- Moguće je da svaki od generičkih parametara ima posebno ograničenje
  - void MojaMetoda<T, V> (T arg1, V arg2) where T : class where U : struct{ … }
- Primer: GenerickeMetode

Generičke klase i interfejsi

- Ako parametar tipa T definišemo na nivou cele klase (interfejsa) dobićemo generičku klasu (interfejs)
- Onda automatski bilo koja metoda i property klase (interfejsa) može biti generička
- Kod deklaracije generičkih klasa (interfejsa) moguće je postaviti ista ograničenja za tip T kao i u slučaju generičkih metoda (ključna reč where)

- Primer: GenerickiStek

Generički mehanizam C# vs templejti C++

- Generički mehanizam slično radi kao i templejtske klase u C++u
- Neke od razlika su sledeće (uglavnom C# nameće dodatna ograničenja)
  - U C++u zamena parametra T stvarnim tipom se vrši u toku kompajliranja koda, a u C#u u toku izvršenja koda (.NET runtime ubacuje u odgovarajuća mesta u MSIL kodu stvarni tip)
  - Nije moguće pozvati aritmetičke operatore nad operandima tipa T u C#u (zbog toga što se zamena stvarnim tipom vrši tek u toku izvršenja koda pa kompajler ne može da proveri da li su za taj stvarni tip definisani aritmetički operatori)
  - Nije moguće postaviti generički parametar koji ne predstavlja tip (u C++u radi *template C<int i>*)
  - …

Generički mehanizam C# vs Java

- Generički mehanizam u C#-u sličniji je generičkom mehanizmu u Javi nego templejtskim klasama u C++u.
- Bitna razlika u odnosu na Javu je prenošenje informacija o generičkim tipovima u međukod.
- Java:
  - Informacija o generičkim tipovima se koristi samo tokom kompajliranja programa da korisnik ne bi morao na više mesta da radi kastovanje;
  - Interno se svaki generički parametar tipa T prevodi u tip Object (npr. generička klasa ArrayList<T> interno čuva niz elemenata tipa Object).
- C#:
  - Informacija o generičkim tipovima se prenosi u međukod (MSIL) – i u međukodu se prepozaje tip T (npr. generička klasa List<T> interno čuva niz elemenata tipa T).

Interfejs System.IComparable

- Ima samo jednu metodu int CompareTo(object obj)
- Ova metoda poredi tekući objekat this sa prosleđenim objektom obj i vraća vrednost koja određuje redosled ta dva objekta prilikom sortiranja:
  - Ako this prethodi objektu obj u sortiranom redosledu vraća vrednost manju od 0;
  - Ako this i obj mogu da stoje na istoj poziciji u sortiranom redosledu vraća vrednost 0;
  - Ako this sledi objekat obj u sortiranom redosledu vraća vrednost veću od 0.

Interfejs System.IComparable

- Primer implementacije metode int CompareTo(object obj)
  - public int CompareTo(object obj) {
  - KompleksniBroj drugi = obj as KompleksniBroj;
  - if (drugi == null) throw new ArgumentException(
  - "Argument mora biti kompleksni broj i različit od null");
  - int rezultat;
  - if (this.Moduo == drugi.Moduo) rezultat = 0;
  - else if (this.Moduo > drugi.Moduo) rezultat = 1;
  - else rezultat = -1;
  - return rezultat;
  - }
- Neoptimalna je provera tipa i kastovanje kod ovakve implementacije (linija koda označena crveno)

Interfejs System.IComparable<T>

- Rešenje za prethodni problem je generički interfejs IComparable<T> gde smo sigurni da je objekat other istog tipa kao i this, samo proveravamo da nije null
  - public int CompareTo(T other) {
  - if (other == null) throw new ArgumentException(
  - "Argument mora biti različit od null");
  - int rezultat;
  - if (this.Moduo == other.Moduo) rezultat = 0;
  - else if (this.Moduo > other.Moduo) rezultat = 1;
  - else rezultat = -1;
  - return rezultat;
  - }

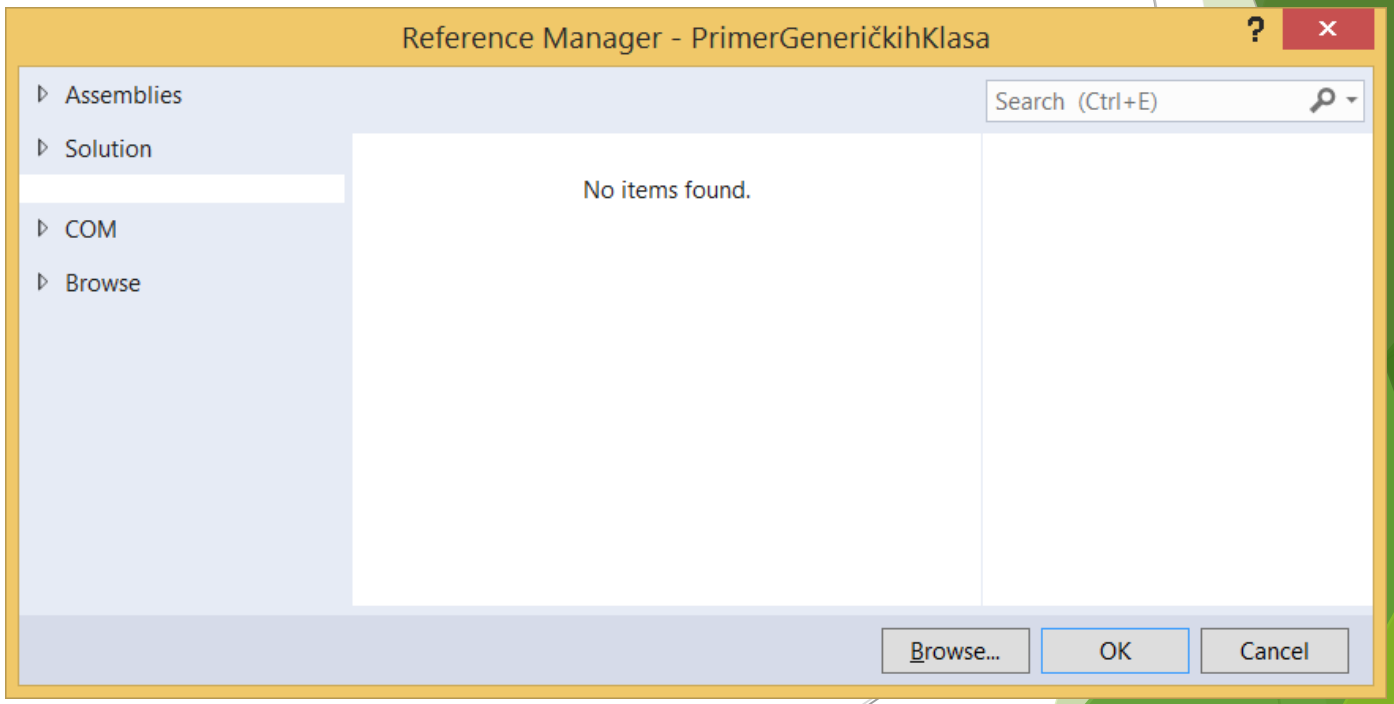
Generičke klase iz .NET Framework-a

- System.Collections.Generic.List<T>
- System.Collections.Generic.Dictionary<TKey,TValue>

- Primer: projekat PrimerGeneričkihKlasa
- Ovaj projekat koristi klase iz projekta StudentiIOcene
- Da bi jedan projekat koristio podatke iz drugog projekta potrebno je da referencira taj drugi projekat

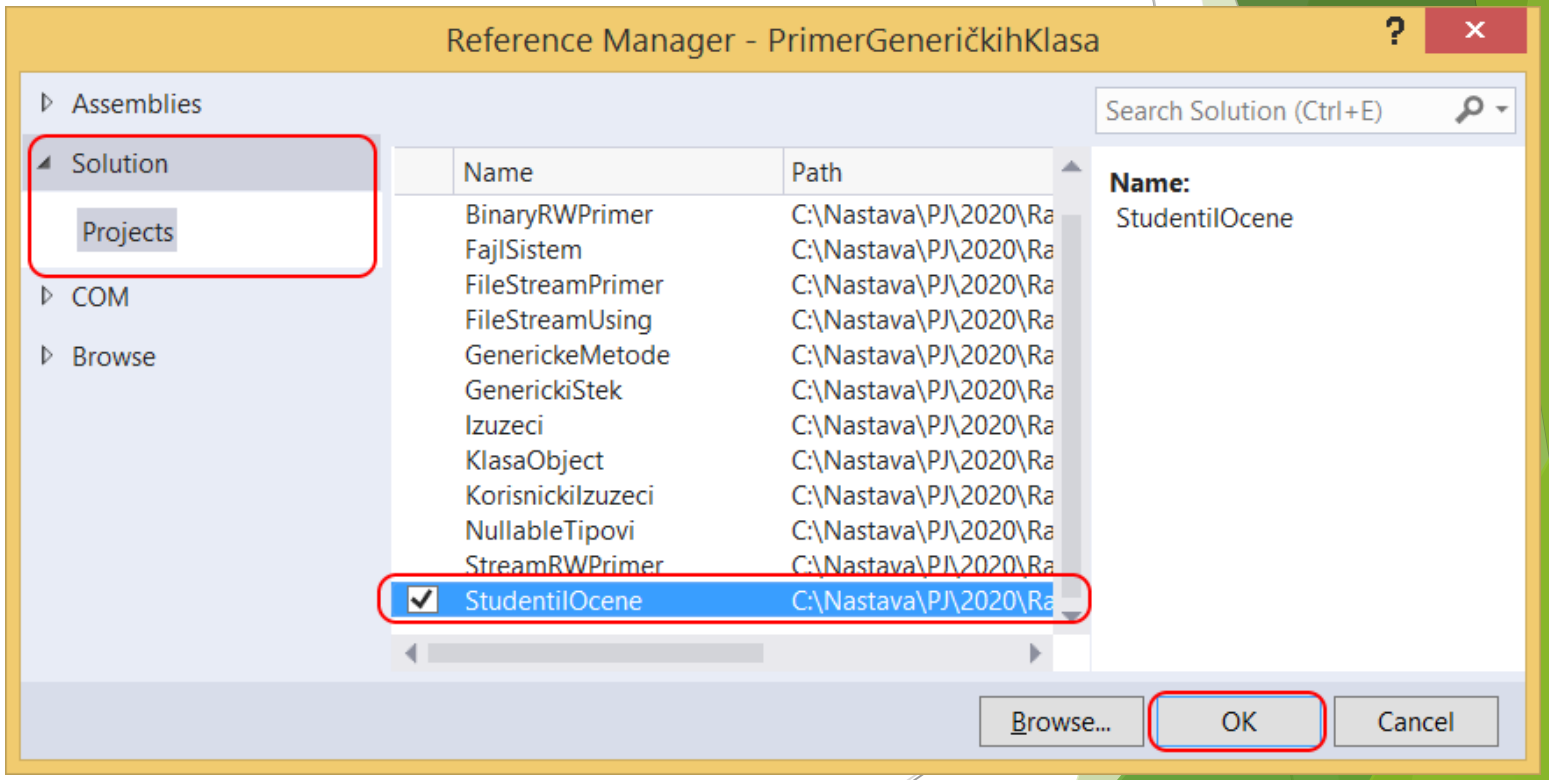
Međusobno referenciranje projekata

- Na projektu koji treba da referencira drugi projekat izvršimo desni klik pa iz kontekstnog menija biramo opciju Add… → Reference...
- Dobijamo dijalog kao na slici



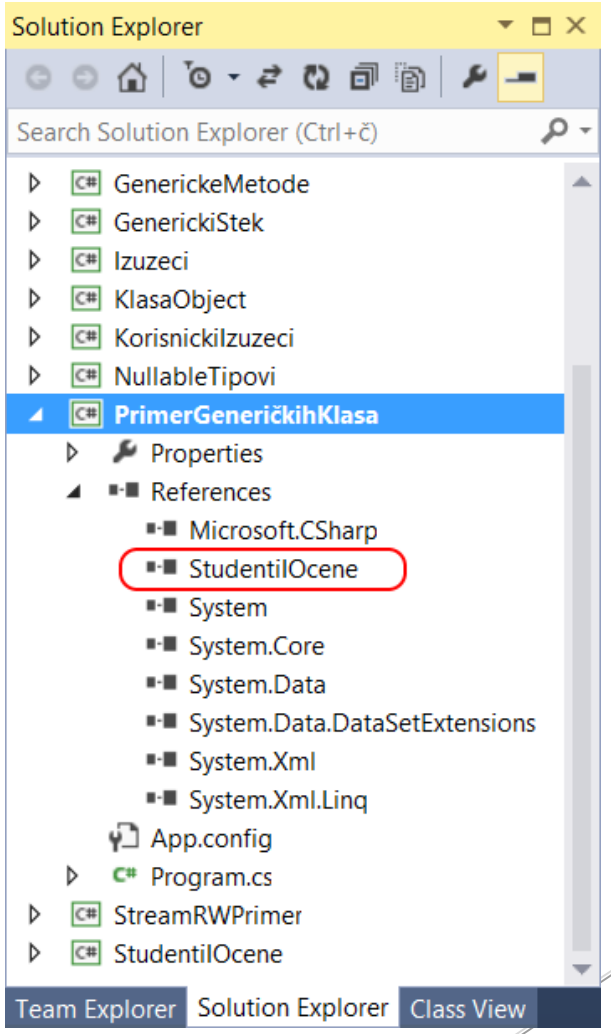
Međusobno referenciranje projekata

- U stablu sa leve strane biramo Solution… → Projects...
- Zatim čekiramo projekat “StudentiIOcene”
- Na kraju potvrđujemo izbor klikom na dugme OK



Međusobno referenciranje projekata

- Na kraju kao potvrdu da smo korektno dodali referencu u Solution Explorer pogledu među referenciranim sistemskim bibliotekama za naš projekat vidimo i projekat “StudentiIOcene”
- Obično su projekti koji se referenciraju tipa Class Library i njihov generisani assembly je .DLL fajl
- Ovde je referencirani projekat “StudentiIOcene” konzolna aplikacija (što ne utiče bitno na referenciranje)



Klasa List<T>

- Iako se zove List nije u pitanju lančana lista.
- Implementirana je kao niz elemenata tipa T. Kapacitet liste je promenljiv.
- Npr. neka lista ima inicijalni kapacitet od 4 elementa i neka su sva 4 mesta trenutno popunjena. Pokušavamo da dodamo naredni element pozivom metode Add.
- Interno se rezerviše novi niz od 8 elemenata (2 puta više od trenutnog kapaciteta) i kopiraju se postojeća 4 elementa na pozicije 0-3 u novom nizu. Na poziciju sa indeksom 4 se smešta novi element. Sve ovo se dešava bez kontorle programera, tj. mi samo treba da pozovemo metodu Add.
- Garbage Collector može nekada da oslobodi stari niz kada se ukaže potreba za tim
- Niz može da naraste do maksimalno 2G elemenata

Klasa List<T>

- Na ovaj način pristup elementu tj. čitanje elementa sa zadate pozicije u listi je jednako efikasno kao čitanje iz običnog niza
- Dodavanje može da bude skupo ako se radi povećanje kapaciteta liste
- Konstruktori:
  - public List() – Podrazumevani konstruktor koji postavlja kapacitet na 4 elementa.
  - public List(int capacity) – Konstruktor koji postavlja kapacitet na zadatu vrednost.
  - public List(IEnumerable<T> collection) – Konstruktor koji smešta sve elemente zadate kolekcije u listu. IEnumerable je interfejs iz C# koji implementiraju sve kolekcije podataka (čak i obični nizovi).

Klasa List<T>

- Property-ji:
  - public int Capacity { get; set; } – Postavlja ili vraća kapacitet liste.
  - public int Count { get; } –Vraća trenutni broj elemenata u listi.
  - public T this[int index] { get; set; } – Indekser koji omogućuje pristup elementima u listi kao da se radi o običnom nizu. Ne koristiti indekser za dodavanje novih elemenata u listu.
    - List<int> l = new List();

      l.Add(2); // radi ok

      l.Add(4); // radi ok

      int p = l[2]; // generiše izuzetak jer na toj poziciji nema elemenata

      l[2] = 8; // generiše izuzetak, dodavanje mora da se izvrši Add metodom

      l[1] = 5; // radi jer se samo menja vrednost postojećeg elementa

Klasa List<T>

- Metode:
  - public void Add(T item) – Dodaje novi element na kraj liste.
  - public void AddRange(IEnumerable<T> collection) – Dodaje celu kolekciju novih elemenata na kraj liste.
  - public void Insert(int index, T item) – Umeće element item na poziciju index.
  - public void InsertRange(int index, IEnumerable<T> collection) – Umeće celu kolekciju elemenata počevši od pozicije index.
  - public void RemoveAt(int index) – Izbacuje iz liste element sa pozicije index.
  - public void RemoveRange(int index, int count) – Izbacuje iz liste kolekciju elemenata dužine count počevši od pozicije index.

Klasa List<T>

- Metode:
  - public void Clear() – Briše sve elemente iz liste.
  - public bool Contains(T item) – Vraća true ako lista sadrži element item.
  - public int IndexOf(T item) – Vraća indeks prvog pojavljivanja elementa item u listi, ako ne postoji vraća se -1.
  - public bool Remove(T item) – Uklanja navedeni element iz liste. Vraća true ako je uklanjanje uspelo, false ako element nije pronađen.

Klasa Dictionary<TKey, TValue>

- Implementiran je kao heš tablica u kojoj je ključ tipa TKey, a vrednost koja se čuva tipa TValue (čuva se par ključ-vrednost u obliku strukture public struct KeyValuePair<TKey, TValue>)
- TKey i TValue mogu da budu bilo koja klasa ili struktura.
- Svaka klasa ima metodu nasleđenu iz klase Object public virtual int GetHashCode() koju može da preklopi i tu metodu koristi klasa Dictionary za heširanje.
- Ako preklopimo Equals metodu u nekoj klasi obavezno moramo da preklopimo i GetHashCode jer ih Dictionary koristi u paru.
- Za svaka dva objekta o1 i o2 iste klase T ako o1.Equals(o2) vraća true mora da važi o1.GetHashCode() == o2.GetHashCode(). U suprotnom se javlja problem sa pronalaženjem objekta u heš tablici.

Klasa Dictionary<TKey, TValue>

- Konstruktori:
  - public Dictionary() – Podrazumevani konstruktor.
  - public Dictionary(int capacity) – Konstruktor koji postavlja kapacitet na zadatu vrednost.
- Property-ji:
  - public int Count { get; } – Vraća trenutni broj parova ključ-vrednost u Dictionary-ju.
  - public TValue this[TKey key] { get; set; } – Indekser koji vraća ili postavlja vrednost TValue koja je uparena sa ključem key. Getter baca izuzetak ako pokušamo da uzmemo vrednost za ključ key koji ne postoji u Dictinary-ju. Ako ne postoji ključ koji postavljamo koristeći setter on se dodaje sa zadatom vrednošću. Ako ključ postoji setter menja postojeću vrednost za taj ključ.

Klasa Dictionary<TKey, TValue>

- Property-ji:
  - public Dictionary<TKey, TValue>.KeyCollection Keys { get; } – Vraća kolekciju svih ključeva u Dictionary-ju.
  - public Dictionary<TKey, TValue>.ValueCollection Values { get; } – Vraća kolekciju svih vrednosti u Dictionary-ju.
- Metode:
  - public void Add(TKey key, TValue value) – Dodaje novi par ključ-vrednost u Dictionary. Baca izuzetak ako je key null ili key već postoji u Dictionary-ju.
  - public bool TryGetValue(TKey key, out TValue value) – Pokušava da iz Dictionary-ja nađe vrednost za prosleđeni ključ key. Ako je ključ pronađen vraća true i postavlja out promenljivu value. Ako ključ nije pronađen vraća false i value ima vrednost null.

Klasa Dictionary<TKey, TValue>

- Metode:
  - public void Clear() – Briše sve parove ključ-vrednost iz Dictionary-ja.
  - public bool ContainsKey(TKey key) – Vraća true ako rečnik sadrži ključ key.
  - public bool ContainsValue(TValue value) – Vraća true ako rečnik sadrži vrednost value.
  - public bool Remove(TKey key) – Briše par sa zadatim ključem iz Dictionary-ja.
- Primer: projekat Biblioteka

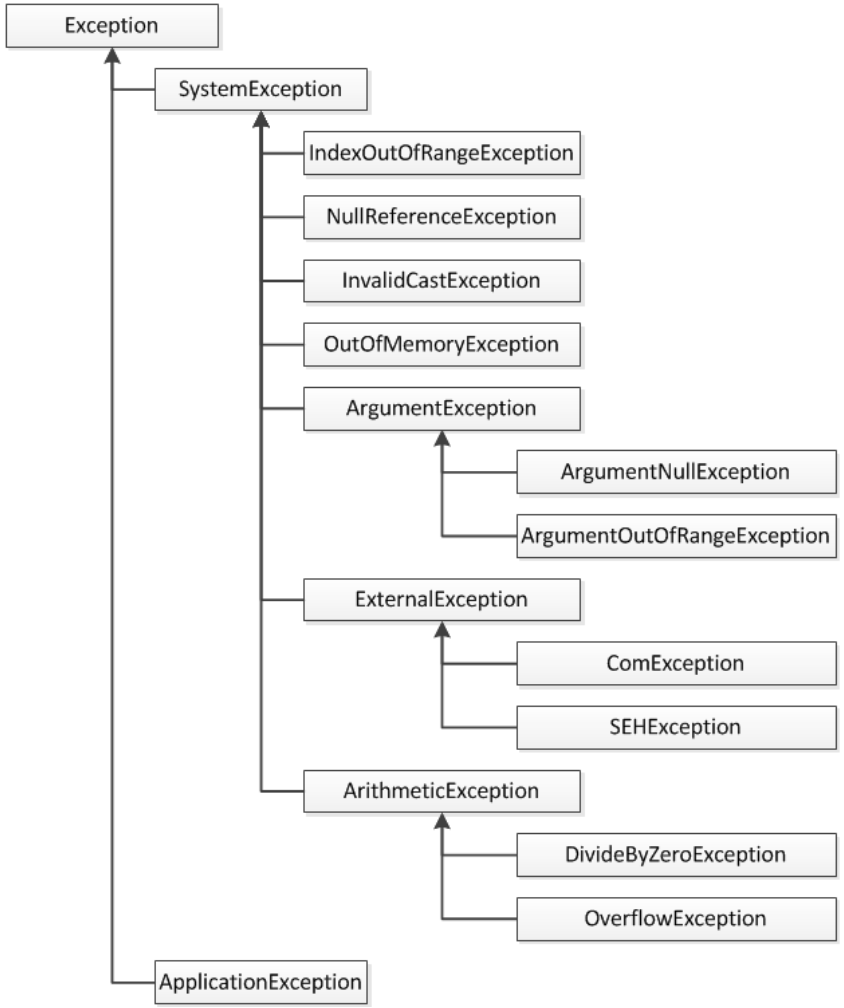
Nullable tipovi

- Promenljive referentnih tipova ukoliko im nije dodeljena vrednost imaju vrednost null
- Promenljive vrednosnih tipova obavezno imaju dodeljenu neku vrednost
- Da bismo mogli da označimo to dodatno stanje kad promenljivoj vrednosnog tipa nije dodeljena vrednost koriste se *nullable* tipovi
- Deklarišu se tako što se iza naziva vrednosnog tipa stavlja znak ?
  - Primer deklaracije nullable tipa: int? a;
  - Kompajler ovo interno prevodi u: System.Nullable<int> a; gde je System.Nullable<T> generička klasa
  - Klasa System.Nullable<T> ima dva važna property-ja bool HasValue i T Value
- Primer: projekat NullableTipovi

Izuzeci

- Upravljačka struktura je ista kao u Javi: try-catch-finally ili try-catch
- Ako sami želimo da generišemo izuzetak ključna reč je isto kao u Javi throw
- Osnovna klasa iz koje moraju biti izvedene klase za predstavljanje izuzetaka je System.Exception
- Za razliku od Jave metode mogu imati blokove koda koji potencijalno generišu izuzetke van try bloka, a da pritom u deklaraciji metode ne stoji ključna reč throws (u C#u ne postoji throws ili neka ekvivalentna ključna reč)

Izuzeci – hijerarhija klasa



Izuzeci - primeri

- Primer: projekat Izuzeci
- Primer: projekat KorisnickiIzuzeci

Ulaz/Izlaz – klasa System.IO.Stream

- Apstraktna osnovna klasa koja pruža pogled na tokove podataka kao na niz bajtova i podržava upis i čitanje bajtova
- Sve klase za rad sa tokovima podataka su izvedene iz klase Stream
- Tokovi podržavaju tri osnovne operacije:
  - Read – čitanje podataka iz toka i upis u neku strukturu podataka (npr. niz bajtova)
  - Write – upis podataka iz neke strukture u tok podataka
  - Seek – ispitivanje i promenu tekuće pozicije u toku podataka
- Zavisno od vrste toka tj. prirode podataka u toku klasa izvedena iz klase Stream može da podržava samo neke od ovih funkcionalnosti
- Npr. klasa PipeStream ne podržava seek operaciju
- Klasa Stream ima property-je CanRead, CanWrite i CanSeek za ispitivanje koje operacije su podržane
- Metoda Flush prazni sve interne bafere jednog toka
- Metoda Close oslobađa sve resurse koje je tok zauzeo

Ulaz/Izlaz – klasa System.IO.Stream

- Evo nekih najčešće korišćenih izvedenih klasa:
- **FileStream**
- IsolatedStorageFileStream
- MemoryStream
- BufferedStream
- NetworkStream
- PipeStream
- CryptoStream

Ulaz/Izlaz – klasa System.IO.FileStream

- Izvedena iz klase System.IO.Stream za rad sa fajlovima
- Konstruktor FileStream(string path, FileMode mode)
  - path – relativna ili apsolutna putanja do fajla kome se pristupa
  - mode – enumeracija FileMode sa jednom od vrednosti Append, Create, CreateNew, Open, OpenOrCreate, Truncate
- Konstruktor FileStream(string path, FileMode mode, FileAccess access)
  - access – enumeracija FileAccess sa jednom od vrednosti Read, ReadWrite, Write

Ulaz/Izlaz – enum System.IO.FileMode

- Append – otvara fajl ako postoji i pozicionira se na kraj fajla, ako ne postoji kreira novi fajl
- Create – kreira novi fajl, ako fajl već postoji novi fajl se prepisuje preko njega
- CreateNew – kreira novi fajl, ako fajl već postoji baca se izuzetak IOException
- Open – otvara se postojeći fajl, ako ne postoji baca se izuzetak System.IO.FileNotFoundException
- OpenOrCreate – otvara se fajl ako postoji, ako ne postoji kreira se novi fajl
- Truncate – otvara se postojeći fajl i njegov sadržaj se briše tako da mu je veličina 0 bajtova

Ulaz/Izlaz – klasa System.IO.FileStream

- Upis void WriteByte(byte value)
- Upis void Write(byte[] array, int offset,int count)
  - offset – indeks u nizu bajtova odakle počinje upis podataka u fajl
  - count – broj bajtova iz niza koji se upisuje u fajl
- Čitanje int ReadByte()
  - vraća -1 kada dođe do kraja toka, u suprtnom treba kastovati vraćenu vrednost u bajt
- Čitanje int Read(byte[] array, int offset, int count)
  - vraća broj pročitanih bajtova iz toka u niz
  - offset – indeks u nizu bajtova odakle počinje upis bajtova u niz
  - count – broj bajtova koji se čita iz fajla i upisuje u niz

Ulaz/Izlaz – klasa System.IO.FileStream

- Pozicioniranje long Seek(long offset, SeekOrigin origin)
  - offset – pozicija relativna u odnosu na origin kojoj se pristupa
  - origin – neka od vrednosti iz enumeracije SeekOrigin – Begin, Current, End
- Primer: projekat FileStreamPrimer

Digresija – IDisposable i ključna reč using

- Postoji interfejs IDisposable koji sadrži samo jednu metodu Dispose
- Implementiraju ga obično klase koje rade sa eksternim (unmanaged) resursima koji nisu pod kontrolom CLR-a (npr. fajlovi, konekcije na baze podataka itd.)
- Sve klase za rad sa fajlovima implementiraju interfejs IDisposable
- Kod klasa za rad sa fajlovima Dispose metoda u sebi sadrži poziv Close metode
- Klase koje implementiraju IDisposable mogu se koristiti u okviru using bloka (using blok radi slično kao try-with-resources blok u Javi)

- Primer: projekat FileStreamUsing

Digresija – IDisposable i ključna reč using

```
FileStream fajlZaUpis = null;
try {
        fajlZaUpis = new FileStream("FajlZaStream.txt", FileMode.Append);
  // kod za upis u fajl
}
finally {
  fajlZaUpis.Dispose();
}
```

```
// sledeći blok koda je skraćeni zapis bloka sa leve strane tj. kompajler ga prevodi u blok sa leve strane
using (FileStream fajlZaUpis = new FileStream("FajlZaStream.txt", FileMode.Append))
{
    // kod za upis u fajl
}
```

Digresija – IDisposable i ključna reč using

```
FileStream fajlZaUpis = null;
try {
      fajlZaUpis = new FileStream ("FajlZaStream.txt", FileMode.Append);
  // kod za upis u fajl
}
catch (Exception e) {
  // obrada izuzetka
}
finally {
  fajlZaUpis.Dispose();
}
```

```
// ako nam treba i catch blok onda using ide u okviru posebnog try bloka
try {
  using (FileStream fajlZaUpis = new FileStream("FajlZaStream.txt", FileMode.Append))
 {
    // kod za upis u fajl
}
catch (Exception e) {
    // obrada izuzetka
}
```

Digresija – IDisposable i ključna reč using

- Najbolji način za otvaranje bilo kog toka podataka u C#u je kao na prethodne 2 strane sa using, tako je obezbeđeno sigurno zatvaranje toka i kada rad sa tokom uspe i kada dođe do greške
- Prihvatljive su i opcije bez using bloka, ali tada je **obavezan** finally blok u kome se zove Close ili Dispose za zatvaranje toka

Ulaz/Izlaz – *Readers* & *Writers*

- Klase za čitanje i upis
- Za razliku od *Stream* klasa podržavaju i druge tipove podataka pored bajtova
- Klase BinaryReader i BinaryWriter čitaju i upisuju primitivne tipove u tok
- Klase TextReader i TextWriter su apstraktne klase koje implementiraju metode za čitanje i upis karaktera
  - Izvedene klase StreamReader i StreamWriter, koje čitaju i upisuju u tok
  - Izvedene klase StringReader i StringWriter, koje čitaju i upisuju u string ili StringBuilder objekte

Ulaz/Izlaz – *BinaryWriter*

- Klasa System.IO.BinaryWriter za formatirani upis u tok podataka (fajl)
- Implementira interfejs IDisposable
- Konstruktor BinaryWriter(Stream output)
  - output – izlazni tok podataka, mi ćemo uglavnom da koristimo FileStream
- Konstruktor BinaryWriter(Stream output, Encoding encoding)
  - encoding – instanca klase System.Text.Encoding koja se dobija iz nekoh od statičkih property-ja klase: Encoding.ASCII, Encoding.BigEndianUnicode, Encoding.Default, Encoding.Unicode, Encoding.UTF32, Encoding.UTF7, Encoding.UTF8
- Property Stream BaseStream { get; } vraća tok podataka nad kojim radi BinaryWriter

Ulaz/Izlaz – *BinaryWriter*

- Metode za upis podataka u fajl
- void Write(bool value)
- void Write(byte value)
- void Write(byte[] buffer)
- void Write(byte[] buffer, int index, int count)
- void Write(char ch)
- void Write(char[] chars)
- void Write(char[] chars, int index, int count)
- void Write(decimal value)
- void Write(double value)
- void Write(short value)
- void Write(int value)
- void Write(long value)
- void Write(sbyte value)
- void Write(float value)
- void Write(string value)
- void Write(ushort value)
- void Write(uint value)
- void Write(ulong value)
- void Write7BitEncodedInt(int value)

Ulaz/Izlaz – *BinaryWriter*

- Pozicioniranje long Seek(long offset, SeekOrigin origin)
  - offset – pozicija relativna u odnosu na origin kojoj se pristupa
  - origin – neka od vrednosti iz enumeracije SeekOrigin – Begin, Current, End
- Pražnjenje svih internih bafera toka void Flush()
- Zatvaranje toka void Close()
- Primer: projekat BinaryRWPrimer

Ulaz/Izlaz – *BinaryReader*

- Klasa System.IO.BinaryReader za formatirano čitanje iz toka podataka (fajla)
- Implementira interfejs IDisposable
- Konstruktor BinaryReader(Stream input)
  - input – ulazni tok podataka, mi ćemo uglavnom da koristimo FileStream
- Konstruktor BinaryReader(Stream input, Encoding encoding)
  - encoding – instanca klase System.Text.Encoding koja se dobija iz nekoh od statičkih property-ja klase: Encoding.ASCII, Encoding.BigEndianUnicode, Encoding.Default, Encoding.Unicode, Encoding.UTF32, Encoding.UTF7, Encoding.UTF8
- Property Stream BaseStream { get; } vraća tok podataka nad kojim radi BinaryReader

Ulaz/Izlaz – *BinaryReader*

- Metode za čitanje podataka iz fajla
- int Read() – vraća naredni karakter iz toka podataka ili -1 ako se došlo do kraja toka
- bool ReadBoolean()
- byte ReadByte()
- byte[] ReadBytes(int count)
- int Read(byte[] buffer, int index, int count)
- char ReadChar()
- char[] ReadChars(int count)
- int Read(char[] buffer, int index, int count)
- decimal ReadDecimal()
- double ReadDouble()
- short ReadInt16()
- int ReadInt32()
- long ReadInt64()
- sbyte ReadSByte()
- float ReadSingle() - Single je u stvari float
- string ReadString()
- ushort ReadUInt16()
- uint ReadUInt32()
- ulong ReadUInt64()
- int Read7BitEncodedInt()

Ulaz/Izlaz – *BinaryReader*

- Ne postoji Seek metoda
- Metodu BaseStream.Seek treba koristiti veoma oprezno jer može da dođe do gubitka ili pogrešnog čitanja podataka
- Punjenje internog bafera zadatim brojem bajtova iz toka void FillBuffer(int numBytes)
- Zatvaranje toka void Close()
- Primer: projekat BinaryRWPrimer

Ulaz/Izlaz – *StreamWriter*

- Klasa System.IO.StreamWriter za upis u tekstualni fajl
- Izvedena iz apstraktne klase TextWriter
- Konstruktor StreamWriter(string path)
  - path – apsolutna ili relativna putanja do fajla za upis
- Konstruktor StreamWriter(string path, bool append)
  - append – ako fajl ne postoji nema efekta, ako fajl postoji i vrednost je true novi podaci se dopisuju na postojeće (*append*), ako fajl postoji i vrednost je false novi podaci se upisuju od početka fajla (*overwrite*)
- Konstruktor public StreamWriter(string path, bool append, Encoding encoding)
  - encoding – instanca klase System.Text.Encoding koja se dobija iz nekoh od statičkih property-ja klase: Encoding.ASCII, Encoding.BigEndianUnicode, Encoding.Default, Encoding.Unicode, Encoding.UTF32, Encoding.UTF7, Encoding.UTF8

Ulaz/Izlaz – *StreamWriter*

- Konstruktor StreamWriter(Stream stream)
  - output – izlazni tok podataka, mi ćemo uglavnom da koristimo FileStream
- Konstruktor StreamWriter(Stream stream, Encoding encoding)
  - encoding – instanca klase System.Text.Encoding koja se dobija iz nekoh od statičkih property-ja klase: Encoding.ASCII, Encoding.BigEndianUnicode, Encoding.Default, Encoding.Unicode, Encoding.UTF32, Encoding.UTF7, Encoding.UTF8
- Property Stream BaseStream { get; } vraća tok podataka nad kojim radi StreamWriter
- Pražnjenje svih internih bafera toka void Flush()
- Zatvaranje toka void Close()
- Primer: projekat StreamRWPrimer

Ulaz/Izlaz – *StreamWriter*

- Metode za upis podataka u fajl
- void Write(bool value)
- void Write(char ch)
- void Write(char[] chars)
- void Write(char[] chars, int index, int count)
- void Write(decimal value)
- void Write(double value)
- void Write(int value)
- void Write(long value)
- void Write(object value)
- void Write(float value)
- void Write(string value)
- void Write(string format, object arg0)
- void Write(string format, object arg0, object arg1)
- void Write(string format, object arg0, object arg1, object arg2)
- void Write(string format, params object[] arg)
- void Write(uint value)
- void Write(ulong value)

Ulaz/Izlaz – *StreamWriter*

- Metode za upis podataka u fajl sa prelazom u novi red
- void WriteLine()
- void WriteLine(bool value)
- void WriteLine(char ch)
- void WriteLine(char[] chars)
- void WriteLine(char[] chars, int index, int count)
- void WriteLine(decimal value)
- void WriteLine(double value)
- void WriteLine(int value)
- void WriteLine(long value)
- void WriteLine(object value)
- void WriteLine(float value)
- void WriteLine(string value)
- void WriteLine(string format, object arg0)
- void WriteLine(string format, object arg0, object arg1)
- void WriteLine(string format, object arg0, object arg1, object arg2)
- void WriteLine(string format, params object[] arg)
- void WriteLine(uint value)
- void WriteLine(ulong value)

Ulaz/Izlaz – *StreamReader*

- Klasa System.IO.StreamReader za čitanje iz tekstualnog fajla
- Izvedena iz apstraktne klase TextReader
- Konstruktor StreamReader(string path)
  - path – apsolutna ili relativna putanja do fajla za čitanje
- Konstruktor StreamReader(string path, bool detectEncodingFromByteOrderMarks)
  - detectEncodingFromByteOrderMarks – da li se proverava redosled bajtova (big endian, little endian) prema oznakama na početku fajla
- Konstruktor public StreamReader(string path, Encoding encoding, bool detectEncodingFromByteOrderMarks)
  - encoding – instanca klase System.Text.Encoding koja se dobija iz nekoh od statičkih property-ja klase: Encoding.ASCII, Encoding.BigEndianUnicode, Encoding.Default, Encoding.Unicode, Encoding.UTF32, Encoding.UTF7, Encoding.UTF8

Ulaz/Izlaz – *StreamReader*

- Metode za čitanje podataka iz toka
- int Read()
- int Read(char[] buffer, int index, int count)
- int ReadBlock(char[] buffer, int index, int count)
- string ReadLine()
- string ReadToEnd()

Ulaz/Izlaz – *StreamReader*

- Property Stream BaseStream { get; } vraća tok podataka nad kojim radi StreamReader
- Zatvaranje toka void Close()

- Primer: projekat StreamRWPrimer

Ulaz/Izlaz – klasa Console

- Statička klasa za rad sa standardnim tokovima podataka za ulaz, izlaz i greške kod konzolnih aplikacija
- Važni property-ji:
- public static TextWriter Out { get; } – vraća standardni izlazni tok
- public static TextReader In { get; } – vraća standardni ulazni tok
- public static TextWriter Error { get; } - vraća standardni tok greške
- public static Encoding OutputEncoding { get; set; } – postavlja i vraća encoding koji se kortisti za standardni izlaz
- public static Encoding InputEncoding { get; set; } – postavlja i vraća encoding koji se kortisti za standardni ulaz

Ulaz/Izlaz – klasa Console

- Važni property-ji:
- public static Encoding OutputEncoding { get; set; } – postavlja i vraća encoding koji se kortisti za standardni izlaz
- public static Encoding InputEncoding { get; set; } – postavlja i vraća encoding koji se kortisti za standardni ulaz
- Po defaultu konzola radi sa osnovnim ASCII enkodingom koji sadrži slova samo iz osnovnog engleskog alfabeta
- Da bismo omogućili da naša konzolna aplikacija prihvata i prikazuje slova i van osnovnog engleskog alfabeta (npr. slova specifična za srpski jezik) treba na početku aplikacije izvršiti ove naredbe:

  Console.InputEncoding = Console.OutputEncoding = Encoding.Unicode;

Ulaz/Izlaz – klasa Console – metode za ispis

- Podržane su sve metode za ispis iz klase TextWriter
- void Write(bool value)
- void Write(char ch)
- void Write(char[] chars)
- void Write(char[] chars, int index, int count)
- void Write(decimal value)
- void Write(double value)
- void Write(int value)
- void Write(long value)
- void Write(object value)
- void Write(float value)
- void Write(string value)
- void Write(string format, object arg0)
- void Write(string format, object arg0, object arg1)
- void Write(string format, object arg0, object arg1, object arg2)
- void Write(string format, params object[] arg)
- void Write(uint value)
- void Write(ulong value)

Ulaz/Izlaz – klasa Console – metode za ispis

- Podržane su sve metode za ispis iz klase TextWriter sa prelazom u novi red
- void WriteLine()
- void WriteLine(bool value)
- void WriteLine(char ch)
- void WriteLine(char[] chars)
- void WriteLine(char[] chars, int index, int count)
- void WriteLine(decimal value)
- void WriteLine(double value)
- void WriteLine(int value)
- void WriteLine(long value)
- void WriteLine(object value)
- void WriteLine(float value)
- void WriteLine(string value)
- void WriteLine(string format, object arg0)
- void WriteLine(string format, object arg0, object arg1)
- void WriteLine(string format, object arg0, object arg1, object arg2)
- void WriteLine(string format, params object[] arg)
- void WriteLine(uint value)
- void WriteLine(ulong value)

Ulaz/Izlaz – klasa Console – metode za unos teksta

- Podržane su sledeće od metoda za unos teksta iz klase TextReader
- int Read()
- string ReadLine()
- string ReadToEnd()

Ulaz/Izlaz – klasa Console – dva načina za pozivanje metoda za upis/čitanje

- Poziv metoda preko property-ja In/Out:
- Console.In.Read(), Console.In.ReadLine()
- Console.Out.Write(…), Console.Out.WriteLine(…)

- Poziv metoda direktno preko objekta Console, isto radi što i poziv preko In/Out property-ja samo je skraćeno pisanje koda:
- Console.Read(), Console.ReadLine()
- Console.Write(…), Console.WriteLine(…)

Klase za rad sa fajl sistemom

- Klase File i FileInfo
  - File je klasa koja ima statičke metode za kreiranje (create), kopiranje (copy), brisanje (delete), premeštanje (move) i otvaranje (open) fajlova
  - FileInfo je klasa koja sadrži iste metode kao i klasa File sa tom razlikom da metode nisu statičke
- Klase Directory i DirectoryInfo
  - Directory je klasa koja ima statičke metode za kreiranje, premeštanje i listanje članova direktorijuma
  - DirectoryInfo je klasa koja sadrži iste metode kao i klasa Directory sa tom razlikom da metode nisu statičke
- Primer: projekat FajlSistem

Klase za rad sa fajl sistemom - *FileInfo*

- Konstruktor FileInfo(string fileName)
  - fileName – apsolutna ili relativna putanja do fajla
- DirectoryInfo Directory { get; }
  - vraća instancu DirectoryInfo klase koja predstavlja direktorijum u kome je smešten fajl
- string DirectoryName { get; }
  - vraća apsolutnu putanju direktorijuma u kome je smešten fajl
- bool Exists { get; }
  - true ako fajl postoji, false ako ne postoji ili zadata putanja predstavlja direktorijum

Klase za rad sa fajl sistemom - *FileInfo*

- string FullName { get; }
  - vraća apsolutnu putanju do fajla (direktorijum + naziv fajla)
- string Name { get; }
  - vraća samo naziv fajla, bez putanje
- long Length { get; }
  - vraća veličinu fajla u bajtovima

Klase za rad sa fajl sistemom - *FileInfo*

- FileInfo CopyTo(string destFileName)
  - kopira tekući fajl na lokaciju zadatu parametrom destFileName, ukoliko takav fajl već postoji kopiranje nije moguće
- FileInfo CopyTo(string destFileName, bool overwrite)
  - overwrite - true dozvoljava kopiranje preko postojećeg fajla, false ne dozvoljava
- FileStream Create()
  - kreira novi fajl
- void Delete()
  - trajno briše fajl (ne prebacuje u Recycle Bin)
- void MoveTo(string destFileName)
  - destFileName – puna putanja lokacije na koju se premešta fajl

Klase za rad sa fajl sistemom - *File*

- Statička klasa System.IO.File
- static bool Exists(string path)
  - vraća true ako fajl na putanji path postoji, false ako ne postoji
- static void Copy(string sourceFileName, string destFileName)
  - kopira fajl sa lokacije sourceFileName na lokaciju string destFileName, ukoliko takav fajl već postoji kopiranje nije moguće
- static void Copy(string sourceFileName, string destFileName, bool overwrite)
  - kopira fajl sa lokacije sourceFileName na lokaciju string destFileName, ukoliko takav fajl već postoji kopiranje nije moguće
  - overwrite - true fajl se može prekopirati preko postojećeg fajla, false znači da ne može

Klase za rad sa fajl sistemom - *File*

- static FileStream Create(string path)
  - kreira novi fajl na zadatoj putanji path
- static void Delete(string path)
  - trajno briše fajl (ne prebacuje u Recycle Bin)
- static void Move(string sourceFileName, string destFileName)
  - sourceFileName – relativna ili apsolutna putanja do fajla koji se premešta
  - destFileName – puna putanja lokacije na koju se premešta fajl

Klase za rad sa fajl sistemom - *DirectoryInfo*

- Klasa System.IO.DirectoryInfo za rad sa direktorijumima na fajl sistemu
- Konstruktor DirectoryInfo(string path)
  - path – putanja do direktorijuma
- DirectoryInfo Parent { get; }
  - vraća instancu DirectoryInfo klase koja predstavlja direktorijum u kome je smešten tekući direktorijum, vraća null ako je tekući direktorijum root
- DirectoryInfo Root { get; }
  - vraća instancu DirectoryInfo klase koja predstavlja root direktorijum tekućeg direktorijuma
- bool Exists { get; }
  - true ako direktorijum postoji, false ako ne postoji

Klase za rad sa fajl sistemom - *DirectoryInfo*

- string FullName { get; }
  - vraća celu apsolutnu putanju do direktorijuma
- string Name { get; }
  - vraća samo naziv tekućeg direktorijuma, ne i celu putanju
- void Create()
  - kreira novi direktorijum
- DirectoryInfo CreateSubdirectory(string path)
  - kreira poddirektorijum tekućeg direktorijuma na putanji path

Klase za rad sa fajl sistemom - *DirectoryInfo*

- void Delete()
  - briše tekući direktorijum
- void Delete(bool recursive)
  - recursive – true briše tekući direktorijum, sve poddirektorijume i fajlove, false briše samo tekući direktorijum
- DirectoryInfo[] GetDirectories()
  - vraća poddirektorijume tekućeg direktorijuma
- DirectoryInfo[] GetDirectories(string searchPattern)
  - searchPattern – naziv direktorijuma koji se traži, podržava i džoker znake * i ? (npr. “*photo*”)
- DirectoryInfo[] GetDirectories(string searchPattern, SearchOption searchOption)
  - searchOption – neka od vrednosti System.IO.SearchOption enumeracije: AllDirectories ili TopDirectoryOnly

Klase za rad sa fajl sistemom - *DirectoryInfo*

- FileInfo[] GetFiles()
  - vraća fajlove koji se nalaze u tekućem direktorijumu
- FileInfo[] GetFiles(string searchPattern)
  - searchPattern – naziv fajla koji se traži, podržava i džoker znake * i ? (npr. “*.txt”)
- FileInfo[] GetFiles(string searchPattern, SearchOption searchOption)
  - searchOption – neka od vrednosti System.IO.SearchOption enumeracije: AllDirectories ili TopDirectoryOnly
- void MoveTo(string destDirName)
  - premešta direktorijum i njegov sadržaj na zadatu putanju
  - destDirName – puna putanja lokacije na koju se premešta direktorijum

Klase za rad sa fajl sistemom - *Directory*

- Statička klasa System.IO.DirectoryInfo za rad sa direktorijumima na fajl sistemu
- static DirectoryInfo GetParent(string path)
  - vraća instancu DirectoryInfo klase koja predstavlja direktorijum u kome je smešten tekući direktorijum, vraća null ako je tekući direktorijum root
  - path – apsolutna ili relativna putanja do direktorijuma čiji se roditeljski direktorijum traži
- static bool Exists(string path)
  - vraća true ako direktorijum postoji, false ako ne postoji
  - path – apsolutna ili relativna putanja do direktorijuma čiji se roditeljski direktorijum traži

Klase za rad sa fajl sistemom - *Directory*

- static DirectoryInfo CreateDirectory(string path)
  - kreira sve direktorijume i poddirektorijume zadate putanjom path
- static void Delete(string path)
  - path - apsolutna ili relativna putanja direktorijuma za brisanje
- static void Delete(string path, bool recursive)
  - recursive – true briše tekući direktorijum, sve poddirektorijume i fajlove, false briše samo tekući direktorijum
- static void Move(string sourceDirName, string destDirName)
  - premešta direktorijum ili fajl na zadatu lokaciju
  - sourceDirName – fajl ili direktorijum koji se premešta
  - destDirName – putanja nove lokacije na koju se radi premeštanje, ako je sourceDirName fajl onda i destDirName mora biti fajl

Klase za rad sa fajl sistemom - *Directory*

- static string[] GetDirectories(string path)
  - vraća poddirektorijume zadatog direktorijuma
  - path - apsolutna ili relativna putanja direktorijuma koji se pretražuje
- static string[] GetDirectories(string path, string searchPattern)
  - searchPattern – naziv direktorijuma koji se traži, podržava i džoker znake * i ? (npr. “*photo*”)
- static string[] GetDirectories(string path, string searchPattern, SearchOption searchOption)
  - searchOption – neka od vrednosti System.IO.SearchOption enumeracije: AllDirectories ili TopDirectoryOnly

Klase za rad sa fajl sistemom - *Directory*

- static string[] GetFiles(string path)
  - vraća fajlove koji se nalaze u zadatom direktorijumu
- static string[] GetFiles(string path, string searchPattern)
  - searchPattern – naziv fajla koji se traži, podržava i džoker znake * i ? (npr. “*.txt”)
- static string[] GetFiles(string path, string searchPattern, SearchOption searchOption)
  - searchOption – neka od vrednosti System.IO.SearchOption enumeracije: AllDirectories ili TopDirectoryOnly