



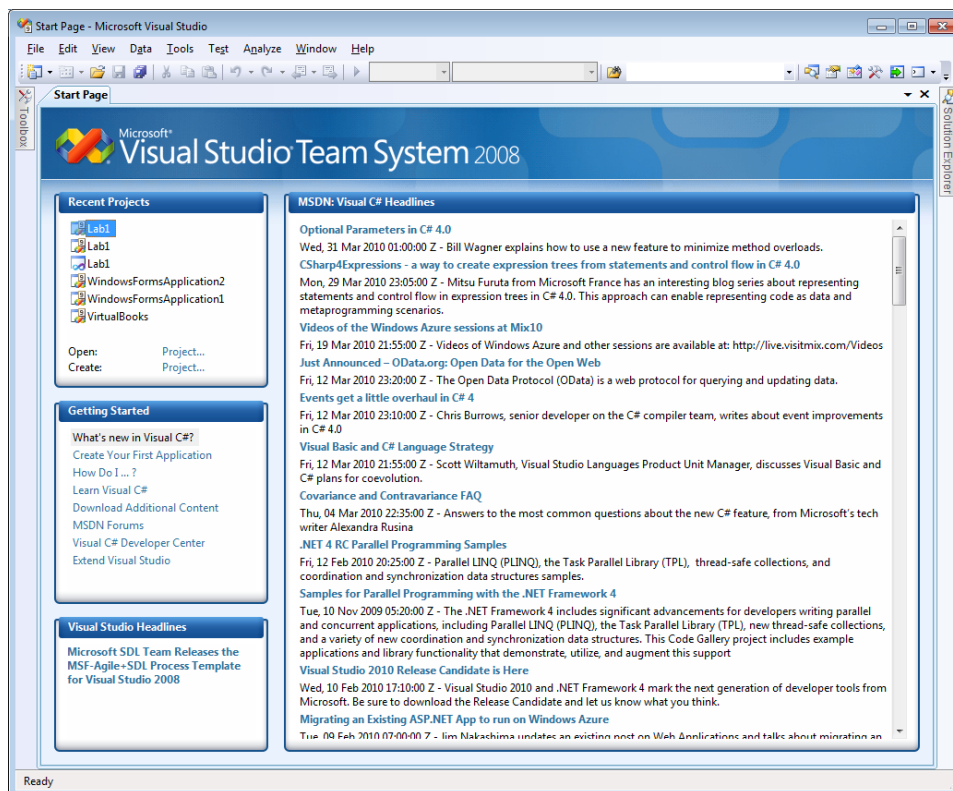
Šī laboratorijas darba mērķis ir iepazīties ar integrētās izstrādes vides *Microsoft Visual Studio 2008* (turpmāk tekstā – izstrādes vide) komponentiem un tās būtiskākām iespējām lietojumprogrammu izstrādei.

Izstrādes vidē tiek nodrošināts atbalsts lietojumprogrammu izstrādei programmatūras realizācijas, testēšanas un uzturēšanas posmos. Standarta instalācijas paketē tiek nodrošināts atbalsts tādām programmēšanas valodām kā C#, C/C++ un Visual Basic.NET, bet, ar trešo pušu papildinājumiem, var pieslēgt iespēju izstrādāt lietojumprogrammas arī citās programmēšanas valodās.

Ja izstrādes vide tiek atvērta pirmo reizi, tiek piedāvāts izvēlēties vēlamo programmēšanas valodu. No šīs izvēles ir atkarīgi izstrādes vides piedāvātie uzstādījumi un klaviatūras saīsnas. Šajā laboratorijas darbu krājumā norādot saīsinājumaustiņus tiek pieņemts, ka ir izvēlēta **Visual C# 2005** programmēšanas valoda.

*Vēlāk šo izvēli var nomainīt izstrādes vides uzstādījumos, no galvenās izvēlnes izvēloties punktu **Tools / Options...**, un tās dialoga **Environment / Keyboard** kombinētajā lodziņā **Apply the following additional keyboard mapping scheme**: izvēloties nepieciešamo programmēšanas valodu, ar kuru paredzēts strādāt.*

Pēc programmēšanas valodas izvēles, vai arī atverot izstrādes vidi nākamās reizes, tiek parādīts izstrādes vides galvenais logs, ar atvērtu sākuma lapu (*Start Page*), kura realizē sava veida atskaites punktu turpmākam darbam:



Izstrādes vides loga augšdaļā atrodas izvēlnes un rīku joslas, bet apakšā – statusa josla. Atlikušo laukumu starp rīku joslu un statusa joslu sauc par darba zonu. Tajā, tiek atspoguļots rediģējamais artefakts (par artefaktu pieņemsim jebkuru atsevišķi no citiem izdalītu vienumu – teksta failu, dokumentu, pirmkoda failu, klasi, diagrammu, attēlu, tīmekļa lapu u.c.) un, atkarībā no resursa, ar to saistītie papildus logi un paneļi specifiskas informācijas atspoguļošanai vai piedāvāšanai.

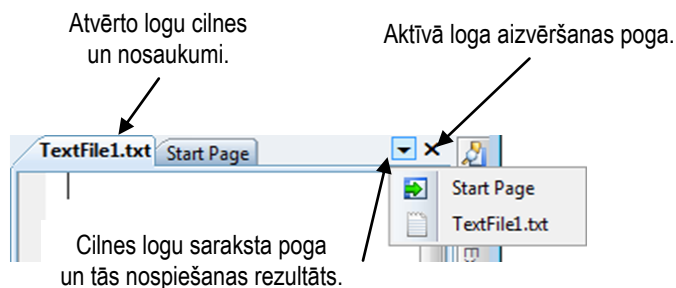
Sākuma lapa piedāvā šādas sekcijas:

- **Recent Projects** – pēdējo atvērto projektu saraksts, kas ļauj ātri atkārtoti atvērt (projekts ir strukturēts failu kopums, kas satur visus nepieciešamos artefaktus lietojumprogrammas izstrādei); atvērt citus projektus (**Open: Project...**) vai arī izveidot jaunu projektu (**Create: Project...**);
- **Getting Started** – saites uz dokumentācijas sadaļām un/vai tīmekļa vietnēm, kurās aprakstīta pamatinformācija par izstrādes vidi, tās iespējām un apgūšanas soļiem;
- **Visual Studio Headlines** – aktuāla informācija par izstrādes vides papildinājumiem un tehniska rakstu blogiem;
- **MSDN: Visual C# Headlines** – saites uz tīmekļa vietnēm par izvēlētās programmēšanas valodas jaunumiem no dažādiem avotiem.

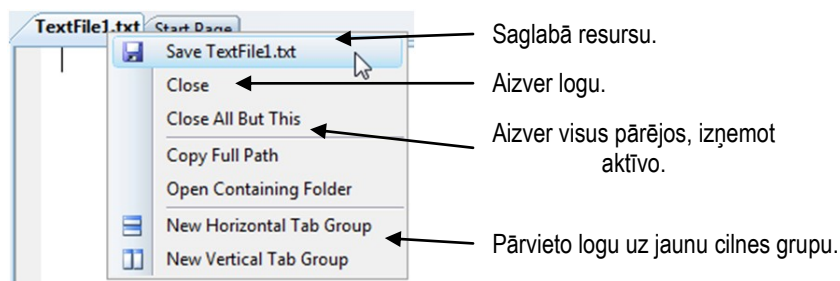
1.1 DARBA ZONAS LOGU PĀRVALDĪBA

Šajā sadaļā tiek apskatīti izstrādes vides darba zonas logu un paneļu organizācijas un pielāgošanas pamati. Visiem darba zonā esošajiem logiem un paneļiem ir noteiktas īpašības, kas raksturo to veidu, stāvokli un novietojumu:

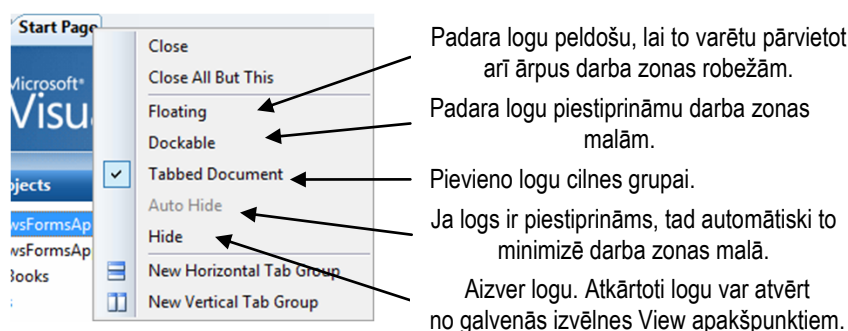
- jebkurš darba zonā atvērts logs var būt vienā no stāvokļiem: cilnes grupā (**Tabbed Document**), peldošs (**Floating**) vai piestiprināms (**Dockable**) kādai no darba zonas malām;
- loga virspusē ir informācijas josla, kuras kreisajā stūrī tiek atspoguļots loga nosaukums, bet labajā – pogas, kas ļauj manipulēt ar logu un tā īpašībām:



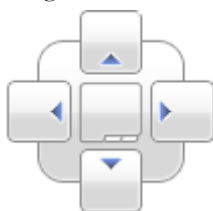
- uz loga nosaukuma var atvērt tā konteksta izvēlni, no kuras var mainīt tā atspoguļošanas un novietošanas īpašības:




- katra loga konteksta izvēlne ir atkarīga no loga veida, stāvokļa un satura. Piemēram, sākuma loga konteksta izvēlnē ir pieejami šādi punkti:



- tikai peldošam logam var mainīt abu tā malu izmērus – piestiprinātiem logiem tiek ierobežota tikai horizontālās vai vertikālās malas izmēra maiņa, bet, ja logs atrodas cilnes grupā, tad tā malas nosaka cilnes grupas izmēri;
- peldošu logu var piestiprināt pie jebkuras darba zonas malas, vai arī ievietot cilnes grupā – lai to izdarītu, vispirms jāpadara logu par piestiprināmu (**Dockable**) un pēc tam, nospiežot peli uz loga nosaukuma, to jāuzvelk uz vienas no atbilstošām piktogrammām:



Peli atlaižot, logs tiek piestiprināts izvēlētai darba zonas malai, vai arī ievietots cilnes logu grupā.

- ja piestiprināmam logam ir iestādīta automātiskas minimizēšanas pazīme (konteksta izvēlnes punkts **Auto Hide**, vai arī pa kreisi no loga aizvēšanas pogas ir poga , tad logs tiek rādīts tikai tad, ja noklikšķina uz tā nosaukuma, vai kamēr tas ir fokusā. Tīklīdz tiek izvēlēts darba zonas apgabals, kas ir ārpus loga robežām, logs automātiski minimizējas, vienlaicīgi palielinot citiem logiem pieejamo darba zonu.

*Ja logu izskārtojums nav skaidrs vai arī nav iespējams atvērt nepieciešamo logu, ar galvenās izvēlnes komandu **Window | Reset Window Layout** var atiestatīt logu konfigurāciju uz tādu, kāda tā ir paredzēta pēc noklusēšanas.*

1.2 PALĪDZĪBAS SISTĒMA UN TĀS LIETOŠANA

Modernās izstrādes vides un to piedāvātās iespējas ir ļoti plašas, lai spētu pilnībā tās apgūt un lietot „pēc atmiņas”. Tāpēc ļoti būtiska loma ir izstrādes vidē piedāvātajai palīdzības dokumentācijai, un spējai izmantot tās iespējas, lai ātri atrastu nepieciešamo risinājumu.

Šim nolūkam izstrādes vide piedāvā dokumentācijas pārlūku (**Document Explorer**), kura izsaukšana notiek no galvenās izvēlnes **Help** apakšpunktiem. Palīdzības informācija ir apkopota dokumentu kolekcijās, kuras tiek uzinstalētas atkarībā no izvēlētajā izstrādes vides nosacījumiem un papildinājumiem:

- lai iepazītos ar dokumentācijas kolekciju pēc tās satura rādītāja, jāizmanto galvenās izvēlnes punkts **Help | Contents**;

- lai meklētu nepieciešamo informāciju pēc atslēgvārda dokumentācijas indeksā, jāizmanto galvenās izvēlnes punkts **Help | Index**;
- lai meklētu informāciju pēc patvaļīgas teksta frāzes visos dokumentācijas tekstos, jāizmanto galvenās izvēlnes punkts **Help | Search**;
- lai atrastu kāda specifiska uzdevuma risināšanas soļus, jāizmanto galvenās izvēlnes punkts **Help | How Do I**, kas strukturē dokumentāciju pēc uzdevumu kategorijām;
- lai atkārtoti izmantotu vai saglabātu vēlākai izmantošanai saites uz noteiktiem palīdzības dokumentiem vai to meklēšanas nosacījumus, jāizmanto galvenās izvēlnes punkts **Help | Help Favorites**;
- lai darba zonā atvērtu intelektuālu palīdzības sistēmas logu (**Dynamic Help**), jāizvēlas galvenās izvēlnes punkts **Help | Dynamic Help**. Šajā logā dinamiski mainīsies saites uz palīdzības dokumentāciju, atkarībā no tā, kur atrodas kursora pozīcija pirmkoda redaktorā vai citā aktīvā darba zonas elementā.

Izstrādes vidē ir iespējams norādīt, vai dokumentācijas avots tiks tiešsaistē lejupielādēts vai arī tiks izmantota lokāli uz datora uzinstalēta versija. Tiešsaistes versijas priekšrocība ir tā, ka tā tiek regulāri aktualizēta, savukārt lokālai versijai nav nepieciešams interneta pieslēgums un informācijas izgušana notiek ātrāk. Šo izvēli norāda no galvenās izvēlnes atverot konfigurācijas dialogu **Tools | Options...** un dialoga **Environment | Help | Online** zarā izvēloties starp izvēles rūtiņām zem **When loading Help Content**.

Vispārīgā gadījumā, ja nepieciešams iegūt specifisku palīdzības informāciju, tad aktivizē vajadzīgo darba zonas elementu vai pirmkoda redaktorā kursoru novieto uz vajadzīgā vārda, un spiež taustiņu **F1** – palīdzības sistēma automātiski mēģina noskaidrot kontekstu un piedāvā būtiskākās saites uz palīdzības sistēmas dokumentiem, kuros meklējamais risinājums varētu būt atrodams.

1.3 PROJEKTI UN RISINĀJUMI

Izstrādes vidē darbs pie jebkuras lietojumprogrammas izstrādes sākas ar tās projekta atvēršanu. Atkarībā no tā, vai tiek veidots jauns projekts, vai atvērts esošais:

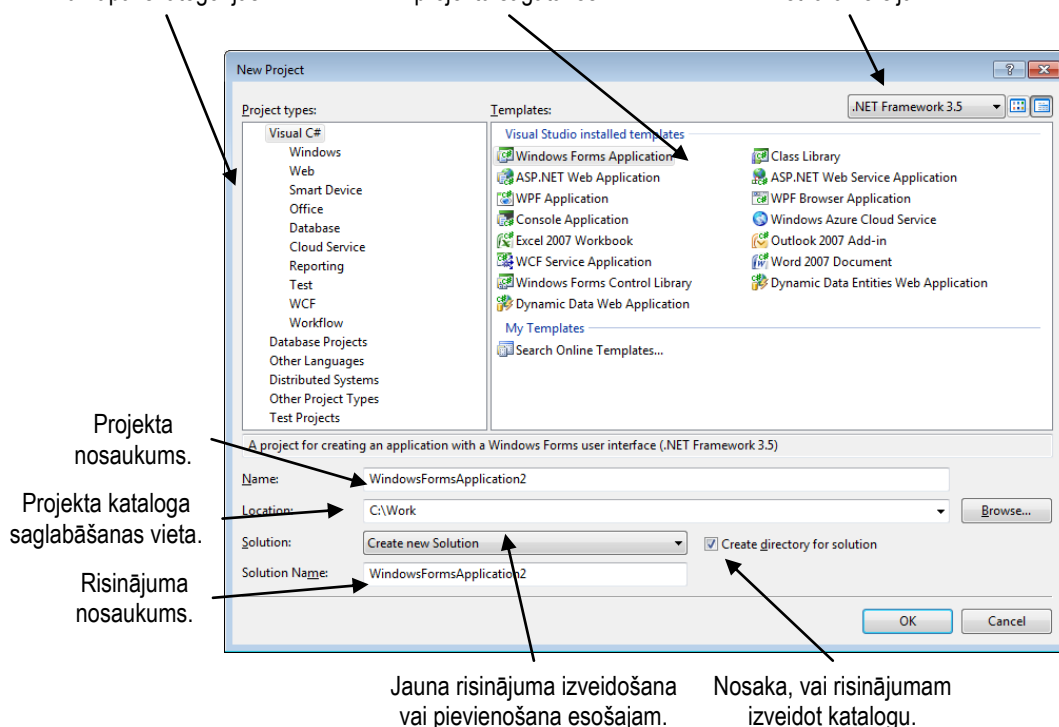
- jauna projekta izveidošanu sāk vai nu ar sākuma lapas sekcijas **Recent Projects** saiti **Create: Project...**, vai arī no galvenās izvēlnes izvēloties **File | New | Project...**, vai arī nospiežot taustiņu kombināciju **Ctrl+Shift+N**;
- esoša projekta atvēršanu veic vai nu no sākuma lapas sekcijas **Recent Projects** izvēloties piedāvātos nesen atvērtos projektus, vai arī izvēloties **Open: Project...**, vai arī no galvenās izvēlnes izvēloties **File | Open | Project/Solution...**, vai arī izvēloties no **File | Recent Projects**, vai arī nospiežot taustiņu kombināciju **Ctrl+Shift+O**.

Izvēloties jauna projekta izveidošanu, tiek atvērts jauna projekta parametru dialogs:

Iespējamās projektu kategorijas un apakškategorijas.

Izvēlētajā kategorijā pieejamās projektu sagataves.

Projekta mērķa .NET ietvara versija.



Tā kā izstrādes vide ir nokonfigurēta ar C# kā primāro programmēšanas valodu, tad pēc noklusēšanas jauna projekta izveidošanā tiek piedāvāti tieši C# projektu veidi. Pretējā gadījumā ir jāizvēlas atbilstoša projektu kategorija vai tās apakškategorija.

Mainot projekta mērķa platformas versiju, automātiski mainās piedāvātās projektu sagataves – parasti jo mazāka .NET ietvara versija izvēlēta, jo mazāks skaits projektu sagatavju tiek piedāvāts. Šo laboratorijas darbu vajadzībām būs nepieciešama vismaz .NET ietvara 2.0 versija.

Projekta nosaukumu izvēlas tādu, lai tas atspoguļotu projekta kompilēšanas rezultātā iegūstamo artefaktu (t.i. izpildāmā faila, bibliotēkas, tīmekļa lietojuma un tml. nosaukumu).

Jebkurš izstrādes vides projekts vienmēr tiek ievietots risinājumā (*solution*). Risinājums ļauj loģiski apvienot vienas lietojumprogrammas saistītas daļas, katra no kurām tiek realizēta projekts. Tāpēc pie jauna projekta parametru dialogā jānorāda, vai izveidot arī jaunu risinājumu, vai arī projektu ievietot jau izstrādes vidē atvērtā risinājumā.

Ieteicams risinājumam izveidot savu katalogu, kura atsevišķos apakškatalogos glabāt tā projektus.

1.4 VIENKĀRŠAS LIETOJUMPROGRAMMAS IZSTRĀDE

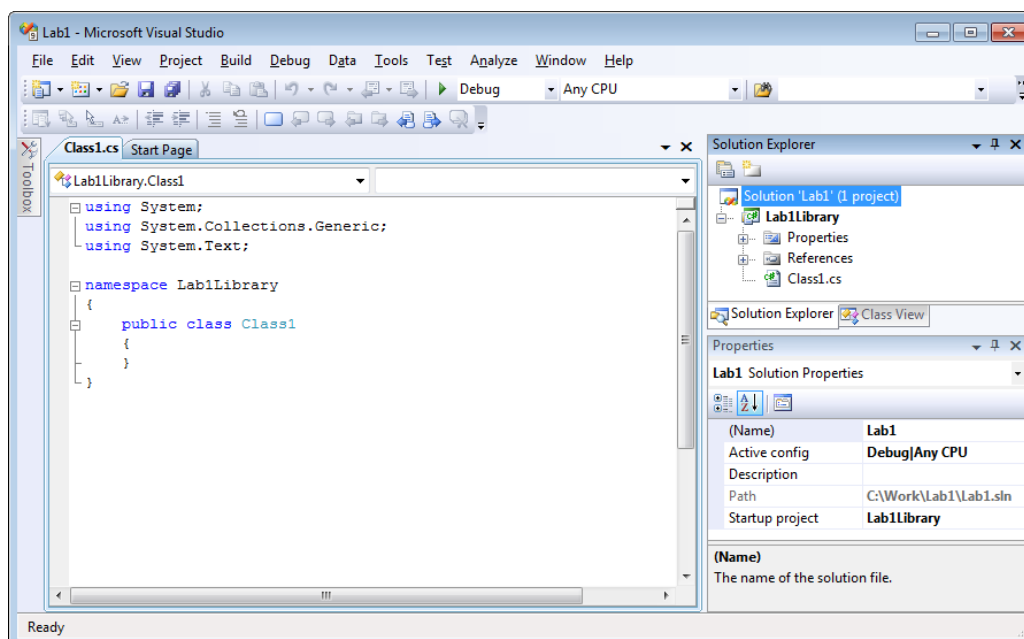
Lai nodemonstrētu izstrādes vides iespējas, turpmāk tiks pa soļiem parādīta vienkāršas lietojumprogrammas izstrāde, kuras mērķis ir uzglabāt informāciju par studentiem saraksta veidā, atspoguļot to uz ekrāna, kā arī saglabāt studentu sarakstu failā un ielādēt to no faila.

Uzdevums:

Izstrādes vidē izveidot jaunu projektu ar šādiem parametriem:

- Projekta kategorija: **Visual C#**
- Projekta sagatave: **Class Library**
- Projekta nosaukums: **Lab1Library**;
- Projekta kataloga saglabāšanas vieta: **C:\Work**;
- Norādiet jauna risinājuma ar nosaukumu **Lab1** izveidošanu.

Apstipriniet šos nosacījumus nospiežot pogu **OK**. Tiks ielādēta jaunā projekta sagatave:






Redzams, ka cilnes logu grupā, papildus sākuma lapai ir ielādēts arī projekta pirmkoda fails **Class1.cs**, kura vārdu telpā **Lab1Library** ir definēta C# klases **Class1** sagatave.

1.5 RISINĀJUMA PĀRLŪKA UN ĪPAŠĪBU LOGI

Atvērtā projekta darba zonas labajā pusē ir piestiprināts risinājuma pārlūka (**Solution Explorer**) logs, bet zem tā – īpašību (**Properties**) logs. Ja šie logi ir minimizēti, tad darba zonas malā būs redzami tikai to nosaukumi, uz kuriem uzklikšķinot tie tiks parādīti. Ja kāds no šiem logiem nav redzams, tos var atvērt ar atbilstošiem galvenās izvēlnes punktiem, attiecīgi **View | Solution Explorer** un **View | Properties Window**.

Risinājuma pārlūks atspoguļo atvērtā risinājuma un tajā esošo projektu hierarhisku struktūru, kura saknē ir pats risinājums **Lab1**. Aktivizējot kādu no risinājuma pārlūka hierarhijas virsotnēm, īpašību logā tiek parādītas izvēlētās virsotnes īpašības, kuras ir iespējams (ja tās ir aktīvas) modificēt.

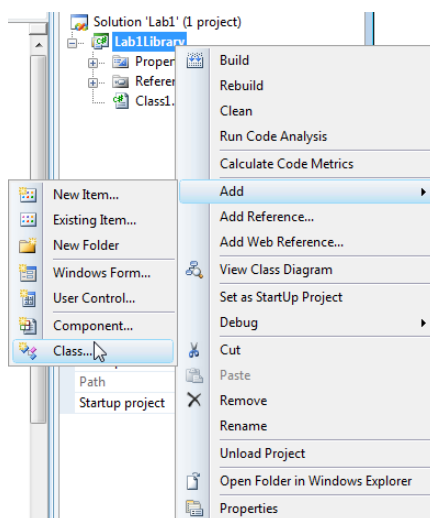
Risinājuma pārlūka hierarhijā zem risinājuma nosaukuma virsotnes atrodas projektu nosaukumu līmenis. Pašlaik tajā ir tikai viens projekts **Lab1Library**, zem kura, savukārt ir papildus apakšzari.

Ja pa kreisi no hierarhijas virsotnes ir piktogramma , tad tas nozīmē, ka šai virsotnei ir pieejami papildus apakšzari, kurus var atvērt, nospiežot uz . Ja zars ir atvērts, tad tam uz virsotnes ir piktogramma .

Tipiska C# projekta sagatavi veido šādas virsotnes:

- **Properties** – šīs virsotnes apakšzarā tiek glabāti dažādi parametri, kas attiecas uz projektu kopumā. Ar peles dubultklikšķi uzklikšķinot uz šīs virsotnes nosaukuma, cilnes logu grupā tiek atvērts logs, kurā iespējams pielāgot visa projekta uzstādījumus;
- **References** – šīs virsotnes apakšzarā uzskaitītas atsauces uz ārējām bibliotēkām, kuras nepieciešamas projekta gala artefakta veiksmīgai izveidošanai;
- projekta C# pirmkoda faili (ar paplašinājumu **.cs**), kuros tiek definēta lietojumprogrammas loģika klašu, tās atribūtu un metožu veidā. Loģiski saistīti pirmkoda faili var tikt apvienoti apakšzaros ar kopīgu virsotni.

Katrai risinājuma pārlūka virsotnei ir pieejama konteksta izvēlne ar komandām, kuras šai virsotnei var piemērot. Piemēram, projekta nosaukuma virsotnei ir šāda konteksta izvēlne:

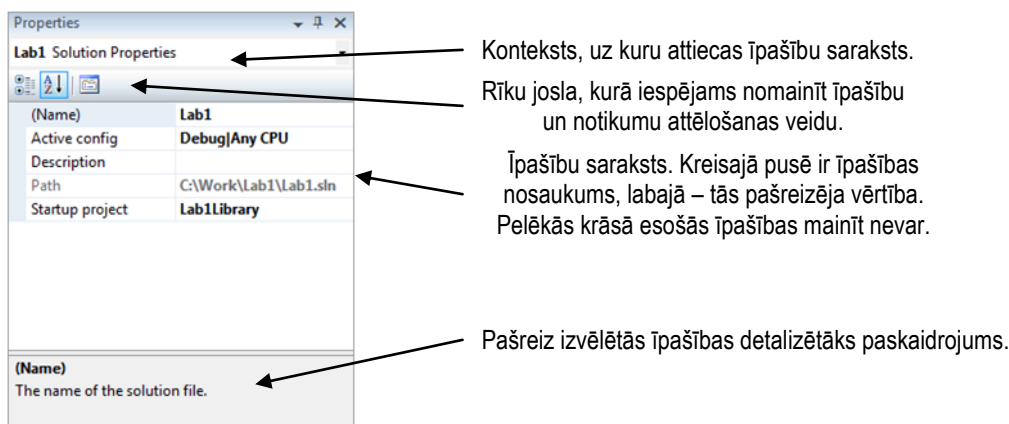


Tās būtiskākie punkti ir šādi:

- **Build** – uzsākt projektā izmainīto artefaktu kompilēšanu (šo punktu var aktivizēt arī ar taustiņu kombināciju **Shift+F6**);
- **Rebuild** – pilnībā pārkompilēt projekta artefaktus;
- **Clean** – izdzēst kompilēšanas procesā radušos artefaktus;
- **Add** – pievienot projektam jaunu vai eksistējošu artefaktu. Šī punkta apakšpunkti piedāvā izvēlēties detalizētāku artefakta tipu;
- **Add Reference...** – pievienot projektam atsauci uz tam nepieciešamo bibliotēku;
- **View Class Diagram** – izveidot vai atvērt projekta klašu diagrammu. Klašu diagramma ir ērti izmantojama, lai gūtu priekšstatu par projektā definēto klašu deklarācijām un savstarpējām relācijām.
- **Set as StartUp Project** – uzstādīt kā sāknēšanas projektu. Kad tiks aktivizēta risinājuma palaišana uz izpildi, tad tiks palaists tieši sāknēšanas projekts;
- **Debug | Start New Instance** – palaist projektu uz izpildi atklādošanas režīmā (sāknēšanas projektam ir pieejama klaviatūras saīssne **F5**);
- **Remove** – izņemt projektu no risinājuma (failu sistēmā projekta faili netiks dzēsti);
- **Rename** – pārsaukt projektu;

- **Unload Project** – izlādēt projektu no izstrādes vides, vienlaicīgi aizverot visus tā atvērtos artefaktus. Atkārtoti projektu ielādē, tā konteksta izvēlnē izvēloties punktu **Reload Project**;
- **Open Folder in Windows Explorer** – atvērt projekta mapi Windows failu pārlūkā;
- **Properties** – cilnes logu grupā atvērt logu ar projekta uzstādījumiem.

Savukārt īpašību logs sastāv no šādiem komponentiem:

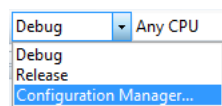


Īpašību skaits, nosaukumi un iespējamās vērtības katram artefakta tipam ir savi. Ja turpmāk šajā laboratorijas darbu krājumā tiks prasīts nomainīt kāda artefakta īpašības vērtību, tad šis artefakts ir vispirms jāaktivē (t.i. jāaktivē tā konteksts īpašību logā), pēc tam īpašību sarakstā jāatrod prasītās īpašības nosaukums, un jānomaina tās vērtība, izvēli apstiprinot ar taustiņu **Enter**.

1.6 RISINĀJUMA KONFIGURĀCIJAS UN RISINĀJUMA KOMPILĒŠANA

Paralēli iepriekš aplūkotajai projektu struktūrai, katram risinājumam tiek definētas vismaz divas kompilēšanas konfigurācijas – **Debug** un **Release**. Pirmā ir paredzēta lietojumprogrammas izstrādes posmā, jo šajā konfigurācijā lietojumprogrammas gala artefaktos tiek iekļauta papildus atklūdošanas informācija, kas izstrādātājam ļauj veikt lietojumprogrammas vai tās daļu atklūdošanu. Savukārt otrā konfigurācija ir paredzēta gala artefaktu iegūšanai, kurus izplata lietojumprogrammas galalietotājiem – tajā netiek iekļauta papildus atklūdošanas informācija un kompilators pirmkodu papildus optimizē, kā rezultātā lietojumprogramma strādā ātrāk un gala artefakti var būt ar mazāku faila izmēru. Katras konfigurācijas parametrus, ja nepieciešams, var pielāgot, kā arī var veidot jaunas kompilēšanas konfigurācijas.

Pēc noklusēšanas, jaunam projektam tiek aktivizēta konfigurācija **Debug**. Pārslēgties starp konfigurācijām var no galvenās izvēlnes punkta **Build | Configuration Manager...**, dialoga kombinētajā sarakstā **Active solution configuration**: norādot nepieciešamo konfigurāciju. To pašu var veikt arī no analogiska rīku joslā pieejamā saraksta:



Risinājuma kompilēšanas procesu ar aktīvo konfigurāciju uzsāk ar izvēlnes komandu **Build | Build Solution** vai arī nospiežot taustiņu **F6**. Veiksmīgas kompilēšanas beigās izstrādes vides statusa joslā tiek izvadīts teksts **Build succeeded**.

Uzdevums:

Veiciet risinājuma kompilēšanu vispirms ar konfigurāciju **Debug**, pēc tam ar konfigurāciju **Release** un pārliecinieties, ka tās rezultāts ir veiksmīgs.

Nobeigumā par aktīvo uzstādiet konfigurāciju **Debug**.

1.7 PROJEKTA PAPILDINĀŠANA AR JAUNIEM ARTEFAKTIEM

Izstrādes vides pirmkoda redaktors piedāvā plašu refaktoringa funkcionalitāti. Ar refaktoringu saprot pirmkoda pārstrukturēšanu, lai tas kļūdu optimālāks, funkcionālāks, vai vienkārši saprotamāks. Refaktoringa komandas atrodas galvenās izvēlnes **Refactor** apakšpunktos, kā arī pirmkoda redaktora konteksta izvēlnē. Tā kā refaktoringa lielākoties attiecas uz noteiktu pirmkoda fragmentu, tad pirms tā veikšanas nepieciešams vai nu kursoru novietot uz atbilstošā vārda pirmkoda redaktorā, vai arī iezīmēt noteiktu pirmkoda fragmentu.

Uzdevums:

1. Pielietojiet vienu no vienkāršākām refaktoringa komandām – pārsaukšanu, lai atvērtajā projektā esošo klasi **Class1** pārsauktu par klasi **Student**:
 - Pārliecinieties, ka aktīvs ir klases **Class1** pirmkoda faila **Class1.cs** logs;
 - Iezīmējiet klases **Class1** nosaukumu, atveriet uz tās konteksta izvēlni, un izvēlieties punktu **Refactor | Rename...**;
 - Dialogā norādiet jauno nosaukumu – **Student**;
 - Nospiežot pogu **OK**, ja ir aktīva izvēles rūtiņa **Preview reference changes**, tad pirms izmaiņu veikšanas, ir iespēja aplūkot visas vietas uz kurām izmaiņa attieksies (**Preview Changes**) – gan pašas klases deklarācijā, gan tās objektu definīcijās. Papildus ir iespēja atcelt izmaiņu veikšanu konkrētām vietām, izmaiņu sarakstā atslēdzot atbilstošo izvēles rūtiņu.
 - Tā kā labā prakse prasa, lai faila vārds atspoguļotu tajā esošo klasi, pārsauciet arī failu kurā klase **Student** ir definēta. Šim nolūkam risinājuma pārūkā izvēlieties faila **Class1.cs** virsotni un īpašību logā nomainiet tā īpašības **File Name** vērtību uz **Student.cs**.
2. Papildiniet klases **Student** definīciju ar šādu pirmkodu (ar dzeltenu fonu apzīmēts sākotnēji dotais pirmkods):

```
[Serializable]
public class Student
{
    /// <summary>
    /// Studenta vārds.
    /// </summary>
    public string Name { get; set; }

    /// <summary>
    /// Studenta uzvārds.
    /// </summary>
    public string Surname { get; set; }

    /// <summary>
    /// Studenta apliecības numurs.
    /// </summary>
    public string Id { get; set; }

    private Student() //nepieciešams priekš [Serializable]
    {
    }

    public Student(string Name, string Surname, string Id)
    {
    }
}
```

```

        if (Name.Length == 0 || Surname.Length == 0 || Id.Length == 0)
            throw new Exception("Invalid student data !");

        this.Name = Name;
        this.Surname = Surname;
        this.Id = Id;
    }

    public override string ToString()
    {
        return Id + " : " + Name + " " + Surname;
    }
}

```

3. Risinājumu pārlūkā projekta **Lab1Library** konteksta izvēlnē izvēlieties komandu **Add | Class...** un dialogā norādiet klases vārdu **StudentList**. Projektam tiks pievienots jauns pirmkoda fails **StudentList.cs** un tas tiks atvērts pirmkoda redaktorā.
4. Papildiniet klases **StudentList** definīciju ar šādu kodu:

```

public class StudentList
{
    /// <summary>
    /// Noklusētais ceļš uz failu, kurā saglabāt studentu sarakstu.
    /// </summary>
    public const string DefaultFilename = @"C:\Work\Lab1\students.xml";

    /// <summary>
    /// Studentu saraksts.
    /// </summary>
    private List<Student> students = new List<Student>();

    /// <summary>
    /// Pievieno sarakstam jaunu studentu.
    /// </summary>
    public void Add(Student newStud)
    {
        if (newStud != null)
            students.Add(newStud);
    }

    public int Count { get { return students.Count; } }

    public Student this[int index]
    {
        get
        {
            if (index < 0 || index >= Count)
                throw new Exception("Invalid student index !");
            return students[index];
        }
    }

    /// <summary>
    /// Saglabā failā tekošo studentu sarakstu.
    /// </summary>
    public void Save(string filename)
    {
        if (filename.Length == 0)
            filename = DefaultFilename;

        FileStream data = new FileStream(filename, FileMode.Create
            , FileAccess.Write);

        XmlSerializer serializer = new XmlSerializer(typeof(List<Student>)
            , new Type[] { typeof(Student) });
        serializer.Serialize(data, students);
    }
}

```

```

        data.Close();
    }

    /// <summary>
    /// Ielādē no faila studentu sarakstu.
    /// </summary>
    public void Load(string filename)
    {
        if (filename.Length == 0)
            filename = DefaultFilename;

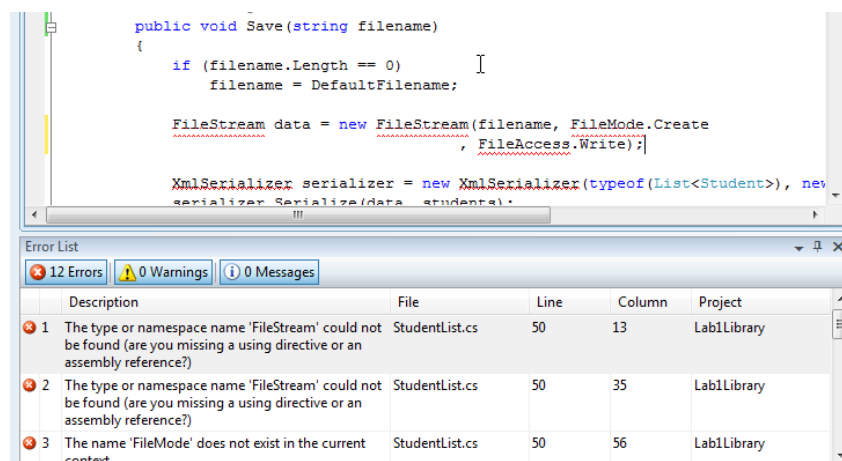
        FileStream data = new FileStream(filename, FileMode.Open
            , FileAccess.Read);

        XmlSerializer serializer = new XmlSerializer(typeof(List<Student>)
            , new Type[] { typeof(Student) });

        students = (List<Student>)serializer.Deserialize(data);
        data.Close();
    }
}

```

Ja mēģināsi veikt pašreizējās risinājuma versijas kompilēšanu (taustiņš **F6**), tad darba zonas apakšējā daļā tiks parādīts kļūdu saraksta logs ar vairāku kļūdu sarakstu, kā arī pirmkoda redaktora fragmenti, kas satur pirmkodu ar nepilnīgu informāciju tiks pasvītroti ar viļņotu sarkanu līniju:



Kļūdu sarakstā katrai konstatētai kļūdai tiek norādīts iespējamā iemesla apraksts (**Description**), faila nosaukums, rindas un kolonnas numuri vietai šajā failā, kā arī projekta nosaukums uz kuru fails attiecas.

Ar peles dubultklikšķi uzklikšķinot uz kļūdas apraksta rindas, koda redaktora kursora pārvietosies uz atbilstošo pozīciju failā.

*Papildus kļūdām var tikt izvadīti arī brīdinājumi (**Warnings**), kas neliedz veiksmīgi pabeigt projekta kompilēšanu, bet potenciāli var norādīt uz problēmām, tāpēc vēlams arī novērst visus brīdinājumus.*

Uzdevums:

1. Ar dubultklikšķi noklikšķiniet uz pirmās kļūdas apraksta, lai pārietu uz rindu, kur minēta klase **FileStream**, kas ir arī pasvītrotā ar viļņotu sarkanu līniju. Kompilators to ir pasvītrojis tāpēc, ka nespēj identificēt šīs klases deklarāciju pēc tās vārda.
2. Lai kļūdu novērstu, kursoram esot uz šīs klases vārda, atveriet konteksta izvēlni un izvēlieties punktu **Resolve | using System.IO**. Pēc šīs darbības klases nosaukums mainīs krāsu un viļņotais pasvītrojums pazudīs, norādot, ka tagad klase ir identificēta.

3. Līdzīgā veidā rīkojieties ar klases **XmlSerializer** identifikatoru.

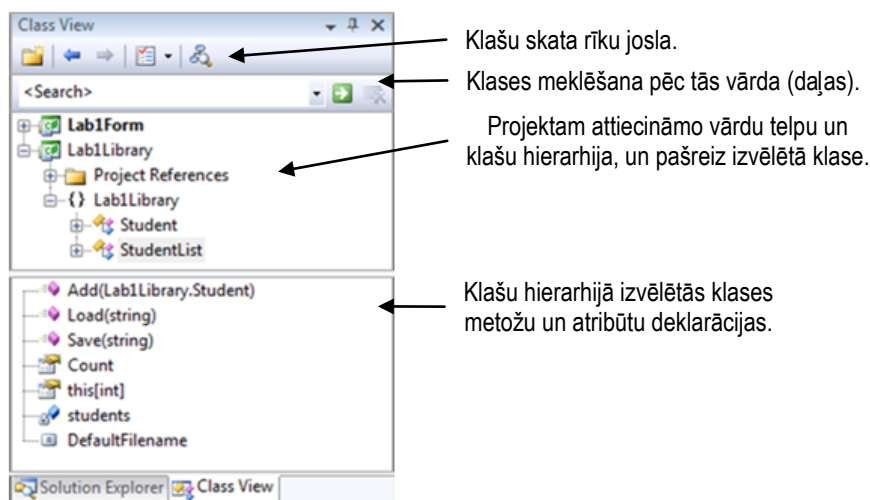
Kā redzams, izstrādes vide spēj ne tikai norādīt uz kļūdām pirmkodā, bet arī automatizēt to novēršanu.

Pirms turpiniet, pārliecinieties, ka risinājuma kompilēšana ir veiksmīga!

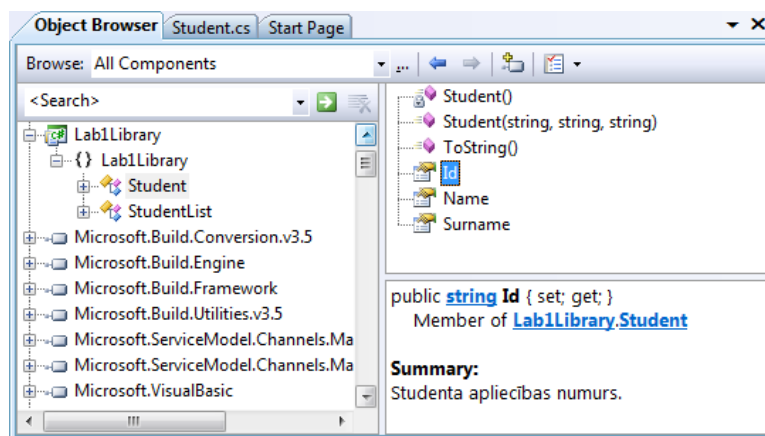
1.8 DARBS AR KLASĒM

Tagad, kad projektā ir definētas vairākas klases (**Student** un **StudentList**), aplūkosim iespējas, kā ar šīm klasēm darboties.

Šim nolūkam izstrādes vide piedāvā speciālu logu – klašu skatu (**Class View**), kuru var atvērt ar galvenās izvēlnes komandu **View | Class View**. Ar šī loga palīdzību var pārlūkot klašu hierarhiju kopumā, kā arī atsevišķas klases īpašības un metodes:

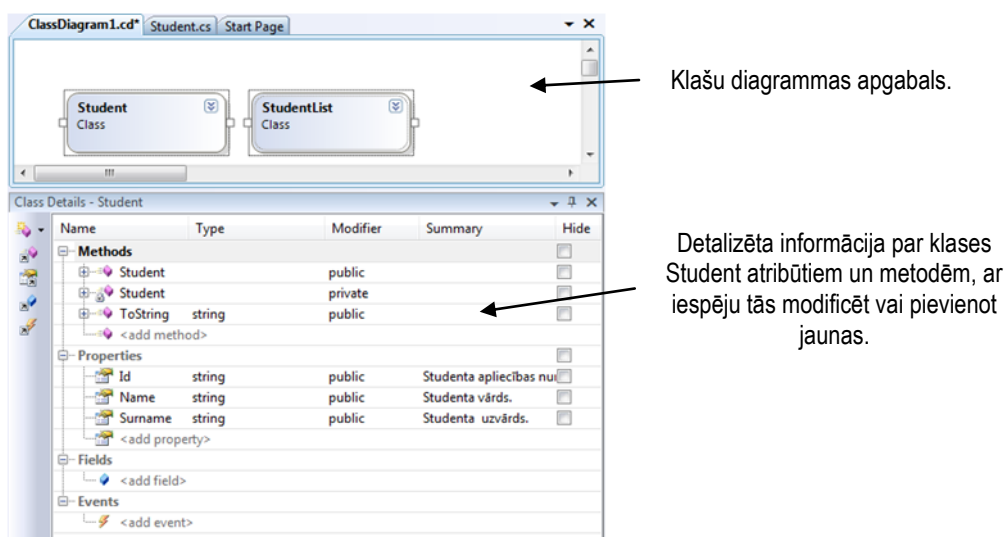


Peles dubultklikšķis uz klases, metodes vai atribūtu deklarācijas, ļauj automātiski atvērt tās definīciju pirmkoda failā. Ja izvēlētai klasei pirmkods nav pieejams, tad tiek atvērts objektu pārlūks (**Object Browser**), kas ļauj apskatīties detalizētāku identifikatora specifikāciju, kā arī meklēt informāciju pēc klases vārda:

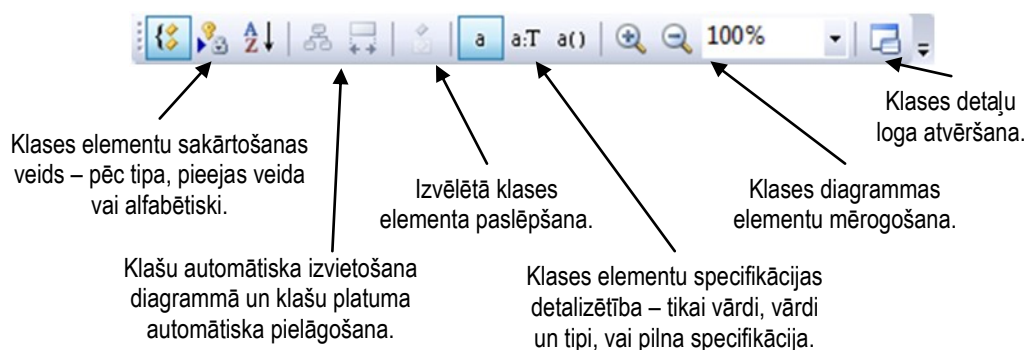


*Objektu pārlūku var atvērt arī ar galvenās izvēlnes komandu **View | Object Browser**.*

Otrs veids, kā apskatīt informāciju par projektā definētajām klasēm un to savstarpējām relācijām, ir izveidot projekta klašu diagrammu. Lai to izdarītu, risinājuma pārlūkā no projekta nosaukuma virsotnes konteksta izvēlnes jāizvēlas punkts **View Class Diagram**. Cilnes logu grupā atvēršies projekta klašu diagramma **ClassDiagram1.cd**, bet zem tās – diagrammā izvēlētais klases detaļas (logs **Class Details**):



Diagrammas apgabalā klases var pārvietot uz nepieciešamo vietu, kā arī mainīt to izmērus. Papildus tam, atvērtai klašu diagrammai galvenajā izvēlnē parādās punkts **Class Diagram**, un rīku joslā piktogrammas, ar kurām var mainīt klases diagrammas attēlošanas veidu un detaļas:




Detalizētāku informāciju par klases atribūtiem un metodēm var aplūkot diagrammā uz klases nospiežot piktogrammu Savukārt klases detaļu logā var rediģēt esošos atribūtus un metodes, kā arī pievienot jaunus – visas izmaiņas automātiski atspoguļosies klases pirmkodā!

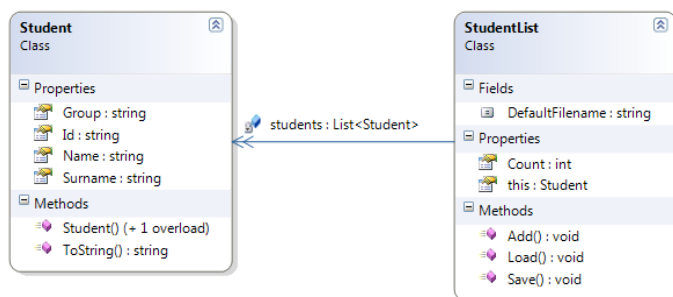
Klašu diagramma ļauj ne tikai labot jau izveidotās klases, bet arī izveidot jaunas (galvenās izvēlnes apakšpunkti **Class Diagram | Add**), kā arī pievienot citas projektā esošās klases, kas klašu diagrammā neparādās (pārvelkot klases faila virsotni no risinājuma pārvaldnieka, vai arī klases nosaukumu no klašu skata logiem).

Uzdevums:

1. Projektam **Lab1Library** atveriet tā klašu diagrammu;
2. Risinājuma pārlūkā pārsauciet diagrammas **ClassDiagram1.cd** nosaukumu uz **StudentClassDiagram.cd**;
3. Paplašiniet klašu diagrammā rādāmās informācijas klāstu, uz abām klasēm nospiežot

piktogrammu ;

4. Paeksperimentējiet ar klases elementu attēlošanas iespējām, diagrammas rīku joslā izvēloties starp dažādiem klases elementu attēlošanas veidiem un klases elementu specifikācijas detalizētību;
5. Lai klases diagrammā izceltu relācijas starp klasi **Student** un **StudentList**, klases **StudentList** elementa students konteksta izvēlnē izvēlieties punktu **Show as Collection Association**. Tā rezultātā starp klasēm tiks parādīta asociācijas relācija, un klases atribūts **students** tiks attēlota uz šīs asociācijas.
6. Pārvietojiet klasi **StudentList** pa labi, lai būtu labāk redzama asociācija.
7. Klasei **Student** pievienojiet jaunu atribūtu **Group** ar tipu **string**. Šim nolūkam klašu diagrammā izvēlieties klasi **Student**, un tās detaļu logā ar peli nospiediet zem zara **Properties** rindā ar tekstu **<add property>**. Teksts **<add property>** pazudīs – tā vietā ierakstiet jaunā atribūta nosaukumu **Group** un, lai pārietu uz lauka tipa norādīšanas kolonnu, nospiediet taustiņu **Tab**. Noklusētā atribūta tipa **int** vietā ierakstiet tipu **string**. Pieejas modifikatora veidu atstājiet **public**, bet lauka kopsavilkuma kolonnā ierakstiet **'Studenta grupa.'**. Tā rezultātā klasei **Student** tiks pievienots jauns atribūts un arī tiks parādīts diagrammā. Saglabājiēt veiktās izmaiņas (**Ctrl+S**).



8. Lai pārliecinātos, ka arī klases **Student** pirmkodā ir ievietots jaunais atribūts, diagrammā ar peles dubultklikšķi noklikšķiniet uz klases **Student** lauka **Group** vai arī uz šī lauka konteksta izvēlnes punkta **View Code** – atvērsies pirmkoda redaktors ar kursora pozīciju pie jaunizveidotā lauka.
9. Nomainiet tās noklusēto definīciju no:

```
/// <summary>
/// Studenta grupa.
/// </summary>
public string Group
{
    get
    {
        throw new System.NotImplementedException();
    }
    set
    {
    }
}
```

uz:

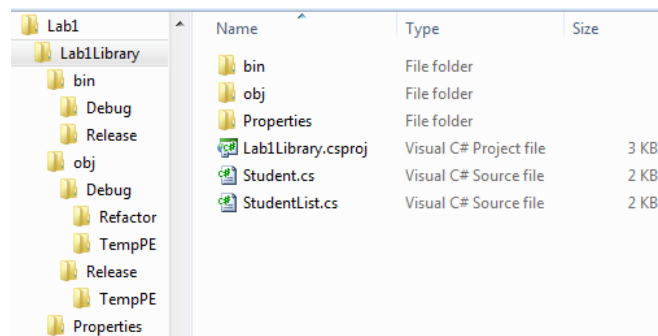
```
/// <summary>
/// Studenta grupa.
/// </summary>
public string Group { get; set; }
```

10. Tā kā klasē **Student** ir definēts konstruktors, kas saņem un inicializē visus tā atribūtus ar sākotnējām vērtībām, papildiniet šīs klases konstruktoru `public Student(string Name, string Surname, string Id)`, pievienojot tam papildus argumentu `string Group`, kā arī tā definīcijā piešķirot šī argumenta vērtību klases atribūtam **Group**.

11. Līdzīgi papildiniet klases **Student** metodes **ToString()** definīciju, lai tā atgrieztu arī atribūta **Group** vērtību teksta virknes veidā.
12. Nokompilējiet risinājumu, lai pārlicinātos, ka tajā nav kļūdas.

1.9 PROJEKTA STRUKTŪRA FAILU SISTĒMĀ

Apskatīsim, kā izskatās pašreizējais risinājums un projekts no failu sistēmas viedokļa. Lai to izdarītu risinājuma pārlūkā uz risinājuma virsotnes izvēlieties konteksta izvēlnes punktu **Open Folder in Windows Explorer**. Atvēršies Windows failu pārlūka logs:



Katalogā **Lab1** atrodas apakškatalogs **Lab1Library** un fails **Lab1.sln**. Pirmais ir projekta katalogs, bet otrais – risinājuma konfigurācijas fails.

Projekta katalogs **Lab1Library**, savukārt, sastāv no trīs apakškatalogiem **bin**, **obj** un **Properties**, kā arī failiem ar paplašinājumu **.cs** un **.csproj**. Faili ar paplašinājumu **.cs** ir C# pirmkoda faili, bet failā **Lab1Library.csproj** glabājas informācija par projekta organizāciju.

*Ja no failu pārlūka nepieciešams atvērt visu risinājumu, tad jāver vaļā fails ar paplašinājumu **.sln**, bet ja tikai konkrētu C# projektu, tad fails ar paplašinājumu **.csproj**.*

Apakškatalogā **Properties** atrodas viens fails – **AssemblyInfo.cs**, kas C# formātā definē papildus informāciju par projektu – versijas numurus, autorus, autortiesības un citu informāciju, kas tie izmantota projekta kompilēšanā un saglabāta projekta gala artefaktos. Apakškatalogi **bin** un **obj** satur apakškatalogus **Debug** un **Release**, katrā no kuriem, savukārt tiek saglabāta informācija atkarībā no risinājuma aktīvās kompilēšanas konfigurācijas. Projekta apakškatalogā **obj** tiek saglabāti kompilēšanas īslaicīgie faili, kurus izmanto arī pie atkārtotas kompilēšanas paātrināšanas. Visbeidzot apakškatalogā **bin** tiek saglabāti faili ar projekta gala artefaktiem – piemēram, klašu bibliotēka failā ar paplašinājumu **.dll**, bet .NET ietvara izpildāmie faili – ar paplašinājumu **.exe**.

Tā kā projekts **Lab1Library** ir klašu bibliotēka, tad tās gala artefakts tiek saglabāts **bin/Debug/Lab1Library.dll** vai **bin/Release/Lab1Library.dll** gala artefaktā. Kā papildus rezultāts abās risinājuma konfigurācijās ir fails ar paplašinājumu **.pdb** – tajā, atsevišķi no gala artefakta tiek saglabāta atklūdošanas informācija. Šo failu var izmantot speciāli atklūdošanas rīki, lai iegūtu papildus informāciju par **Lab1Library.dll** saturu. Tomēr to obligāta kopēšana galalietotāju datoros kopā ar **Lab1Library.dll** nav nepieciešama.

No galvenās izvēlnes izvēlieties komandu **Build | Clean Solution**, tiek izdzēsti **bin** un **obj** apakškatalogu saturi, ieskaitot arī gala artefaktus – lai tos atjaunotu, nepieciešams veikt atkārtotu projekta kompilēšanu.

1.10 JAUNA PROJEKTA PIEVIENOŠANA ESOŠAM RISINĀJUMAM

Uzdevums:

Izveidojiet jaunu projektu risinājumā **Lab1**.

1. Šim nolūkam risinājumu pārlūkā uz risinājuma nosaukuma konteksta izvēlnes izvēlieties punktu **Add | New Project...**;
2. Par projekta sagatavi izvēlieties **Windows Forms Application** un par tās nosaukumu norādiet **Lab1Form**.
3. Tiks izveidots jauns projekts, kas automātiski būs pievienots arī risinājumam **Lab1**, kā arī cilnes grupā tiks atvērts jauns formas dizainera logs ar formas **Form1** sagatavi (logs **Form1.cs [Design]**).

Tā kā līdz šim netika apskatīts projekta gala artefakts, kurš būtu patstāvīgi izpildāms, tad tagad ir iespējams palaist projekta **Lab1Form** gala artefaktu – **Lab1Form.exe**. Palaīšanu atklādošanas režīmā var veikt ar galvenās izvēlnes komandu **Debug | Start Debugging** (vai ar taustiņu **F5**), vai arī palaist parastā režīmā ar komandu **Debug | Start Without Debugging** (taustiņu kombinācija **Ctrl+F5**).

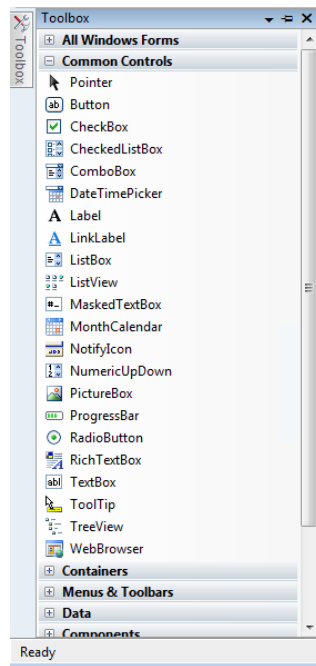
Abos gadījumos vispirms tiek veikta nepieciešamo projekta artefaktu kompilēšana, un tikai tad, ja kompilēšana ir veiksmīga, tiks palaists uz izpildi projekta gala artefakts.

Tomēr, ja to mēģināsi veikt risinājuma kompilēšanu vai palaīšanu uz izpildi, tad formas parādīšanās vietā tiks izvadīts kļūdas ziņojums, ka projektu **Lab1Library** nav iespējams palaist uz izpildi. Tas ir tāpēc, ka risinājuma konfigurācijā kā palaizamais projekts ir kļūda bibliotēka **Lab1Library.dll**. Nepieciešams norādīt, ka palaizot risinājumu, ir jāizpilda projekts **Lab1Form.exe**.

Uzdevums:

1. Risinājuma pārlūkā, uz projekta **Lab1Form** nosaukuma atveriet konteksta izvēlni un izvēlieties komandu **Set as StartUp Project**.
2. Par risinājuma aktīvo izpildāmo projektu liecinās tas, ka tā nosaukuma virsotne risinājuma pārlūkā tiek atspoguļota treknrakstā;
3. Nospiediet risinājuma palaīšanas taustiņu **F5** – tiks atvērta tukša formas lietojumprogramma;
4. Aizveriet palaisto lietojumprogrammas formu.

Lietojumprogrammu projektiem, kuri sastāv no formām, ir pieejamas iespējas, kas nodrošina darbu ar vizuālo programmēšanu. Galvenais logs jeb palete, kurā glabājas pieejamie vizuālie (kontroles) un nevizuālie komponenti, pēc noklusēšanas ir piestiprināts un minimizēts darba zonas kreisajā pusē un to var atvērt uzklikšķinot uz šī loga virsraksta **Toolbox**:



Kā redzams, komponenti ir sakārtoti pa kategorijām, kas ļauj tos vieglāk atrast.

Lai nepieciešamo komponentu pārvietotu uz aktīvās formas virsmu, jārikojas vienā no veidiem:

1. Noklikšķinot vispirms uz komponenta paletē un pēc tam ar peli velkot to uz formu, kur nepieciešamajā vietā atlaiž – šādā veidā uz formas norādītajā vietā tiks novietota komponenta instance, ar tās noklusēto izmēru vērtībām;
2. Noklikšķinot uz komponenta paletē un pēc tam noklikšķinot uz formas nepieciešamā vietā, un, neatlaižot peli, norādot komponenta izmērus. Atlaižot peli uz formas norādītajā vietā tiks novietota komponenta instance ar uzstādītiem izmēriem;
3. Ar dubultklikšķi noklikšķinot uz komponenta paletē – šajā gadījumā komponents tiks novietots uz formas patvaļīgā vietā.

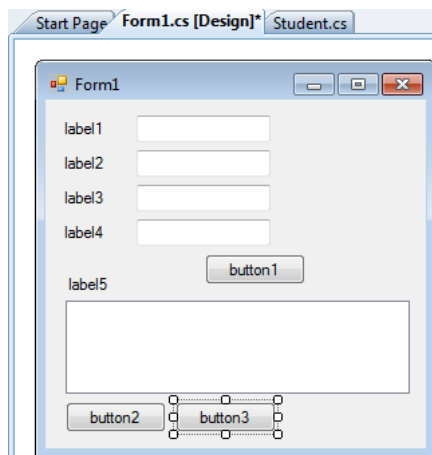
Uz formas novietotās kontroles, var pārvietot pa formas virsmu un, ja to pieļauj kontroles īpašības, mainīt tās izmērus.

Katram komponentam ir definēta sava īpašību kopa, kuras elementu vērtības nosaka komponenta vizuālo atspoguļojumu un uzvedību gan programmēšanas, gan izpildes laikā.

Tā kā šajā brīdī mērķis ir iepazīties tieši ar izstrādes vidi, tad nepieciešamie komponenti tiks apskatīti tikai minimāli nepieciešamajā apjomā – detalizēts katra komponenta apskats un pielietojums tiek dots turpmākajos laboratorijas darbos.

Uzdevums:

Izmantojot tikko aprakstītos komponentu instanču novietošanas mehānismus, novietojiet uz formas instances **Form1**, no komponentu paletes **Common Controls** kategorijas piecas kontroles **Label**, četras kontroles **TextBox**, trīs kontroles **Button** un vienu kontroli **ListView**, izkārtējot tās šādā veidā:



Ievērojiet, ka kontroļu izvietošanas laikā parādās palīglīnijas, kas atvieglo to izvietošanu, lai tās būtu izlīdzinātas un ar līdzīgiem attālumiem viena no otras, kā to nosaka saskarņu projektēšanas labā prakse.

Palaidiet risinājumu uz izpildi – atvēršies forma un tajā būs redzamas visas izvietotās kontroles. Jāuzsver, lai iegūtu šo funkcionalitāti nekāds pirmkods nebija manuāli jāraksta.

*Jā traucē, ka komponentu paletei ir iestādīta pazīme **Auto Hide**, tad to vienkārši atslēdziet – komponentu palete būs redzama visu laiku.*

Nākošais solis ir pielāgot katras izvietotās kontroles īpašības, lai tās realizētu lietojumprogrammas uzvedību. Tas tiek darīts, aktivizējot nepieciešamo komponentu uz formas un pēc tam īpašību logā nomainot komponenta nepieciešamo īpašību vērtības.

Daudzām komponentēm īpašību nosaukumi un iespējamās vērtības ir vienādas. Ir īpašības, kurām tikai sakrīt nosaukumi, bet vērtības var būt dažādas. Visbeidzot ir komponentu īpašības, kas ir unikālas tikai vienam noteiktam komponentam.

Pilnīgi visiem komponentiem ir īpašība **(Name)**, kas unikāli identificē komponentes instances nosaukumu tās redzamības apgabalā. Piemēram, ja teksta lauka kontroles instances **textBox1** īpašību **(Name)** nomaina uz **txtName**, tad turpmāk uz šo komponenti var atsaukties tikai izmantojot **txtName**. Daudziem komponentiem ir īpašība **Text**, kas glabā tekstu, kas tiek atspoguļots uz komponenta gan programmēšanas, gan izpildes laikā, pie tam šo īpašību var mainīt arī izpildes laikā.


*Programmēšanas labā prakse nosaka, ka pilnīgi visiem komponentiem ir jāpiešķir tādi nosaukumi, kas atspoguļo to mērķi – nav pieņemts atstāt noklusētos instanču nosaukumus. Bet, lai pēc komponenta nosaukuma varētu noteikt arī tā tipu, nosaukuma sākumā raksta 2-3 simbolu kombināciju, kas saīsināti atspoguļo komponenta tipu. Piemēram, teksta lauka komponenta nosaukumu sāk ar rakstzīmēm **txt...***

Uzdevums:

Izmainiet kontroles instanču īpašības uz šādām (ja īpašību logs nav atvērts, tad izdariet to ar galvenās izvēlnes komandu (**View | Properties Window**):

Kontroles instance	Īpašība	Vērtība
Form1	(Name)	frmStudents
	Text	Studenti

label1	(Name) Text	lblName Vārds:
label2	(Name) Text	lblSurname Uzvārds:
label3	(Name) Text	lblId Apl.Nr.:
label4	(Name) Text	lblGroup Grupa:
label5	(Name) Text	lblStudentList Studentu saraksts:
textBox1	(Name)	txtName
textBox2	(Name)	txtSurname
textBox3	(Name)	txtId
textBox4	(Name)	txtGroup
button1	(Name) Text	btnAddStudent Pievienot sarakstam
button2	(Name) Text	btnSave Saglabāt
button3	(Name) Text	btnLoad Ielādēt
listView1	(Name) View VirtualMode	lstStudents Details true

Detalizētāk jāapskata kontroles **ListView** īpašības **Columns** izmaiņa – ja to izvēlas īpašību logā, tad pa labi parādās poga ar daudzpunktiem (**Columns** **(Collection)** ), uz kuras uzspiežot tiks atvērts saraksta virsrakstu redaktors.

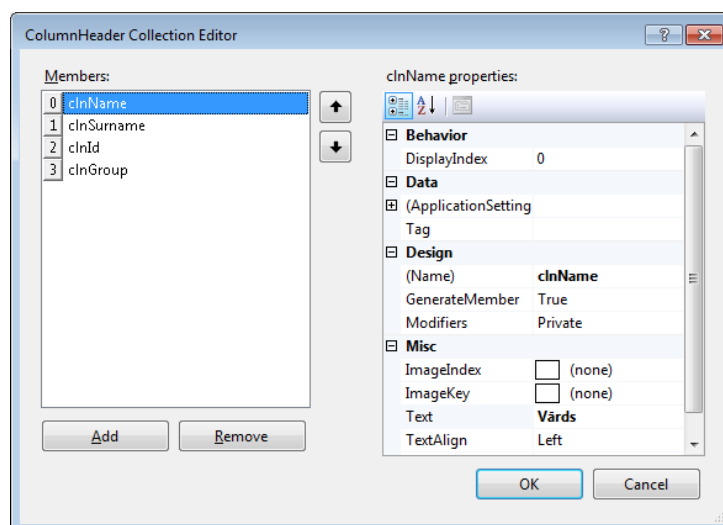
Tajā redzams, ka katrs saraksta virsraksts arī ir objekts ar savām īpašībām. Lai saraksta kontrolei pievienotu jaunu kolonnu un definētu tās virsrakstu, jāspiež poga **Add** un pa labi esošajās saraksta kolonnas virsraksta īpašībās jānorāda tās instances nosaukums (īpašība **(Name)**) un kolonnas virsraksta teksts (īpašība **Text**).

Uzdevums patstāvīgai izpildei:

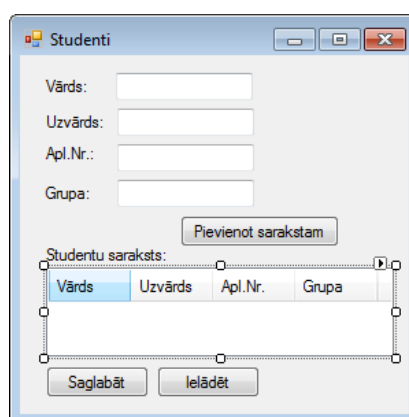
1. Patstāvīgi pievienojiet četras kolonnas, un to instancēm piešķiriet šādas vērtības:

Īpašības <i>(Name)</i> vērtība	Īpašības <i>Text</i> vērtība
clnName	Vārds
clnSurname	Uzvārds
clnId	Apl.Nr.
clnGroup	Grupa

2. Rezultātā tiks iegūts šāds saraksta kolonnas virsrakstu objektu saraksts:



3. Ar **OK** aizveriet dialogu, pēc visu instanču īpašību maiņas formas dizainam jābūt šādam:



4. Palaižot risinājumu uz izpildi, būs iespējams ievadīt tekstu teksta kontrolēs, bet, spiežot pogas, nekāda reakcija nesešos. Tas ir tādēļ, ka pogu kontrolēm nav piesaistīti notikumu apstrādātāji, kuru uzdevums ir reaģēt uz atbilstošo notikumu.

Vizuālajā programmēšanā notikumu apstrādāšanas princips tiek ļoti plaši pielietots. Tas ir tāpēc, ka grafiskajai saskarnei parasti nav iespējama secīga mijiedarbība ar lietotāju – lietotājs patvaļīgā secībā darbojas ar saskarnes kontrolēm. Tāpēc reakcija uz lietotāja darbībām tiek piesaistīta katrai saskarnes kontrolei individuāli. Piemēram, ja lietotājs teksta lauka kontrolē vada tekstu, šai kontrolei iestājas notikums par teksta izmaiņu. Lietojumprogrammā par šo notikumu tiek informēts kontrolei piesaistīts notikumu apstrādātājs, kurā realizēta reakcija uz šī notikuma iestāšanos. Vai, ja lietotājs nospiež ar peli vai klaviatūras taustiņu pogu, tad šai pogai iestājas tās nospiešanas notikums, uz kuru, notikumu apstrādātām atbilstoši reaģējot, var realizēt lietojumprogrammas funkcionalitāti.

Izstrādes vidē katrai kontrolei ir definēts t.s. noklusētais notikums – tā apstrādātājs tiek izveidots un piesaistīts kontrolei brīdī, kad izstrādes vidē uz kontroles instances ar peli tiek veikts dubultklikšķis.

Uzdevums:

Piesaistīt notikumu apstrādātājus trīs pogu kontrolēm, lai tās reaģētu uz nospiešanu, kas pogu kontrolēm ir noklusētais notikums.

1. Ar dubultklikšķi nospiediet uz pogas 'Pievienot sarakstam', atvērsies pirmkoda redaktors,

ar pogas nospiešanas notikuma apstrādātāja tukšu metodi. Papildiniet metodes realizāciju ar šādu saturu:

```
private void btnAddStudent_Click(object sender, EventArgs e)
{
    try
    {
        students.Add(new Student(txtName.Text, txtSurname.Text
                                , txtId.Text, txtGroup.Text));
        InvalidateList();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

2. Redzams, ka pēc iekopēšanas ir pasvītroti identifikatori **students**, **Student** un **InvalidateList()**. Klase **Student** ir pasvītrotā, jo kaut arī projekts **Lab1Form** atrodas tajā pašā risinājumā, kur **Lab1Library**, lai no formas varētu izmantot bibliotēkā definētos objektus, nepieciešams projektam **Lab1Form** norādīt, kur jāmeklē klases bibliotēka **Lab1Library**;
3. Risinājuma pārlūka konteksta izvēlnē uz projekta **Lab1Form** izvēlieties punktu **Add Reference...** un dialogā, pārejot uz cilni **Projects**, iezīmējiet **Lab1Library** un spiediet pogu **OK**. Pēc šīm darbībām klase **Student** joprojām būs pasvītrotā ar viļņotu sarkanu līniju, jo tā atrodas vārdu telpā **Lab1Library**.
4. Atveriet pirmkoda redaktoru failam **frmStudents.cs** un tajā pārejiet uz klases **Student** vārdu, un uz tā konteksta izvēlnē izvēlieties punktu **Resolve | using Lab1Library**; Pēc šīm darbībām klase **Student** kļūst pieejama projektā **Lab1Form**.
5. Lai ieviestu identifikatoru **students**, klasē **frmStudents** definējiet jaunu atribūtu ar šādu specifikāciju (ja ir atvērts formas redaktors, tad uz pirmkoda redaktoru var pāriet arī nospiežot taustiņu **F7**, bet atpakaļ uz formu – **Shift+F7**):

```
public partial class frmStudents : Form
{
    private StudentList students = new StudentList();

    public frmStudents()
    {
```

6. Atveriet formas redaktoru un līdzīgā veidā pievienojiet pogu ‘Saglabāt’ un ‘Ielādēt’ notikumu apstrādātājus:

```
private void btnSave_Click(object sender, EventArgs e)
{
    try
    {
        students.Save(StudentList.DefaultFilename);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

private void btnLoad_Click(object sender, EventArgs e)
{
    try
    {
        students.Load(StudentList.DefaultFilename);
        InvalidateList();
    }
    catch (Exception ex)
```

```

        {
            MessageBox.Show(ex.Message);
        }
    }
}

```

7. Visbeidzot klasē **frmStudents** nedefinējiet metodi **InvalidateList()** uzreiz aiz atribūta **students** deklarācijas:

```

private StudentList students = new StudentList();


private void InvalidateList()
{
    lstStudents.VirtualListSize = students.Count;
    lstStudents.Invalidate();
}

```

```

public frmStudents()
{

```

8. Pēc šīm darbībām nokompilējiet risinājumu (**F6**) – kompilēšanai jābūt veiksmīgai.
9. Atliek vēl viena notikuma apstrādātāja realizācija – tā kā iepriekš kontroles **lstStudents** īpašība **VirtualMode** tika uzstādīta uz **true**, tad nepieciešams realizēt speciālu notikumu, kas iestāsies, kad saraksta kontrolei būs nepieciešams pārzīmēt saraksta saturu (šī notikuma izraisīšanu nodrošina iepriekš definētā metode **InvalidateList()**) – šī notikuma nosaukums ir **RetrieveVirtualItem()**.
10. Izstrādes vides formas redaktorā atveriet **frmStudents**, iezīmējiet kontroles instanci **lstStudents** un, nospiežot īpašību loga piktogrammu , pārslēdziet uz notikumu apstrādātāju sarakstu.
11. Notikumu sarakstā atrodiet rindu ar notikumu **RetrieveVirtualItem** un ar peles dubultklikšķi izveidojiet un kontrolei piesaistiet šī notikuma apstrādātāju, papildinot tā reakciju ar šādu realizāciju:

```

private void lstStudents_RetrieveVirtualItem(object sender,
RetrieveVirtualItemEventArgs e)
{
    try
    {
        Student s = students[e.ItemIndex];
        e.Item = new ListViewItem(s.Name);
        e.Item.SubItems.Add(s.Surname);
        e.Item.SubItems.Add(s.Id);
        e.Item.SubItems.Add(s.Group);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

```

Pēc šīm darbībām formas projekta kompilēšanai jābūt veiksmīgai, kā arī to palaižot uz izpildi formas pogas strādās tā kā tas nepieciešamas – nospiežot pogu *‘Pievienot sarakstam’* teksta lauku vērtības atspoguļosies studentu saraksta elementā. Nospiežot pogu *‘Saglabāt’* saraksta dati tiks saglabāti XML formātā failā **C:\Work\Lab1\students.xml**, bet nospiežot pogu *‘Ielādēt’*, no šī faila, ja tas būs korekts, tiks ielādēti saglabātie dati un parādīti formas sarakstā.

1.11 UZDEVUMI PASTĀVĪGAI IZPILDEI

1.11.1 KONSOLĒS TIPIA LIETOJUMPROGRAMMAS IZVEIDE

Līdzīgi kā projekta **Lab1Form** gadījumā, risinājumam **Lab1** pievienot vēl vienu projektu – šoreiz ar sagatavi **Console Application**. Nosaukt projektu par **Lab1Console**, un pabeigt projektu, lai bez kļūdām strādātu šāda realizācija:

```
namespace Lab1Console
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                StudentList students = new StudentList();
                students.Load(StudentList.DefaultFilename);
                for (int i = 0; i < students.Count; i++)
                    System.Console.WriteLine(students[i]);
            }
            catch (Exception ex)
            {
                System.Console.WriteLine(ex.Message);
            }
            System.Console.ReadLine();
        }
    }
}
```

1.11.2 JAUNA DATU ATRIBŪTA PIEVIENOŠANA

Projektā **Lab1Library** papildāt klases **Student** definīciju ar jaunu atribūtu **Email**, kura tips ir **string**.

Realizēt šī atribūta ievades un parādīšanas iespēju formā **frmStudents**, ievietošanu studentu sarakstā, kā arī nodrošināt, lai pēc veiksmīgas pievienošanas sarakstam visi teksta lauki formā tiktu automātiski attīrīti.