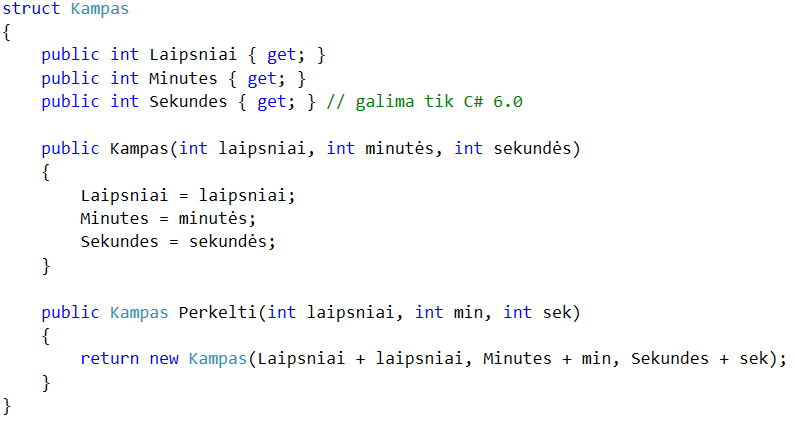
# **KODO KONTRAKTAI**

## STRUKTŪROS

* Struktūros yra kaip ir klasės, išskyrus:
* Struktūros yra reikšmių tipai, o klasės – nuorodų tipai
* Struktūroms negalimas paveldėjimas
* Struktūra negali turėti konstruktoriaus be parametrų, tačiau sistema jį suteikia
* Struktūra negali turėti virtualių narių
* Apibrėžiant konstruktorių, reikia priskirti visiems laukams reikšmes
* Struktūra negali turėti užbaigimo (finalizer) metodo
* Paveldi System.ValueType
* Rekomenduojama nekurti struktūrų, užimančių daugiau kaip 16 baitų

*Pavyzdys:*



## REIKŠMĖ NULL IR TIPAI

* Reikšmės ***null*** pirmiausia reikia, programuojant duomenų bazes, nes būna stulpeliuose neužpildytų reikšmių. Gerai, jei tipas ***string***, tačiau skaitmeniniai tipai neturi ***null*** reikšmės.
* Reikšme ***null*** galim pasinaudoti išvestinėse klasėse, jei leidžiam išvestinei klasei turėti neapibrėžtą savybę, kurią tokiais atvejais galime paimti iš bazinės klasės.
* Reikšmė ***null*** padeda tais atvejais, kai grąžinimui reikia „magiškos“ reikšmės pranešimui apie nesėkmę, bet papildomai nenaudojam loginio kintamojo.

## DIAGNOSTIKA IR KODO KONTRAKTAI

**Diagnostikos poreikis**

Kai gaunami netinkami rezultatai, reikia diagnostinės informacijos. Derinimo metu – nesudėtinga, tačiau, jei programa jau eksploatacijoje. Tam sudaromi kodo kontraktai.

Kodo kontraktai leidžia metodams bendrauti per tarpusavio įsipareigojimus. Metodai gali išduoti diagnostiką, jei šie kontraktai pažeidžiami.

Dauguma tipų diagnostikai yra ***System.Diagnostics*** ir ***System.Diagnostics.Contracts***.

**Sąlyginio kompiliavimo direktyvos**

* Naudojamos preprocesoriaus direktyvos. Prasideda #. Kompiliavimui tokios direktyvos: #if, #else, #endif, #elif.
* Sąlygai tikrinti naudojami apibrėžti simboliais. Jie gali būti nusakyti direktyva #define arba kompiliavimo raktu.
* #define taikoma atskiram failui; kompiliavimo raktas taikomas visai asamblėjai
* #define turi būti failo viršuje, pagal susitarimą rašoma didžiosiomis raidėmis

**Sąlyginio kompiliavimo privalumai**

1. Galima įjungti atributą
2. Galima keisti deklaruojamą kintamojo tipą
3. Galima perjungti tarp skirtingų vardų erdvių, įtraukiant skirtingas ***using*** direktyvas. Taip galima įtraukti seną ar naują bibliotekos versiją

**Atributas Conditional**

Atributas *Conditional* nurodo kompiliatoriui ignoruoti visus metodo kvietimus, jei nurodytas simbolis neapibrėžtas.

**Debug ir Trace klasės**

* Debug ir Trace yra statinės klasės, suteikiančios bazines įrašymo ir teigimo galimybes.
* Klasės panašios; pagrindinis skirtumas – jų naudojimo paskirtis.
* ***Debug*** klasė skirta programos derinimo eigai.
* ***Trace*** klasė skirta ir derinimui, ir programos eksploatavimui.
* Abi klasės turi metodus *Write, WriteLine, WriteIf*.
* Abi klasės turi metodus *Fail* ir *Assert.*

**Kodo kontraktai**

Kai funkcijos bendrauja per įsipareigojimus ir naudas (prieš sąlyga, kurią nurodo funkcija, ją užtikrina funkcijos klientas, po sąlyga – ją užtikrina funkcija).

* Kontraktai turi būti metodo priekyje. Tai gero projekto reikalavimas.
* Metodo sąlygos:
  + Kviečiamas su ne *null* įrašomu sąrašu
  + Kai grąžins, sąrašas turės pateiktą elementą
* Šie faktai gali nukeliauti į XML dokumentacijos failą
* Kontraktų išpildymą gali tikrinti statinės analizės priemonės Vartotojas gali numatyti, kaip reaguoti į kontraktų pažeidimus
* Kontraktų trūkumas – įgyvendinimas remiasi *binary rewriter* įrankiu, kuris keičia asamblėjos kodą po kompiliacijos. Lėtina bendrą procesą
* Kodo kontraktus sudaro:
* Prieš sąlygos – tikrinama, kai funkcija įeina
* Po sąlygos - tikrinama, prieš funkcijos išėjimą
* Teigimai – tikrinama, kai sutinkama kode
* Objektų invariantai – tikrinama, po kiekvienos atvirosios funkcijos
* Jie skiriasi tuo, kada yra tikrinami.
* Kodo kontraktus apibrėžia *Contract* klasės statiniai metodai. Todėl jie nepriklausomi nuo kalbos
* Jie gali pasirodyti ne tik metoduose, bet it konstruktoriuose, savybėse, užklotuose operatoriuose
* Visos funkcijos, kurios kviečiamos kontraktų metodų argumentuose, turi būti grynosios be šalutinių efektų
* Visa, kas kviečiama kontrakto metodų argumentuose, turi būti grynieji: sąsajos metodai, LINQ operatoriai, delegatai.

*Prieš sąlygos:*

Nustato 3 metodai:

1. **Contract.Requires();**
2. **Contract.Requires<TEXception>();**
3. **Contract.EndContractBlock();**

Pasirinkimas tarp išimties ir prieš sąlygos:

* Jei nesėkmė visuomet rodo klaidą pas klientą, pasirinkite prieš sąlygą
* Jei nesėkmė rodo nenormalią sąlygą, kuri galėtų reikšti klaidą, pasirinkite išimtį

*Po sąlygos:*

* **Contract.Ensures(...);**
* Binary rewriter perkelia po sąlygą į kamieno pabaigą. Po sąlyga tikrinama ir tuomet, kai grįžtate iš metodo anksčiau, kaip ankstesniame pavyzdyje. Bet netikrinama, jei grįžtate per nevaldomą išimtį.
* Po sąlygos aptinka klaidas esamam metode, gali pasiekti privačią būseną **Contract.EnsuresOnThrown**
* Kartais svarbu, kai tam tikra sąlyga yra, išmesti tam tikrą išimtį: **Contract.EnsuresOnThrown(this.ErrorMessage != null);**

*Objekto invariantai*

* Vykdomi po kiekvieno atvirojo metodo. Leidžia užtikrinti, kad objekto būsena tinkama. Tam reikia atskiro metodo su spec. atributu.

Puiki kontraktų savybė – galima prikabinti sąlygas sąsajos nariams ir abstraktiems metodams

## Asamblėja

* Asamblėja yra bazinis išdėstymo vienetas .NET. Ji taip pat yra visų tipų konteineris
* Asamblėja turi kompiliuotus tipus su jų kodais, vykdymo resursus, informaciją dėl versijų, saugumo, nuorodų į kitas asamblėjas
* Bendrai, asamblėja sudaro vienintelį Windows Portable Executable (PE) failą – .exe plėtinys, jei vykdoma programa arba .dll, jei daug kartų naudojama biblioteka

## Refleksija

* Asamblėjos metaduomenų tyrimas programos vykdymo metu vadinamas refleksija
* Kompiliuotas kodas asamblėjoje turi viską iš originalaus kodo, išskyrus:
  + Lokalius kintamųjų vardus
  + Komentarus
  + Preprocesoriaus direktyvas
* Naudojamos System.Reflection ir System.Reflection.Emit vardų erdvės. Nauda:
  + Gaunam metaduomenis
  + Galima dinamiškai iškviesti, naudojant metaduomenis, tipo narius
  + Nauja kryptis programavime

# **IŠIMTYS**

## Įžanga

* Išimtys yra objektai, todėl joms apdoroti taikomi visi objektinio programavimo mechanizmai.
* 4 baziniai žodžiai: ***try, catch, throw, finally***. Programa aptinka išimtis, vykdydama bloką ***try***. Betarpiškai už šio bloko rašomas ***catch*** blokas, kuriame esantys sakiniai nusako, kas turi būti atliekama esant išimčiai. Vienam ***try*** blokui gali būti užrašyta vienas arba keli ***catch*** blokai.
* Pati išimtis generuojama (ir pereinama į išimties apdorojimo bloką) vykdant ***throw*** sakinį. Tokie sakiniai rašomi ***try*** bloke. ***catch*** blokų turi būti tiek, kiek skirtingų tipų išimčių apibrėžta ***throw*** sakiniuose.
* ***finally*** blokas vykdomas visada, nepriklausomai nuo eigos ***try*** bloke. Vykdymas po bloko ***finally*** priklauso nuo eigos ***try*** bloke.

## Rekomendacijos išimčių klasėms

1. Turėtų paveldėti klasę ***ApplicationException***.
2. Vardas turėtų baigtis ***Exception***.
3. Turėtų bent 3 konstruktorius:
4. Be parametrų – numatytasis pranešimas perduodamas bazinei klasei
5. Su string parametru – perduoti klaidos pranešimui
6. Su string parametru – perduoti klaidos pranešimui ir ***Exception*** parametru dėl vidinių išimčių perdavimo

## Išimčių modelis

* Kai ***try*** bloke pasirodo išimtis, vykdymas nutraukiamas ir einama į ***catch*** bloką, jei toks yra. Gali likti neįvykdytų sakinių
* Įvykdžius ***catch*** bloką, valdymas perduodamas žemiau paskutinio ***catch*** bloko.
* Toks išimčių valdymas vadinamas užbaigimo modeliu.
* Jei išimtis neiššaukiama, ***catch*** blokai nevykdomi
* Atitikimas ***catch*** blokui – identiška klasė arba išvestinė klasė

# **Failai ir anoniminiai metodai**

## Klasės File metodai

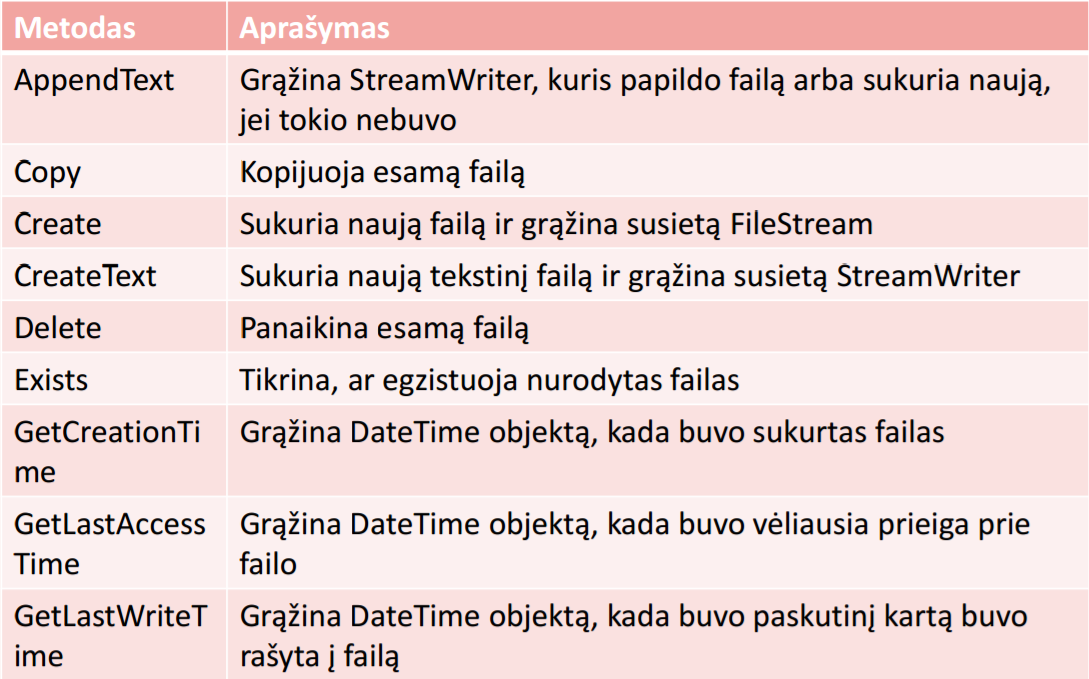
**Skaitymo metodai**

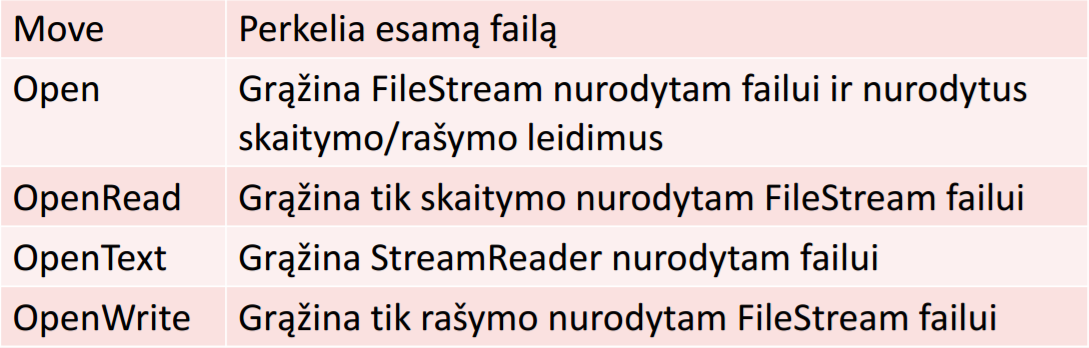
* **ReadAllText** leidžia skaityti visą failo turinį į vieną ***string*** kintamąjį.
* **ReadAllLines** leidžia skaityti visą failą ir saugoti failo eilutes ***string*** masyve.
* **ReadAllBytes** leidžia skaityti failo turinį kaip dvejetainius duomenis ir saugoti ***byte*** masyve.

**Rašymo metodai**

* **WriteAllText** leidžia rašyti ***string*** kintamojo turinį į failą.
* **WriteAllLines** leidžia rašyti ***string*** masyvo turinį į failą.
* **WriteAllBytes** leidžia rašyti ***byte*** masyvo turinį į dvejetainį failą.

**Klasės File statiniai metodai**





* StreamReader – tekstinių failų skaitymas
* StreamWriter – tekstinių failų išvedimas
* FileStream – failų įvedimas ir išvedimas

## Klasės Directory metodai



## Klasės Path metodai

* Artimai susijusi su jau minėtomis 4 klasėmis.
* Pateikia panašius metodus, ypač patogius dėl darbo su laikinais failais
* Metodai statiniai
* Kai kurie metodai:
  + **GetTempPath** paima kelią iki esamo vartotojo Windows laikino katalogo
  + **HasExtension** tikrina, ar yra plėtinys. Galima atskirti failą nuo katalogo
  + **GetExtension** paima plėtinį
  + **GetTempFileName** sukuria laikiną failą laikiniame kataloge ir grąžina pilną kelią

## Objektų nuoseklinimas

* Programos skaito duomenis
* Laikomės tam tikrų susitarimų dėl duomenų – reikia žinoti duomenų tipus
* Kartais patogiau būtų rašyti objektus. Toks mechanizmas vadinamas nuoseklinimu
* Taip galima išsaugoti objektų būsenas, jas pernešti tarp atskirų programų

**Nuoseklinimo formatai**

* Dvejetainis (klasė BinaryFormatter)
* XML (klasė SoapFormatter)
* JSON (klasė DataContractJsonSerializer)
* Vartotojo sukurtas

**Reikalavimai**

• Kiekviena klasė, kurios objektus norime nuoseklinti, turi įgyvendinti sąsają ***ISerializable***

**Dvejetainis formatas**

* Dvejetainis formatas yra nesudėtingas, greitai apdorojamas.
* Naudojamas objektų turinio pernešimui tarp tų pačių platformų.
* Išsaugo formatus
* Sukuriamas dvejetainis failas

**XML formatas**

* Tai atvirasis standartas, tinkamas bet kuriai programai, nepriklauso nuo platformos.
* Neišsaugo formatų
* Turi daugiau žodžių, nes išsaugoti duomenys papildomi XML raktiniais žodžiais
* Mažiau efektyvus, daugiau procesoriaus laiko
* Duomenų perdavimui tarp saityno paslaugų per SOAP protokolą

**JSON formatas**

* Formatas remiasi JavaScript kalbos poaibiu
* Žmogus gali skaityti, nesudėtinga apdoroti kompiuteriu
* Naudojamas duomenų perdavimui tarp asinchroninio JavaScript ir XML (AJAX) kreipinių.

## Bendriniai delegatai

1. Paskelbti delegatą, atitinkantį metodo formatą
2. Paskelbti delegato objektą, konstruktoriui perduodant metodo vardą
3. Iškviesti metodą netiesiogiai

Sukuriami delegatai, tinkami vienai programai

Tačiau tai nėra būtina daryti...

Yra bendriniai **Action<>** ir **Func<>** delegatai, galintys priimti iki 16 parametrų.

* **Action<>** metodams, kurie negrąžina reikšmės
* **Func<>** metodams, kurie grąžina reikšmę

## Anoniminis metodas

1. Anoniminis metodas gali pasiekti to metodo, kuriame jis yra apibrėžtas, lokalius kintamuosius
2. Anoniminis metodas negali pasiekti to metodo, kuriame jis yra apibrėžtas, ref ir out parametrų
3. Anoniminis metodas negali turėti lokalių kintamųjų tais pačiais vardais, kaip ir metode, kuriame jis yra apibrėžtas
4. Anoniminis metodas gali pasiekti to metodo, kuriame jis yra apibrėžtas, klasės kintamuosius
5. Anoniminis metodas gali turėti lokalius kintamuosius sutampančiais vardais su to metodo, kuriame jis yra apibrėžtas, klasės kintamaisiais

# **Sąsajos klasės ir delegatai**

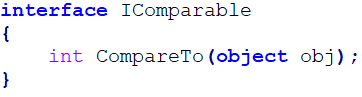
## Sąsajos klasės

* Sąsajos apibrėžia ir standartizuoja būdus, kaip žmonės ir sistemos bendrauja
* Programiniai objektai bendrauja taip pat per sąsajas
* Klasė įsipareigoja įgyvendinti tam tikras sąsajas
* Sąsajos įgyvendinamos per paveldėjimo mechanizmą
* Sąsaja skelbia metodų, kurie gali būti, iškviesti aibę
* Klasė turi įgyvendinti visus paveldėtos sąsajos metodus
* Sąsaja neturi kodo; tik apibrėžia metodus ir savybes, kurias turi įgyvendinti paveldinti klasė
* Naudodami sąsają, pilnai atskiriate paskelbimus nuo įgyvendinimų
* Klasių paveldėjimas – gerai
* Sąsajų paveldėjimas – dar geriau
* Galima paveldėti daug sąsajų. Bet kuri klasė gali įgyvendinti bet kurią sąsają.
* Paveldint sąsajas, klasės nesudaro medžio.
* Sąsajų naudojimas – daug universalesnis variantas, nei abstrakčių klasių virtualūs metodus

**Reikalingumas**

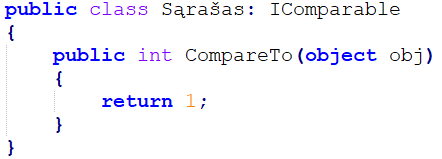
Sakykim, sudarote konteinerio klasę. Norėsite rikiuoti konteinerio elementus. Nenorite apriboti konteinerio tipo. Viskas priklauso nuo elementų palyginimo

**Apibrėžimas**



* Metodas be prieigos modifikatorių;
* Sąsajai rekomenduojama raidė I;
* Ši sąsaja yra įgyvendinta sistemoje;
* Gali būti keletas metodų vienoje sąsajoje.

**Įgyvendinimas**

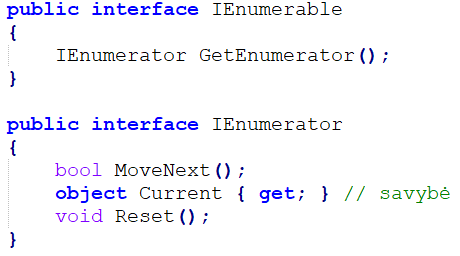


**Reikalavimai**

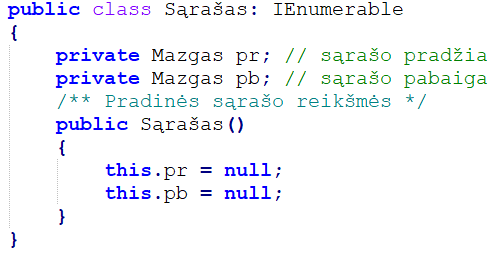
* Metodų vardai ir grįžimo tipai sutampa tiksliai.
* Visi parametrai (ref ir out raktiniai žodžiai) sutampa tiksliai.
* Visi įgyvendinami sąsajos metodai turi būti atviri.
* Prieš metodo vardą yra sąsajos vardas – išreikštas įgyvendinimas, tačiau jis riboja. Verta naudoti, kai iš skirtingų sąsajų ateina metodai tais pačiais vardais

**Iteratorius**

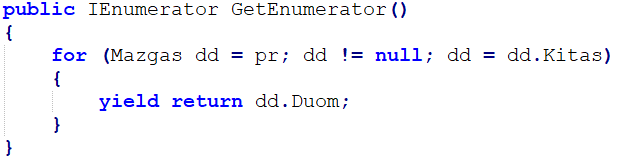
Ciklui per sąrašą sukūrėme ir naudojome sąsajos metodus Būtų patogu naudoti ***foreach*** ciklą Tam yra ***IEnumerable*** sąsaja ir jos metodas ***GetEnumerator()***.



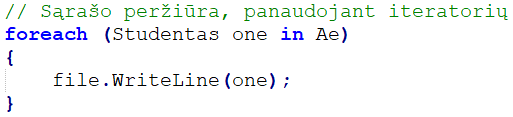
**Klasė su iteratoriumi**



**Įgyvendinimas**



**Panaudojimas**



## Delegatai

* Įprasta siųsti objektui pranešimą, kviečiant metodą.
* Bet objektas norėtų pasiųsti pranešimą atgal, tam kuris iškvietė
* **Delegatas** – objektas, kuris rodo į metodą ar jų sąrašą, kurie gali būti iškviesti vėliau
* Su delegatais dirba ***event*** raktinis žodis ir lambda išraiškos – anoniminiai metodai
* Delegatai leidžia pertraukti nuoseklų programos veikimą, o po to – grįžti ir tęsti
* **Delegatas** – tai rodyklė į metodą
* Kai iškviečiamas delegatas, vykdomas tas metodas, į kurį jis rodo
* Galima dinamiškai keisti metodą, į kurį rodo delegatas.
* Todėl kiekvieną kartą gali būti skirtingas metodas.
* Delegatų prasmė yra, kai yra keletas objektų, turinčių vienodas metodų antraštes pagal parametrų tipus ir grąžinamos reikšmės tipą, bet skirtingus metodų vardus. Objektų tipai gali būti skirtingi.

## Įvykiai

* Tiesioginis delegato kvietimas nėra patogus. Geriau, kad būtų kviečiama pagal įvykį, kai kažkas svarbaus ar netikėto aptinkama. Tam yra ***įvykiai***.
* **Įvykis** – tai klasė, kažką stebinti.
* Įvykis turi metodų, prenumeratorių, kuriuos galima iškviesti dėl įvykio, sąrašą.
* Delegatų objektai skelbiami įvykiais