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 klasės aprašas, nusakantis, kokius metodus 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.
Gali 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

```
public class Sąrašas: IComparable
{
    public int CompareTo(object obj)
    {
        return 1;
    }
```

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.)

Metodas tampa privačiu – nematomas. Tik taip galima iškviesti:

```
Sąrašas Dr = new Sąrašas();
IComparable Isar = (IComparable)Dr;
Isar.CompareTo(Dr);
```

Standartinių sąsajų pavyzdžiai

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

- Įgyvendinimas

```
public IEnumerator GetEnumerator()
 {
  for (Mazgas dd = pr; dd != null; dd = dd.Kitas)
  {
     yield return dd.Duom;
  }
 }
```

- Panaudojimas

```
foreach (Studentas one in Ae)
{
    file.WriteLine(one);
}
```

- Specializuotas iteratorius

```
public IEnumerable DaugiauUz(int kiek)
 {
   for (Mazgas dd = pr; dd != null; dd =
dd.Kitas)
   {
      if (dd.Duom.kiek > kiek)
          yield return dd.Duom;
   }
 }
```

- Panaudojimas

```
foreach (Studentas one in Ae.DaugiauUz(kiek))
{
    file.WriteLine(one);
}
```

Klonavimas

```
public class Sąrašas: IEnumerable, ICloneable
{
public object Clone()
{
    Sąrašas Cr = new Sąrašas();
    for (Mazgas dd = pr; dd != null; dd = dd.Kitas)
    {
        Cr.DėtiDuomenisT(dd.Duom);
    }
    return Cr;
}
```

```
var Ar = new Sąrašas();
ĮvestiA(CFd, Ar);
Sąrašas Dr = (Sąrašas)Ar.Clone();
```

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.

```
public virtual void Dispose()
   {
       if (!this.disposed)
       {
           try
           {
               Naikinti();
           }
           finally
           {
               this.disposed = true;
               GC.SuppressFinalize(this);
           }
       }
   }
```

```
public void Naikinti()
    {
        while (pr != null)
        {
            Mazgas d = pr;
            pr = pr.Kitas;
            d.Kitas = null;
        }
        pb = pr;
    }
```

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.

```
using (var Ar = new Sąrašas())
{
```

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.

```
class paprasta
{
    public static int Prideti(int xx, int yy)
    {
        return xx + yy;
    }
    public static int Atimti(int xx, int yy)
    {
        return xx - yy;
    }
}
```

```
class Program
{
  public delegate int Dupar(int xx, int yy);

  static void Main(string[] args)
  {
     Dupar deleg = new Dupar(paprasta.Prideti);
     Console.WriteLine(" 5 + 4 = " + deleg(5, 4));
  }
}
```

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*.

```
DirectoryInfo dir = new DirectoryInfo(kelias);
FileInfo info = new FileInfo(CFd1);
```

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:

```
[Serializable]
    public class Studentas :
IComparable<Studentas>, IEquatable<Studentas>
```

- 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

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

`Action`<> metodams, kurie negrąžina reikšmės

`Func`<> metodams, kurie grąžina reikšmę

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.

```
Button myButton = new Button();
myButton .Click +=
delegate
{
    MessageBox.Show("Hello from anonymous method!");
};
```

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.

System.ArgumentException

System.OutOfMemoryException

System.NullReferenceException

ir kt.

```
throw new Exception(string.Format(" {0} has overheated...", Brand));
```

Norint kurti naują atvejį, patartina naudoti bendrąsias išimtis:

Kai kurios savybės		
Message	Pranešimas apie klaidą	`Console.WriteLine(" Pranešimas {0}"`
HelpLink	Priskiria puslapio adresą, kur daugiau paaiškinimų	`AutoIsDeadException ex = new AutoIsDeadException(string.Format(" {0} has overheated...", Brand), "Too much press on accelerator", DateTime.Now);` `ex.HelpLink = "http://proin.ktu.lt";` `throw ex;`
TargetSite	Detalės apie metodą, kuris sukėlė išimtį	`Console.WriteLine(" Metodas {0}", ex.TargetSite)`
Source	Priskiria objekto, kuris sukėlė išimtį, vardą	`Console.WriteLine(" Šaltinis`

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.

```
try
{
    myAuto.Accelerate(-30);
}
catch (AutoIsDeadException ex)
{
    Console.WriteLine(ex.Message);
}
catch (ArgumentOutOfRangeException ex)
{
    Console.WriteLine(ex.Message);
}
```

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

```
public void Accelerate(int delta)
{
. . .
 if (delta <= 0)
   throw new
ArgumentOutOfRangeException("delta", "Speed
must be greater than 0");
. . .
```

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.

```
catch (Exception ex)  {
    Console.WriteLine(ex.Message);
}
finally            {
    Console.WriteLine("Reikalingi veiksmai");
}
```

```
try {
  // Kodas, kuriame gali būti išimtis
        throw [reiškinys]
}
[ catch (išimties aprašas) {
  // Kodas, vykdomas catch eilutėje atsiradus nurodytoms
  // išimtims
}
[catch (išimties aprašas) {
  // Kodas, apdorojantis kitokias išimtis
}]...]
```

Vartotojo išimčių klasės

Rekomendacijos kuriant išimčių klasę:

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

```
public class AutoIsDeadException : ApplicationException
    {
        private string messageDetails = String.Empty;
        public DateTime ErrorTimeStamp { get; set; }
        public string CauseOfError { get; set; }

        public AutoIsDeadException() { }
        public AutoIsDeadException(string message, string
cause, DateTime time)
        {
            messageDetails = message;
            CauseOfError = cause;
            ErrorTimeStamp = time;
        }
```

Išimčių peradresavimas

Išimties persiuntimas

```
try
{
    myAuto.Accelerate(-30);
}
catch (AutoIsDeadException ex)
{
    Console.WriteLine(ex.Message);
    throw;
}
```

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.

```
public string Modulis
{
  get  {
   return modulis;
  }
  private set  {
     if (value.Length < 21)
         modulis = value;
     else
       throw new ArgumentOutOfRangeException(
        "Modulis", value, "Modulio pavadinimas < 21
simbolį");
  }
}
```

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	`public class Person { public string Name { get; private set; } public int Age { get; private set; } public Person(string name, int age) { Name = name; Age = age; } //Other properties, methods, events... }`	`Person p=new Person("abc",15)`

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.

```
struct Kampas
{
  public int Laipsniai { get; }
  public int Minutes { get; }
  public int Sekundes { get; } // galima tik C# 6.0
  public Kampas(int laipsniai, int minutės, int sekundės)
  {
     Laipsniai = laipsniai;
     Minutes = minutės;
     Sekundes = sekundės;
   }
```

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:

```
string w1 = null;
string w2 = String.Empty;
string w3 = "";
```

```
char c = '\0';
char d = "\0".ToCharArray()[0];
char e = char.MinValue;
char f = Convert.ToChar(0);
```

- Bet kokį tipą galima padaryti nunulinamą.

```
int? ii = null;
```

Nunulinamas tipas T?:

- Neturi palyginimo operatorių, todėl juos pasiskolina iš T.

```
int? xz = null;
int? yz = null;
bool bz = xz == yz;
Console.WriteLine(bz);
```

Jei bent vienas kintamasis `null`, rezultatas visuomet `false`

Jei abu `null`, tai „ar lygu“ duoda `true`.

- 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.

```
int? xx = null;
int yy = xx ?? 15;
Console.WriteLine(yy);
```

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 kontraktus metodai išduoda diagnostiką.

Sąlyginis kompiliavimas

```
#define TESTMODE
using System;
using System.Collections;
using System.Collections.Generic;

        static void Main(string[] args)
        {
#if TESTMODE
            Console.WriteLine("Viso");
#endif
        }
```

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 &&, ||, !. `#if TESTMODE && !PLAYMODE`
- Jei apibrėžėme asamblėjos lygmenyje, atskiram failui direktyvą galima pašalinti su *#undef*.

Privalumai:

Sąlyginį kompiliavimą galėtume atlikti su loginiu statiniu kintamuoju:

```
static internal bool TestMode = true;
```

Tačiau sąlyginis kompiliavimas turi daugiau galimybių:

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

```
#define CONDITION1

class Test
{

[Conditional("CONDITION1")]
public static void Method1(int x)
{
    Console.WriteLine("CONDITION1 is defined");
}
```

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

```
[Conditional("DEBUG")]
[Conditional("TRACE")]
```

- 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:

Nustato 3 metodai:

- `Contract.Requires()`
- `Contract.Requires<TEXception>();`
- `Contract.EndContractBlock();`

Prieš sąlygos turėtų:

- Klientui (kvietėjui) būti lengvai patikrinamos
- Remtis tik duomenimis ir funkcijomis, pasiekiamomis taip pat, kaip ir pats metodas
- Visuomet žymėti klaidą, jei pažeistos

Palyginimas su išimtimis

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

Po: tikrinama net kai iš metodo grįžtama anksčiau, bet ne tada, kai per nevaldomą išimtį.

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

Sudaro 4 rūšių dalykai:

- Asamblėjos manifestas (privalomas)
  - Pateikia informaciją .NET vykdymui: asamblėjos vardas, versija, reikalingi leidimai ir nuorodos į kitas asamblėjas
- Programos manifestas (XML failas)
  - Pateikia informaciją operacinei sistemai: kaip asamblėja turi būti išdėstyta, ar reikia admino teisių
- Sukompiliuoti tipai
  - Kompiliuotas IL (intermediate language) kodas, asamblėjos tipų metaduomenys
- Resursai
  - Kiti duomenys įterpti į asamblėją: paveikslai, lokalizuojamas tekstas

Asamblėjos manifesto dalys:

- Paprastas asamblėjos vardas
- Versija
- Atvirasis raktas ir pasirašyta asamblėjos maišos lentelė, jei stiprus vardas
- Kitų, į kurias kreipiamasi, asamblėjų sąrašas su atviraisiais raktais
- Modulių, kurie sudaro asamblėją, sąrašas
- Asamblėjos tipų sąrašas ir moduliai, turintys tuos tipus
- Kultūra, kuriai orientuota

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