***1. Sąsajos klasės ir delegatai.*** *Prasmė, poreikis, aprašymas, įgyvendinimas. Kai kurios standartinės sąsajos (palyginimas, iteratoriai, klonavimas). Resursų valdymas. Resursų grąžinimo sąsaja. Delegatų poreikis, naudojimas. Keleto metodų sąrašo formavimas delegatui. Įvykiai.*

*Pavyzdiniai klausimai: Kokią klasę vadiname sąsajos klase? Kokios jos ypatybės? Kas yra įvykiai?*

**Prasmė, neformalus ir formalus apibrėžimai**

Sąsajos apibrėžia ir standartizuoja būdus, kaip žmonės ir sistemos bendrauja (radijo sąsaja, automobilio valdymo sąsaja). Programiniai objektai bendrauja taip pat per sąsajas.

Sąsaja – abstraktus [objekto](https://lt.wikipedia.org/wiki/Objektas_(programavimas)) [klasės](https://lt.wikipedia.org/wiki/Klas%C4%97_(programavimas)) aprašas, nusakantis, kokius [metodus](https://lt.wikipedia.org/wiki/Metodas_(programavimas)) ši klasė privalo turėti.

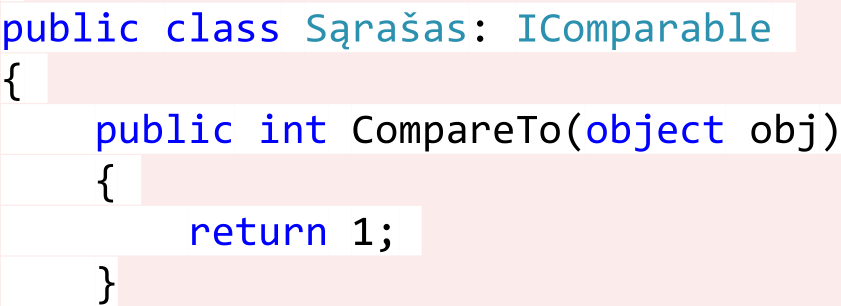
**Klasių palyginimas, poreikis**

|  |  |
| --- | --- |
| Sąsajos klasė | Abstrakti klasė |
| Sąsaja negali turėti realizacijos aprašo. Metodus ir savybes turi įgyvendinti paveldinti klasė. | Leidžia skelbti ir virtualius metodus, kurių neprivalu įgyvendinti. |
| Galima paveldėti daug sąsajų. Bet kuri klasė gali įgyvendinti bet kurią sąsają. | Glaudi sąsaja. Galima paveldėti tik vieną klasę, kuri gali įgyvendinti paveldėtos abstr. klasės metodus. |

**Įgyvendinimas**

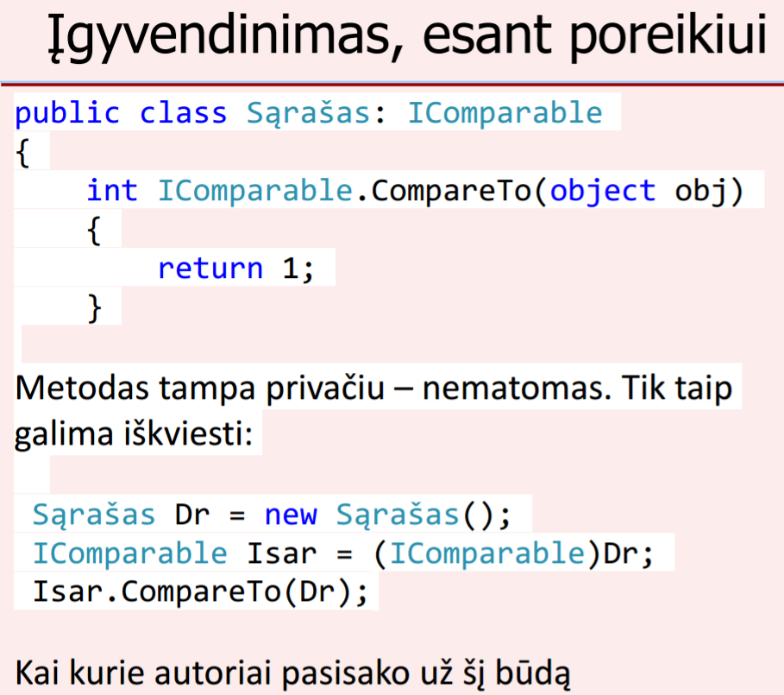
* Sąsaja yra įgyvendinta sistemoje (IComparable, IEnumerable, IEquatable,...).
* Sąsajos įgyvendinamos per paveldėjimo mechanizmą.
* Sąsaja skelbia metodų, kurie gali būti iškviesti, aibę.
* Klasė įsipareigoja įgyvendinti tam tikras sąsajas.
* Klasė turi įgyvendinti visus paveldėtos sąsajos metodus.

Įprastas įgyvendinimas – standartinė palyginimo sąsaja



**Aprašymas**

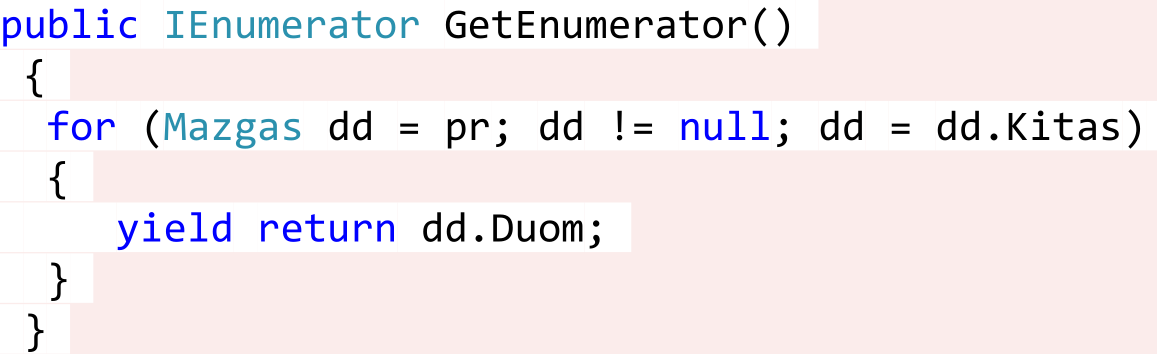
* Gali būti keletas metodų vienoje sąsajoje.
* Metodų vardai ir grįžimo tipai tiksliai sutampa.
* Visi parametrai (*ref* ir *out* raktiniai žodžiai) tiksliai sutampa.
* Visi įgyvendinami sąsajos metodai turi būti atviri. Metodams negalima nurodyti prieigos modifikatorių (private, protected, public).
* Prieš metodo vardą yra sąsajos vardas – išreikštas įgyvendinimas. (Verta naudoti, kai iš skirtingų sąsajų ateina metodai tais pačiais vardais.)



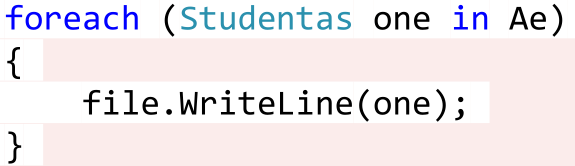
**Standartinių sąsajų pavyzdžiai**

**Iteratoriai** - tai apibendrintos rodyklės, jungiančios konteinerių elementus. Didinant ar mažinant iteratoriaus reikšmę, pasiekiamas vis kitas konteinerio elementas.

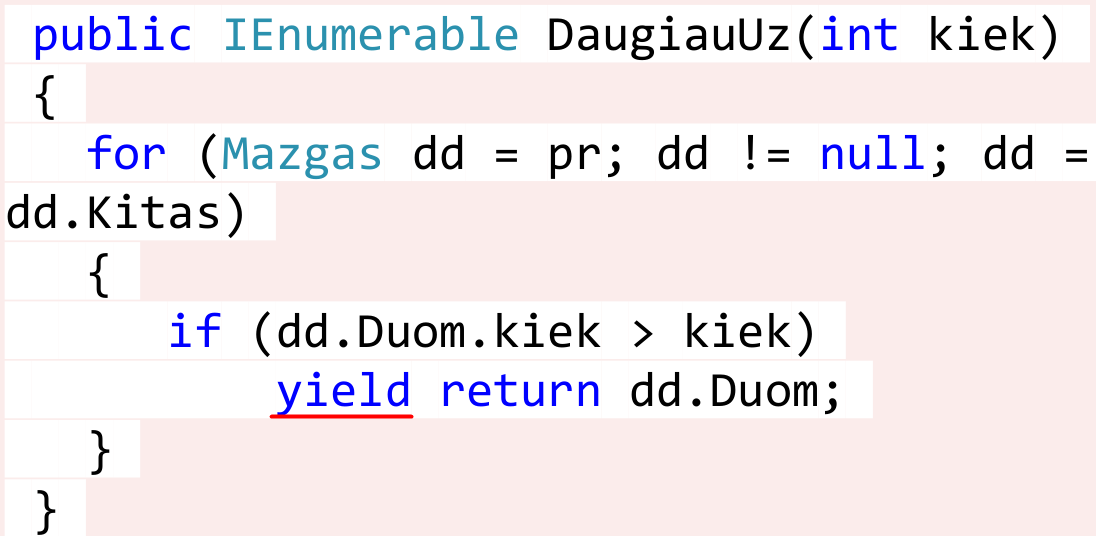
* Įgyvendinimas



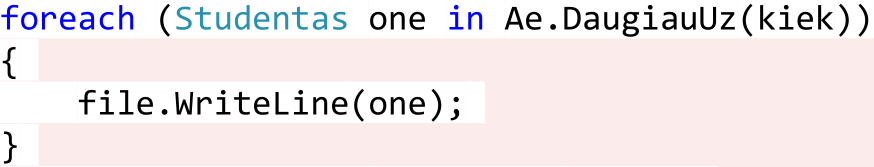
* Panaudojimas



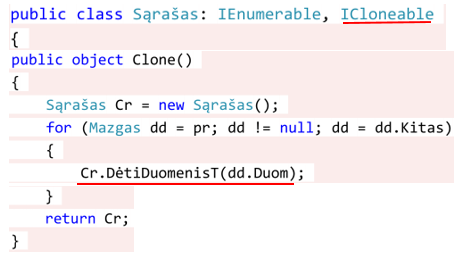
* Specializuotas iteratorius

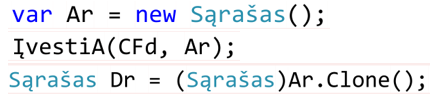


* Panaudojimas



**Klonavimas**

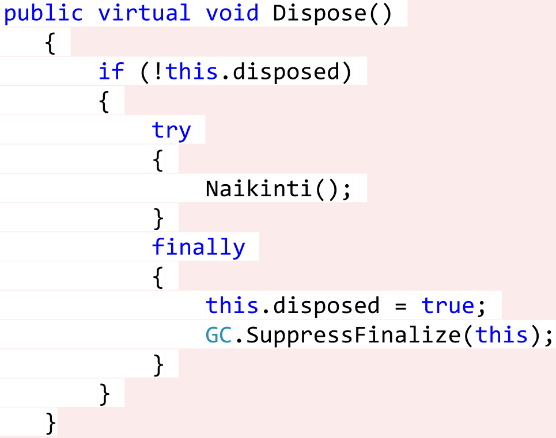


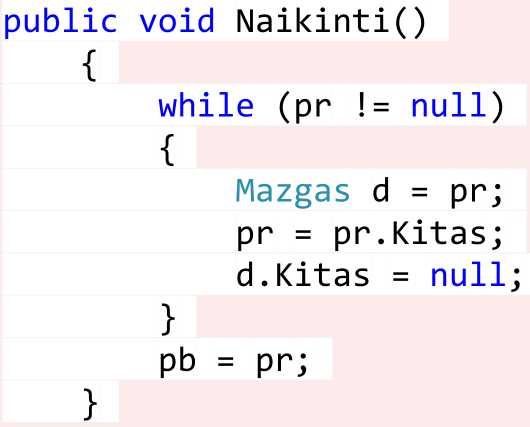


**Resursų/išteklių valdymas**

Objekto sukūrimas: atminties išskyrimas (nevaldote), reikšmių inicializavimas (per konstruktorių).

Objekto naikinimas: išvalymas (per destruktorių), atminties grąžinimas (nevaldote).

**Destruktorius** – specialus objekto metodas, atsakingas už visų objekto užimamų resursų (atminties, užmegztų tinklo jungčių, laikinų bylų) užleidimą kitiems tikslams. Baigus vykdyti destruktoriaus kodą, objektas laikomas „sunaikintu“ – jokie vėlesni kreipiniai į jį nebeleistini.



**Šiukšlių surinkėjas** (GC, garbage collector) – vykdomosios sistemos dalis, automatiškai naikinanti vykdomojo kodo nebepasiekiamus objektus tam, kad nesibaigtų atmintis.

Jis užtikrina:

* Visi objektai bus sunaikinti, jų destruktoriai įvykdyti
* Kiekvienas objektas bus sunaikintas tik kartą
* Bus sunaikinti tik neaktyvūs objektai

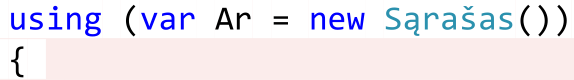
**Objektų kartos:**

* 0 karta: naujai sukurti objektai. Nė karto nepažymėti šalinimui.
* 1 karta: Objektas išgyveno GC (buvo pažymėtas, bet nepašalintas), nes užteko atminties.
* 2 karta: Objektas išgyveno daugiau nei vieną valymą.

GC veiklą pradeda nuo 0 kartos. Tikrina kiekvieną, jei grąžina atmintį ir pakanka, perveda likusius į 1 kartą.

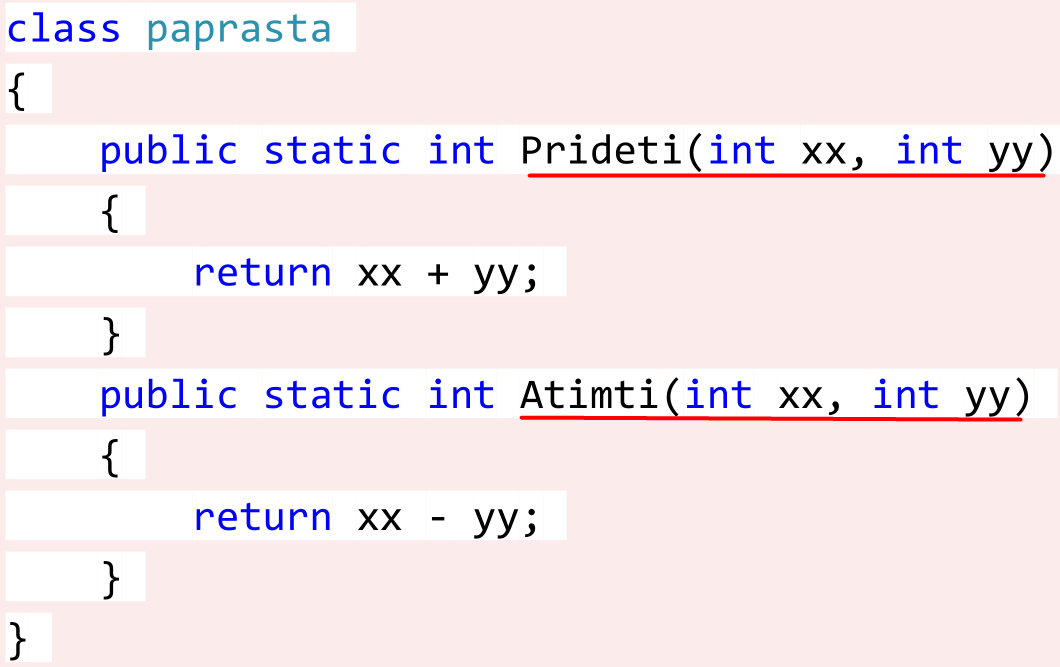
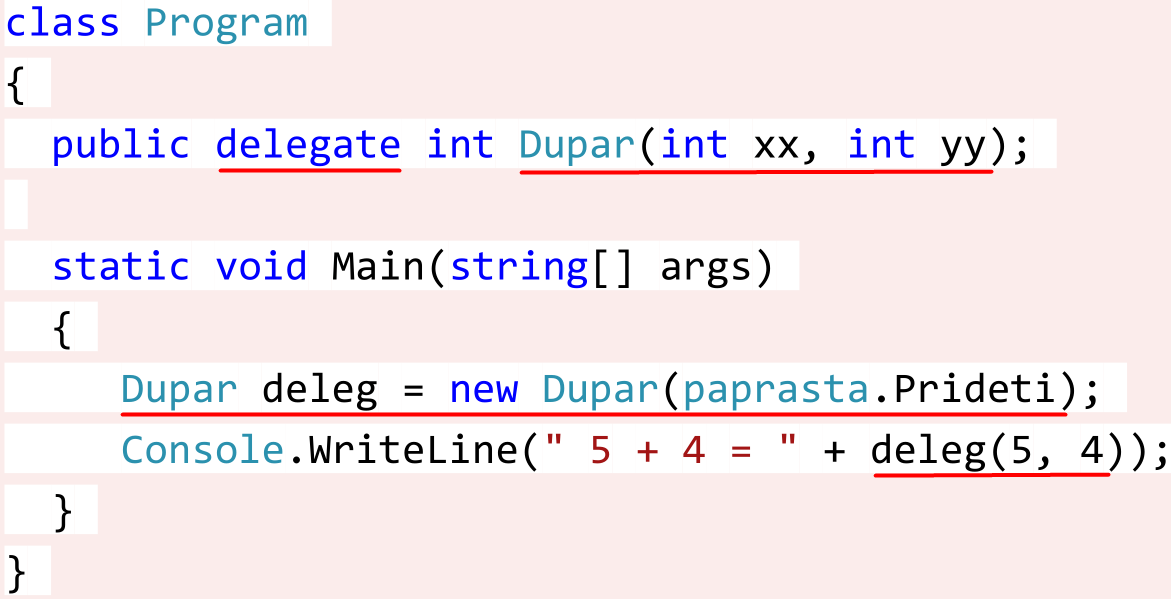
**Resursų/atminties grąžinimo sąsaja:**

Išteklius kartais geriau atiduoti automatiniam apdorojimui. Pvz., metodui, kurį turi StreamReader klasė. Po *using* sakinio iš karto grąžinami ištekliai.



# **Delegatai ir įvykiai**

**Delegatas** – rodyklė į metodą ar jų sąrašą. Iškvietus delegatą, vykdomas tas metodas, į kurį jis rodo. Yapč naudingi, kai yra keletas objektų, turinčių vienodas metodų antraštes, bet skirtingus metodų vardus.

Delegatų poreikis. Jie leidžia:

* Kviesti metodą.
* Siųsti objektui pranešimą, kviečiant metodą, ir atgal gauti pranešimą iš objekto.
* Pertraukti nuoseklų programos veikimą, o po to – grįžti ir tęsti.

Su delegatais dirba *event* raktinis žodis (įvykiai) ir lambda išraiškos – anoniminiai metodai.

**Įvykis** – klasė, stebinti programos dalį ir aptinkanti pasikeitimą, kažką svarbaus ar netikėto (pelės mygtuko, klavišo paspaudimą; programos pranešimą). Įvykis turi metodų, prenumeratorių, kuriuos galima iškviesti dėl įvykio, sąrašą.

***2. Failai ir anoniminiai metodai.*** *Statiniai klasių File , Directory ir Path metodai. Klasių FileInfo ir DirectoryInfo objektai ir jų metodai. Objektų nuoseklinimas (dvejetainiai, XML, JSON). Bendriniai delegatai. Anoniminiai metodai.*

*Pavyzdiniai klausimai: Kas yra anoniminis metodas? Kur jis naudojamas?*

**Statiniai metodai** – algoritmai, priklausomi tik nuo konkrečios klasės statinių duomenų ir bendri visiems tos klasės objektams. Jie gali būti iškviečiami net ir nesukūrus nei vieno tos klasės objekto, tačiau norint, kad metodas galėtų naudotis klasės aprašymu, būtina tai padaryti. Tokie metodai taip pat dažnai naudojami objektų kūrimui, radimui, naikinimui.

Klasė *DirectoryInfo* atlieka panašias funkcijas, kaip ir klasė *Directory*, bet ji yra galingesnė: turi daugiau metodų ir savybių. Tačiau reikia kurti klasės *DirectoryInfo* objektą. Tas pats galioja ir klasei *FileInfo*.





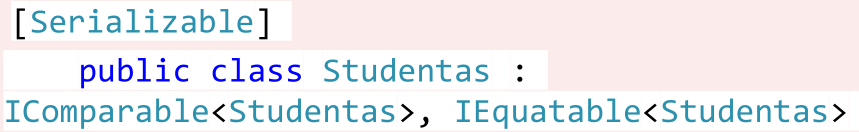
|  |  |
| --- | --- |
| Klasės **File** metodai | |
| ReadAllText | Skaito visą failo turinį į vieną *string* kintamąjį |
| ReadAllLines | Skaito visą failą ir saugo failo eilutes *string* masyve |
| ReadAllBytes | Skaito failo turinį kaip dvejetainius duomenis ir saugo *byte* masyve |
| WriteAllText | Rašo *string* kintamojo turinį į failą |
| WriteAllLines | Rašo *string* masyvo turinį į failą |
| WriteAllBytes | Rašo *byte* masyvo turinį į dvejetainį failą |
| AppendText | Grąžina *StreamWriter*, kuris papildo failą arba sukuria naują, jei tokio nebuvo |
| Copy | Kopijuoja esamą failą į bet kurį katalogą |
| Create | Sukuria naują failą ir grąžina susietą *FileStream* |
| CreateText | Sukuria naują tekstinį failą ir grąžina susietą *StreamWriter* |
| Delete | Panaikina esamą failą |
| Exists | Tikrina, ar egzistuoja nurodytas failas |
| GetCreationTime | Grąžina *DateTime* objektą, kada buvo sukurtas failas |
| GetLastAccess | Grąžina *DateTime* objektą, kada buvo vėliausia prieiga prie failo |
| GetLastWriteTime | Grąžina *DateTime* objektą, kada buvo paskutinį kartą buvo rašyta į failą |
| Move | Perkelia esamą failą |
| Open | Grąžina *FileStream* nurodytam failui ir nurodytus skaitymo/rašymo leidimus |
| OpenRead | Grąžina tik skaitymo nurodytam *FileStream* failui |
| OpenText | Grąžina *StreamReader* nurodytam failui |
| OpenWrite | Grąžina tik rašymo nurodytam FileStream failui |
| Klasės **FileInfo** metodai | |
| Attributes | Paima arba priskiria failui ar katalogui atributus |
| CreationTime | Paima arba priskiria failui ar katalogui sukūrimo laiką |
| CreationTimeUtc | Paima arba priskiria failui ar katalogui sukūrimo laiką UTC (coordinated universal time) formatu |
| Directory | Paima tėvinio katalogo objektą |
| DirectoryName | Paima eilutę su pilnu katalogo keliu |
| Exists | Paima reikšmę, rodančią ar failas egzistuoja |
| Extension | Paima eilutę, rodančią failo plėtinį |
| FullName | Paima pilną failo ar katalogo kelią |
| IsReadOnly | Paima arba priskiria failui reikšmę, nurodančią, ar failas tik skaitymui |
| LastAccessTime | Paima arba priskiria failui ar katalogui paskutinės prieigos laiką |
| LastAccessTimeUtc | Paima arba priskiria failui ar katalogui paskutinės prieigos laiką UTC formatu |
| LastWriteTime | Paima arba priskiria failui ar katalogui paskutinio rašymo laiką |
| LastWriteTimeUtc | Paima arba priskiria failui ar katalogui paskutinio rašymo laiką UTC formatu |
| Length | Paima failo dydį baitais |
| Name | Paima failo vardą |
| Klasės **Directory** metodai | |
| CreateDirectory | Sukuria katalogą ir grąžina *DirectoryInfo* objektą |
| Delete | Naikina nurodytą katalogą |
| Exists | Tikrina, ar egzistuoja nurodytas katalogas |
| GetCurrentDirectory | Grąžina einamojo katalogo vardą |
| GetDirectories | Gražina *string* masyvą katalogų, esančių nurodytame kataloge |
| GetFiles | Grąžina *string* masyvą failų, esančių nurodytame kataloge |
| GetCreationTime | Grąžina *DateTime* objektą, kada buvo sukurtas katalogas |
| GetLastAccessTime | Grąžina *DateTime* objektą, kada buvo vėliausia prieiga prie katalogo |
| GetLastWriteTime | Grąžina *DateTime* objektą, kada buvo paskutinį kartą buvo rašyta į katalogą |
| Move | Perkelia nurodytą katalogą į nurodytą vietą |
| Klasės **Path** 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**

**Nuoseklinimas** (serialization) – procesas, kai objektų būsenos ar duomenų struktūros išsaugomos laikmenoje tam, kad jas būtų galima pernešti tarp atskirų programų.

Reikalavimai:

* Kiekviena klasė, kurios objektus norime nuoseklinti, turi turėti atributą:

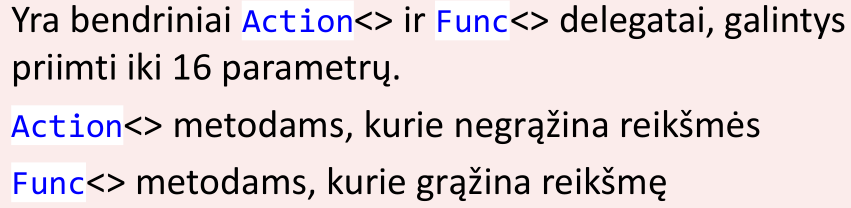


* Arba įgyvendinti sąsają *ISerializable*.

|  |  |  |
| --- | --- | --- |
| **Nuoseklinimo formatai** | | |
| **Dvejetainis** | **XML** | **JSON** |
| Nesudėtingas, greitai apdorojamas | Mažiau efektyvus, daugiau procesoriaus laiko | Žmogus gali skaityti, nesudėtinga apdoroti kompiuteriu |
| Naudojamas objektų turinio pernešimui tarp tų pačių platformų | Atvirasis standartas, tinkamas bet kuriai programai, nepriklauso nuo platformos | Naudojamas duomenų perdavimui tarp asinchroninio  JavaScript ir XML (AJAX) kreipinių |
| Išsaugo formatus | Neišsaugo formatų |  |
| Reikalingi papildomi programos failai | Reikalingi papildomi programos failai | Reikalingi papildomi programos failai |

**Bendriniai delegatai**

[Grįžti prie delegatų](#_Delegatai_ir_įvykiai)



**Anoniminiai metodai**

Tai metodai, kurie neturi vardo, bet gali turėti parametrų sąrašą. Iš karto aprašomi toje vietoje, kur būtų kviečiamas įprastas metodas. Jie reikalingi delegatams, nenorint kurti atskiro metodo.



Anoniminiai metodai ir lokalūs kintamieji:

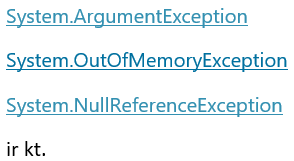
* **Gali** **pasiekti** metodo, kuriame jis yra apibrėžtas, **lokalius kintamuosius**.
* **Gali** **pasiekti** metodo, kuriame yra apibrėžtas, **klasės kintamuosius**.
* **Negali pasiekti** metodo, kuriame jis apibrėžtas, ***ref* ir *out* parametrų**.
* Lokalių kintamųjų vardai **negali** sutapti **su metodo**, kuriame jis apibrėžtas.
* Lokalių kintamųjų vardai **gali** sutapti su metodo, kuriame jis apibrėžtas, **klasės** kintamųjų vardais.

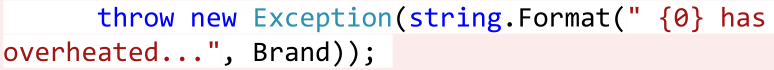
***3. Išimtys.*** *Sisteminės išimtys. Išimčių gaudymo blokas ir jo dalys. Vartotojo išimčių klasės. Išimčių peradresavimas. Išimčių panaudojimas reikšmių priskyrime.*

**Išimtis** – neįprastas ar išskirtinis programos vykdymo atvejis, kuriam reikalingas specialus apdorojimas.

**Pertraukimų (išimčių) apdorojimas** (exception handling) – programavimo kalbų mechanizmas, skirtas aprašyti programos reakciją į vykdymo klaidų atsiradimą ir kitas galimas problemas (vadinamąsias išimtis).

**Sisteminės išimtys**. Jų labai daug, nes vartotojo patogumui jau aprašyti visi galimi atvejai. Jais pasinaudoti galima per SystemException() klasę.



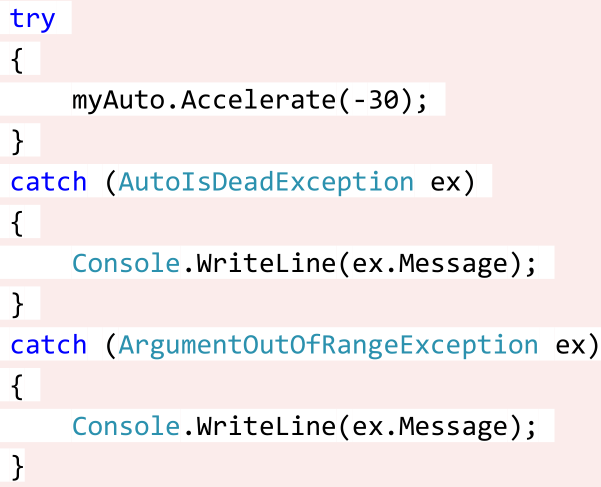
Norint kurti naują atvejį, patartina naudoti bendrąsias išimtis:

|  |  |  |
| --- | --- | --- |
| Kai kurios savybės | | |
| Message | Pranešimas apie klaidą |  |
| HelpLink | Priskiria puslapio adresą, kur daugiau paaiškinimų |  |
| TargetSite | Detalės apie metodą, kuris sukėlė išimtį |  |
| Source | Priskiria objekto, kuris sukėlė išimtį, vardą |  |

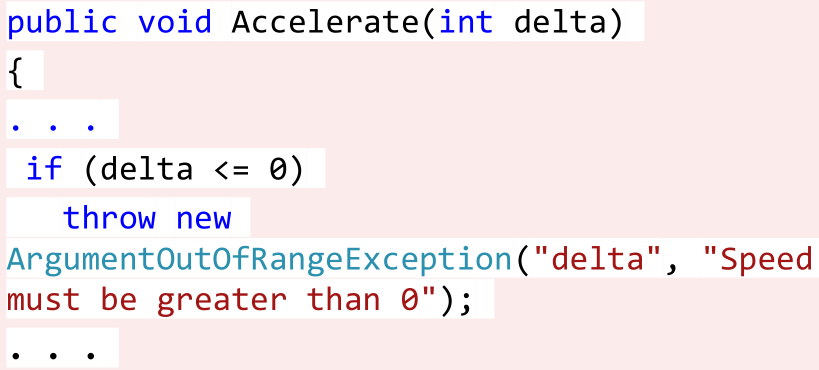
**Išimčių gaudymo blokas ir jo dalys**

**Try**: vykdydama šį bloką, programa aptinka išimtis. Kai pasirodo išimtis, vykdymas nutraukiamas ir einama į *catch* bloką.

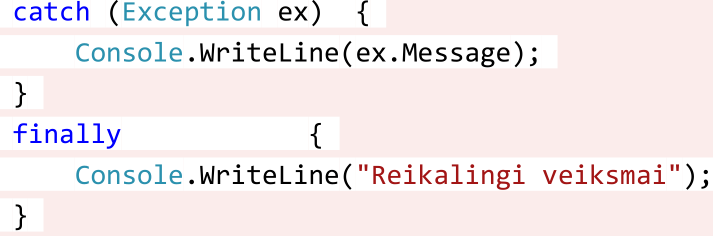
**Catch**: visada rašomas po *try*. Jame esantys sakiniai nusako, kas turi būti atliekama esant išimčiai. Šių blokų turi būti tiek, kiek skirtingų tipų išimčių apibrėžta *throw* sakiniuose.

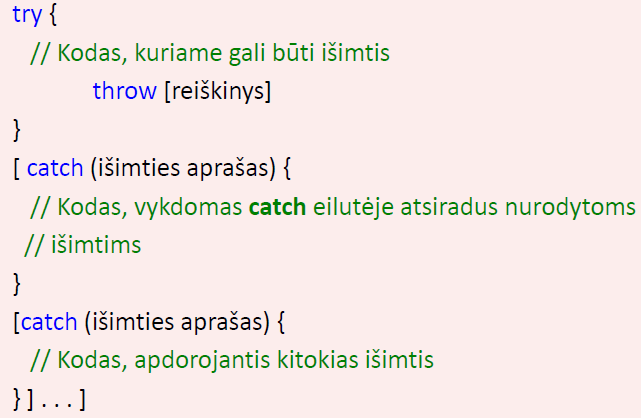


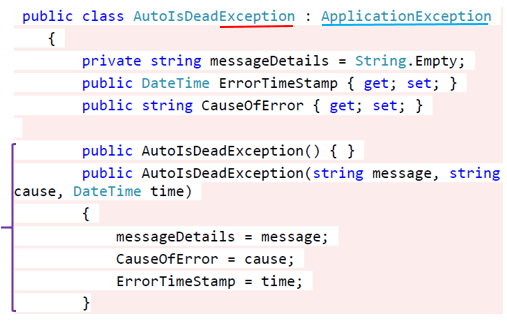
**Throw**: sakinys, kurį vykdant generuojama išimtis ir pereinama į išimties apdorojimo bloką.

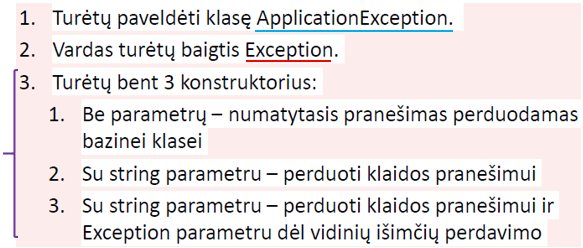


**Finally:** blokas, kuris būtinai bus įvykdytas, nesvarbu, ar *try* blokas nutrauktas su komanda *break*, ar *return*. Šis blokas užtikrina, kad net įvykus klaidai svarbus programinis algoritmas bus įvykdytas.

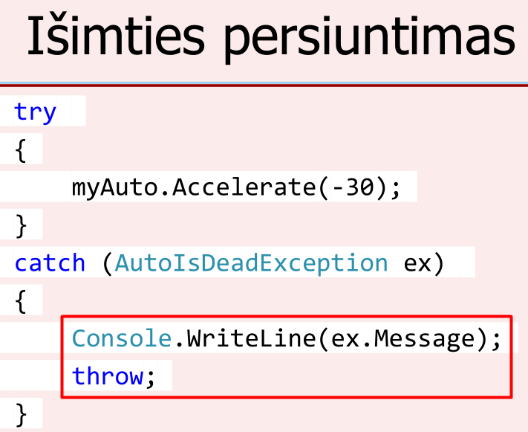




**Vartotojo išimčių klasės**

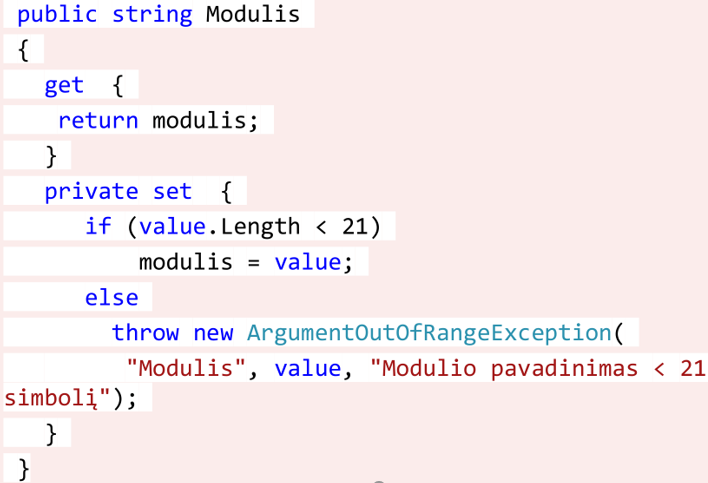
Rekomendacijos kuriant išimčių klasę:

**Išimčių peradresavimas**



**Išimčių panaudojimas reikšmių priskyrime**

Išimtys gali būti vykdomos ir objekto viešuosiuose (get/set – skaitymo/keitimo) metoduose, kurie reikalingi manipuliuoti objekto vidiniais duomenimis. Šitaip galima kontroliuoti, kokios reikšmės suteikiamos objekto elementams.



***4. Kodo kontraktai.*** *Kintamųjų tipų kategorijos. Struktūros. Reikšmė null ir tipai. Diagnostikos klasės. Prieš sąlygos, po sąlygos, teigimai, palyginimas su išimtimis. Asamblėja. Refleksija.*

*Pavyzdinis klausimas: Ką žinote apie atributą Conditional?*

**Kintamųjų tipų kategorijos**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sveikieji skaičiai | int (32 bitų, s) | uint (32, u) | byte (8, u) | sbyte (8, s) |
| long (64, s) | ulong (64, u) | short (16, s) | ushort (16, u) |
| Unsigned/signed – neturintys arba galintys turėti **–** ženklą.  Pvz.: *byte* nuo 0 iki 255, *sbyte* nuo -128 iki 127. | | | | |

|  |  |  |
| --- | --- | --- |
| Slankaus kablelio/realieji skaičiai | Double (64), float (32) | Pvz. 64.32 |
| Loginis | bool | true, false |
| Simbolis | char (16) | Pvz. ‘a’, ‘B’ |
| Simbolių eilutė | string | Pvz. “AbC” |
| Struktūrinis kintamasis – masyvas | string[], double[], … | Pvz. savdienos[0]=”pirmad”; |
| Objektas |  | |

Galima skirti ir taip:

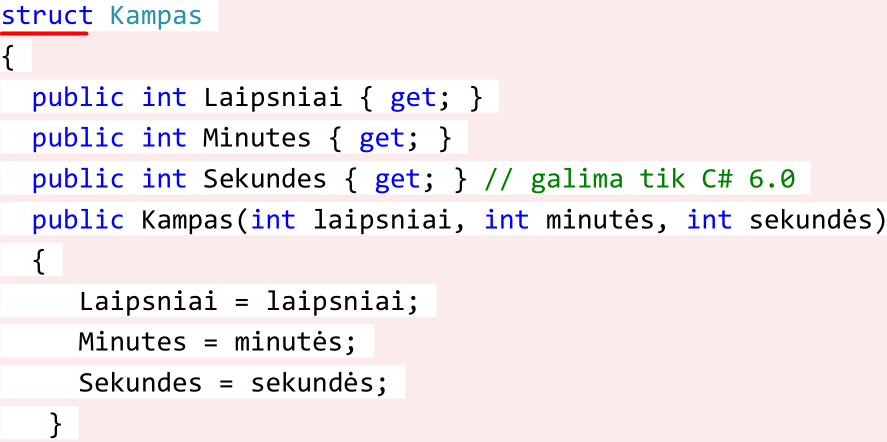
* Tekstiniai/simboliniai tipai – char, string..
* Skaitiniai/aritmetiniai tipai – int, double, long, short…
* Loginis bool.

**Struktūros**

Struktūra – konstrukcija, skirta sudėtingų duomenų tipų kūrimui.

Pvz., kampą galima aprašyti ne tik laipsniais, bet ir minutėmis, sekundėmis.

Konstrukcijos struct pagalba visus šiuos kintamuosius galima susieti viename kintamajame.



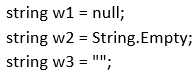
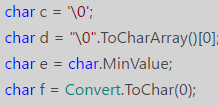
Skirtumai nuo klasės:

* Struktūros – reikšmių tipai, klasės – nuorodų tipai
* Negalimas paveldėjimas
* Negali turėti konstruktoriaus be parametrų
* Negali turėti virtualių narių
* Apibrėžiant konstruktorių, visiems laukams reikia priskirti reikšmes

**Reikšmė null ir tipai**

Kintamieji, kuriems galima priskirti *null* reikšmes

* Skaitiniai tipai negali būti *null*. ~~int sk = null; double sk = null;~~
* Loginiam bool priskyrus *null*, reikšmė bus *false*.
* Simbolių eilutei ir simboliui *null* galima priskirti šiais būdais:

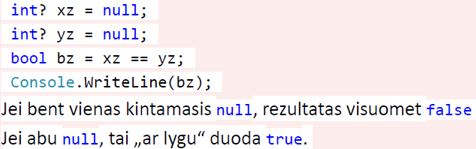


* Bet kokį tipą galima padaryti nunulinamą.

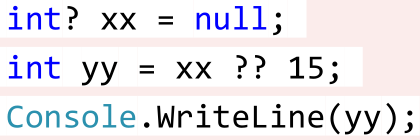


Nunulinamas tipas T?:

* Neturi palyginimo operatorių, todėl juos pasiskolina iš T.



* Visi kiti operatoriai (+, -, \*, |, >>, …..) grąžina null, jei bent vienas operandas null.
* Null suvienijimo operatorius ??. Jei kairiojo operando reikšmė ne null, ją grąžina, priešingu atveju grąžina dešinio operando reikšmę.



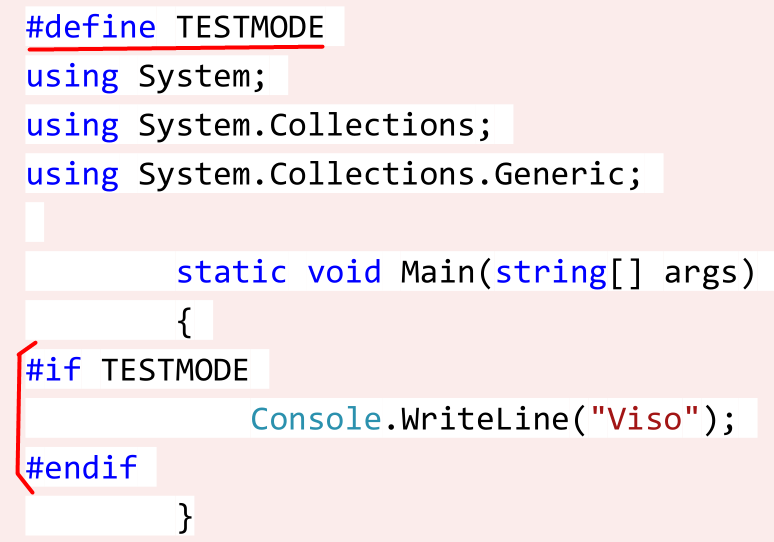
Pritaikymai:

* Programuojant duomenų bazes, nes būna stulpeliuose neužpildytų reiškmių.
* Išvestinėse klasėse, jei leiždiam išvestinei klasei turėti neapibrėžtą savybę, kurią tokiais atvejais galime paimti iš bazinės klasės.
* Kai reikia ,,magiškos’’ reikšmės pranešimui apie nesėkmę, papildomai nenaudojant loginio kintamojo.

**Diagnostikos klasės**

Kai gaunami netinkami rezultatai, reikia diagnostinės informacijos. Tam sudaromi **kodo kontraktai**. Jie leidžia metodams bendrauti per tarpusavio įsipareigojimus. Pažeidus šiuos kontaktus, metodai išduoda diagnostiką.

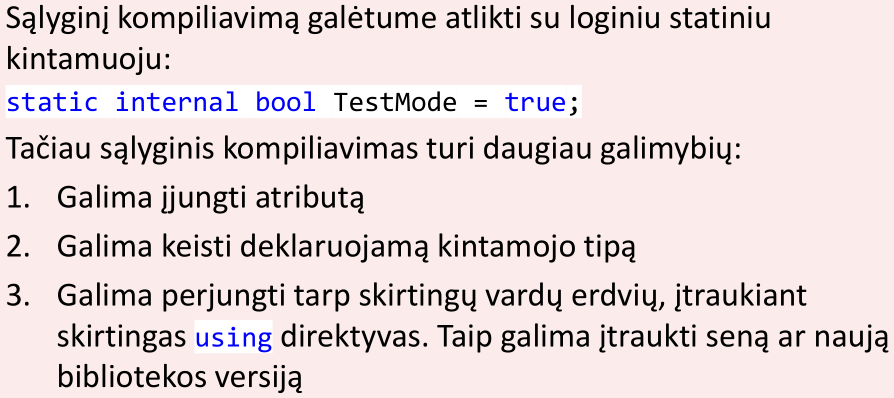
Sąlyginis kompiliavimas

Pašalinus pirmą eilutę, pranešimo nematytume.

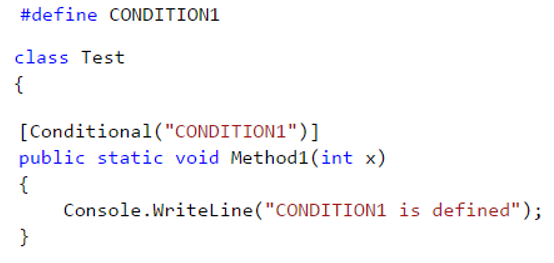
Kompiliatoriaus direktyvomis *#if*, *#else* ar *#endif* galima apibrėžti metodus, kurie vykdomi nusakius direktyvą *#define* ir kompiliavimo raktą (šiuo atveju TESTMODE), rašomą didžiosiomis raidėmis, failo viršuje.

* Sąlygos operatoriuje *#if* galima taikyti logines operacijas &&, ||, !. 
* Jei apibrėžėme asamblėjos lygmenyje, atskiram failui direktyvą galima pašalinti su *#undef*.

Privalumai:



Atributas *Conditional*. Nurodo kompiliatoriui ignoruoti visus metodo kvietimus, jei neapibrėžtas nurodytas simbolis (#undef SIMBOLIS). Priešingu atveju (#define SIMBOLIS) visi metodo kvietimai vykdomi.



Derinimo klasės Debug ir Trace

* Tai statinės klasės, suteikiančios bazines įrašymo ir teigimo galimybes.
* Debug skirta programos derinimo eigai, Trace – ir derinimui, ir eksploatavimui.
* Visi Debug ir Trace metodai apirėžti su





* Abi turi metodus *Write, WriteLine, WriteIf*, kurių pranešimai siunčiami į derintojo langą. Taip pat *Fail*, kuris atidaro dialogo langą su pranešimais *abort, retry, ignore*.
* Abi turi *Listeners* savybę.

Listeners savybė. Turi ,,pėdsakų klausytojų‘‘ (TraceListener) kolekciją, kurie atsakingi už metodų Write, Fail ir Trace apdorojimą:

* Kai sujungta su derintoju, pranešimai siunčiami ten, priešingu atveju – ignoruojami.
* Kai kviečiamas Fail metodas, pasirodo dialogo langas.

Kodo kontraktai

Seka programos klaidas, kai programa paleista, ir užtikrina, kad visi duomenys įeinantys ir paliekantys metodus, yra teisingi. Visą informaciją galima įrašyti į dokumentacijos failus ar sekti vykdymo metu.

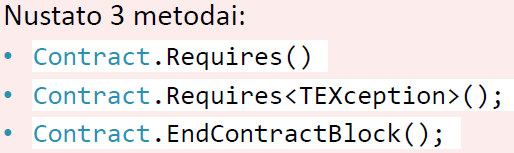
**Prieš ir po sąlygos, teigimai** (preconditions, postconditions, assertions)

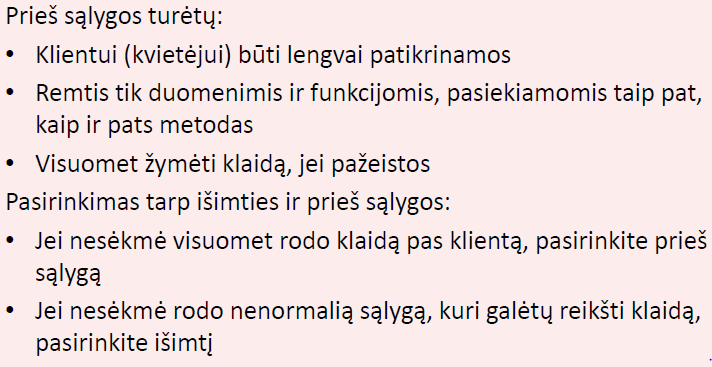
Prieš-sąlygos – reikalavimai, kurie turi būti įgyvendinti prieš pradedant vykdyti metodą.

Po-sąlygos – atvejai, kurie tikėtini baigus vykdyti metodą.

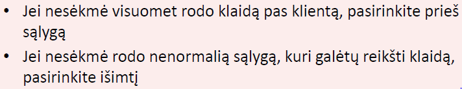
Teigimai – tikrinimai, ar tenkinamos tam tikros sąlygos vykdant kodą. Jei ne – rodoma klaida duomenyse.

Prieš:

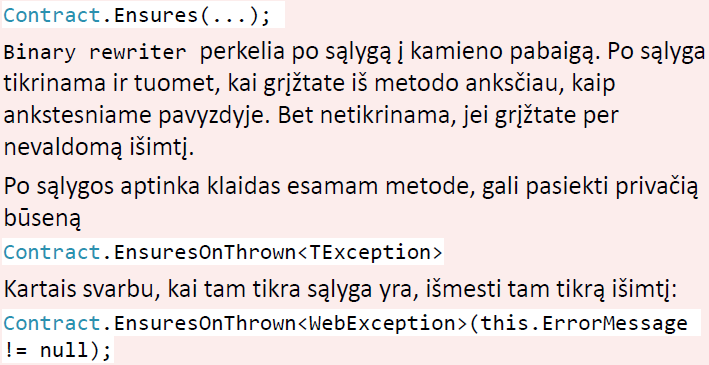




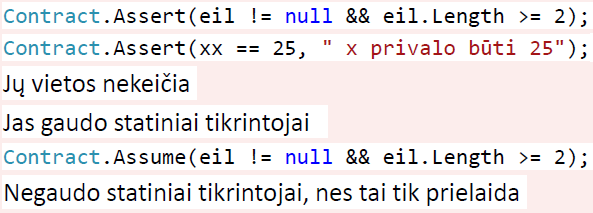
Palyginimas su išimtimis



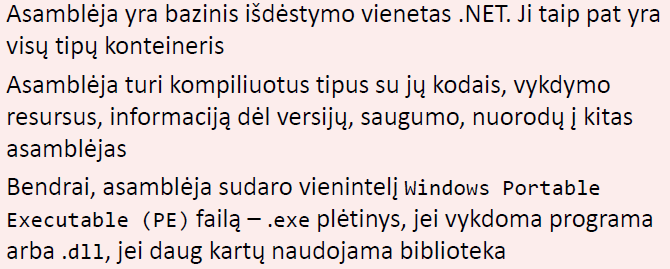
Po:

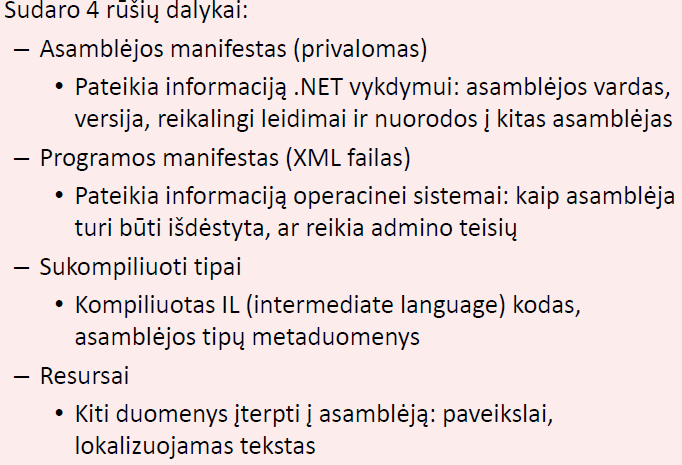


Teigimai:

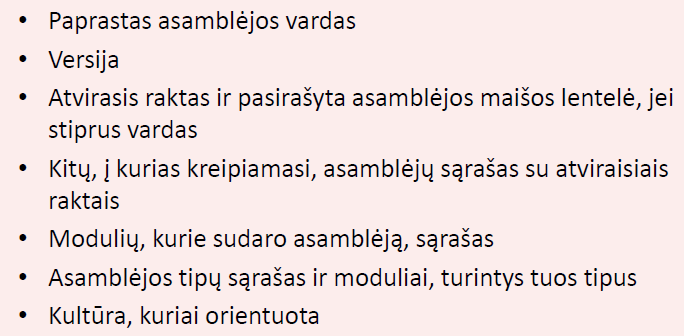


Asamblėja:





Asamblėjos manifesto dalys:



Refleksija:

