Лекція 2. Інструменти конструювання програмного забезпечення

Елементарні інструменти конструювання ПЗ

Редактор коду

Якщо згадати історію галузі розробки ПЗ, процес складання програм довгий час залишався ручним. Програміст записував програму на спеціальному бланку, відносив у відділ перфорації, де оператори за допомогою спеціального устаткування наносили програму на перфокарти або перфострічки. З них програма завантажувалася в ЕОМ, запускалася, в ній виявлялися помилки, програміст їх виправляв, знову "набивав" перфокарти і так далі.

Нині цей рутинний процес пішов в минуле. Сучасний програміст, як правило, не користується папером для запису програм, а відразу заносить її текст в комп'ютер "з голови", користуючись так званими редакторами коду (редакторами текстів) або текстовими процесорами.

Редактор коду – це програмна система, що забезпечує первинну підготовку початкового тексту програми і його виправлення в процесі розробки. В якості інструментів для створення і модифікації вихідних кодів використовувались і використовуються понині текстові редактори загального призначення, що є поширеним в UNIX і UNIX-подібних середовищах. Наприклад: vi (UNIX), Vim, SciTE (кросплатформенні), Notepad++, Far (Windows). Подібно до текстових процесорів (мабуть найвідомішим з них є Microsoft Word), які спеціалізовані для створення текстових документів існують і редактори коду спеціалізовані для роботи саме з початковими текстами програм. Вони не мають маси функцій звичайних для текстових процесорів (на зразок роботи з таблицями і малюнками), зате надають інші функції не менш корисні з підтримкою специфіки цільової мови програмування. Існує досить велика кількість різних редакторів коду, список їх можливостей також дуже великий. Серед стандартних функцій можна виділити:

- простий набір тексту;
- комбінування окремих фрагментів;
- пошуку за зразком;
- виділення кольором різних елементів програми;
- автоматичне форматування відповідно до сталих правил оформлення коду для тої або іншої мови програмування (ці правила часто називають стилем) [1].

Редактор коду Notepad++ [4]

Notepad++ — текстовий редактор, призначений для програмістів і тих, кого не влаштовує скромна функціональність Блокнота, що входить до складу Windows. Notepad++ базується на компоненті Scintilla (потужному компоненті для редагування), написаному на C++ з використанням тільки Win32 API і STL, що забезпечує максимальну швидкість роботи при мінімальному розмірі програми.

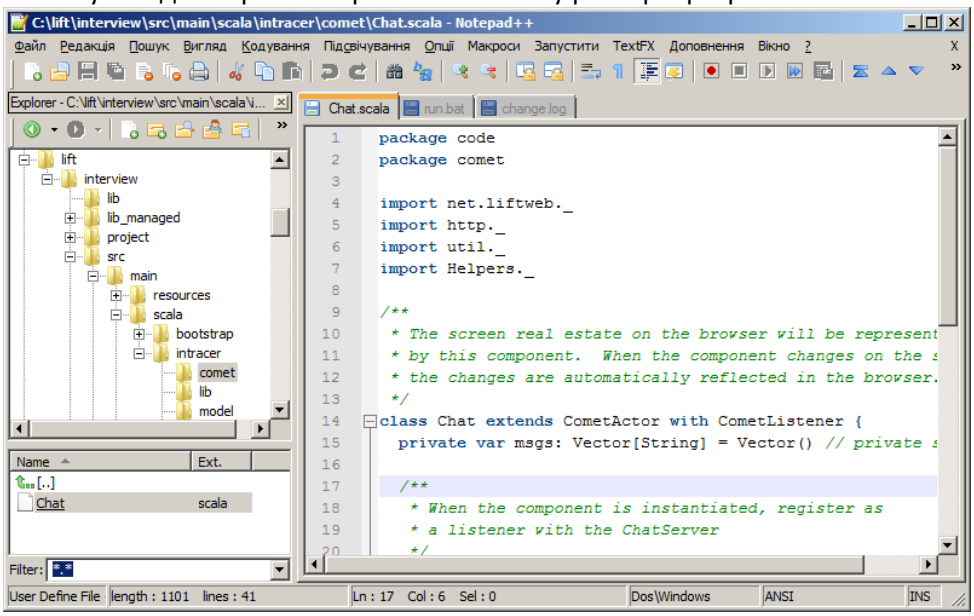

Рис. 1. Редактор коду Notepad++

Інтерфейс у Notepad++ — багатомовний. Серед особливостей програми — підсвічування синтаксису, підтримка великої кількості мов (C, C++, Java, XML, HTML, PHP, Java Script, ASCII, VB/VBS, SQL, CSS, Pascal, Perl і Python), багатомовна підтримка, робота з декількома документами.

Основні можливості Notepad++:

- Підсвічування тексту і можливість згортання блоків, згідно з синтаксисом мови програмування
- WYSIWYG (друкуєш і отримуєш те, що бачиш на екрані)
- Режим підсвічування синтаксису, що налаштовується користувачем
- Авто-завершення слова, що набирається
- Одночасна робота з безліччю документів
- Підтримка регулярних виразів для пошуку/заміни
- Повна підтримка перетягування фрагментів тексту
- Динамічна зміна вікон перегляду
- Автоматичне визначення стану файлу
- Збільшення і зменшення
- Підтримка великої кількості мов
- Замітки
- Плаґіни
- Запис макросу і його виконання.

Редактор коду MS Visual Studio [2]

Visual Studio має декілька текстових редакторів. Усі текстові редактори базуються на загальному ядрі, яке надає основний набір функціональних можливостей для кожного редактора (такі як поле вибору, можливість згортати вкладені елементи, забарвлення тексту). Кожен редактор походить від цього ядра і має налаштування для мови (C#, VB…), для XML, для HTML (ASPX) або для редактора таблиць стилів.

Редактор коду Visual Studio, як один з найкращих редакторів, включає широкий набір функцій:

- обробляє відступи і пробільні символи для того, щоб ваш код був зрозумілим і читабельним;
- забезпечує технологію IntelliSense (*про яку можна поговорити докладніше пізніше*) і дописування операторів (для того, щоб звільнити розробника від необхідності пошуку (чи запам'ятовування) кожної бібліотеки об'єктних модулів або ключового слова);
- групує код у блоки;
- забезпечує забарвлення ключових слів і коментарів;
- виділяє помилки;
- виділяє новий код відносно раніше скомпільованого;
- пошук та заміна.

Редактор коду C# показаний на рис. 1. Необхідно відмітити деякі елементи.

1. Код автоматично групується в логічні області (помічаються зліва). Ви можете використати знак мінуса для того, щоб закрити цілий клас, метод, властивість або іншу подібну групу. Ця можливість дозволяє вам приховувати той код, з яким ви в даний момент не працюєте. Ви можете також створити ваші власні (іменовані) області коду для цієї ж мети (в C# це робиться за допомогою директив `#region … #endregion`).

2. Новий код усередині поля індикації (ліве поле редактора) позначається кольоровою лінією. Жовтий колір використовується для нового коду, який ще не збережений. Лінія стає зеленою після збереження і зникає після того, як ви закриєте і знову відкриєте файл. Ця функціональна можливість дозволяє вам (і редакторові) відстежувати місця виконаних під час поточної сесії змін в коді.

3. Ім'я відкритого файлу показане на заголовку вікна коду. Зірочка вказує, що з моменту останнього збереження код змінився.

4. При наборі коду запускається IntelliSense. Для швидкого знаходження в списку потрібного елементу ви можете використати клавіші із стрілками. При наведенні покажчика миші на елемент вам будуть показані подробиці цього елементу (текст підказки буде справа). Ви можете натиснути клавішу <Tab> для дописування елементу усередині IntelliSense.

5. Код виділяється різними кольорами. За умовчанням ключові слова мають синій колір, коментарі — зелені, текст — чорний, створювані вами типи — блакитні, строкові значення — червоні, і т. д.

6. Два випадні списки у верхній частині редактора коду дозволяють вам переміщатися між класами у файлі (лівий випадний список) і методами, полями і властивостями цього класу (правий випадний список).

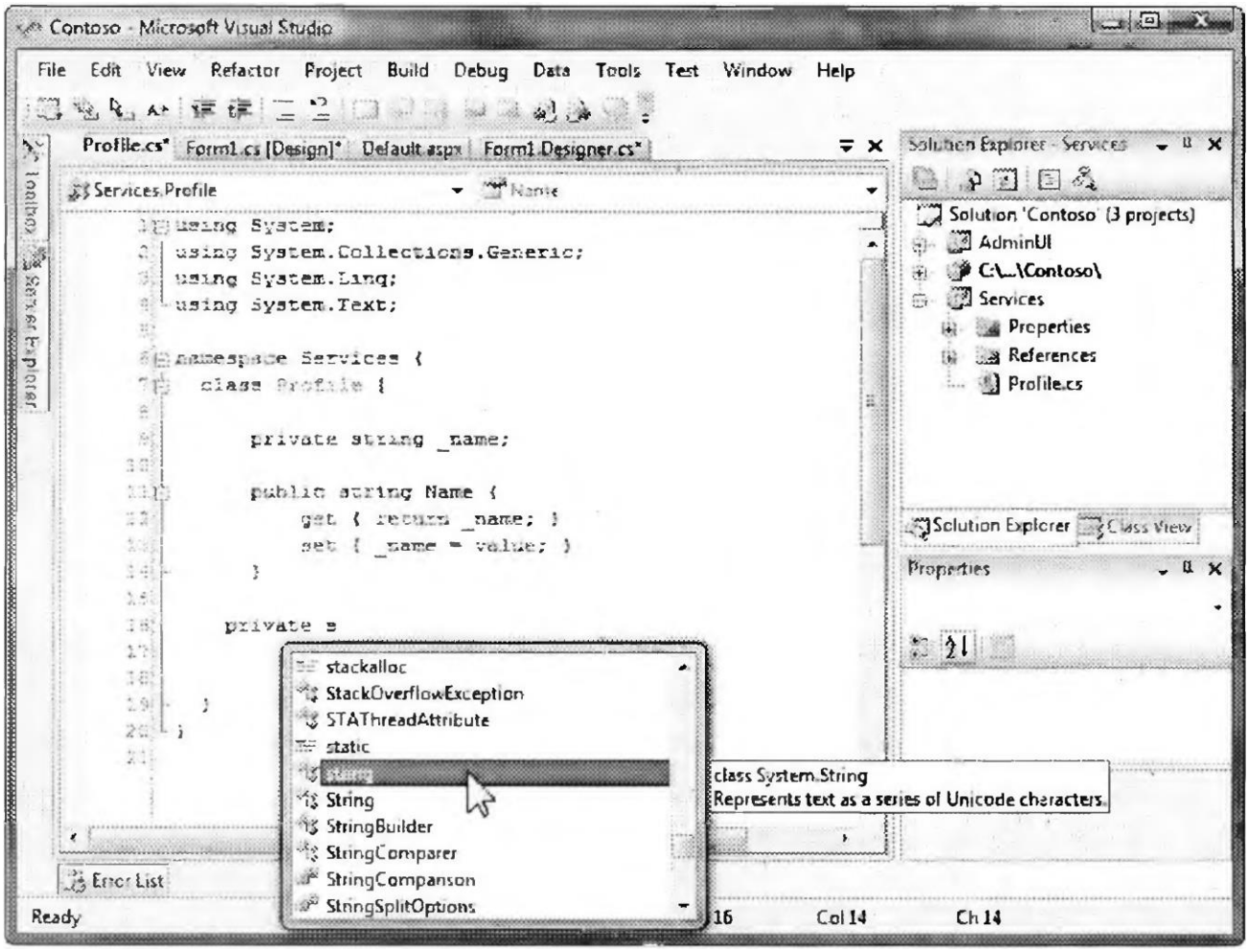

Рис. 1. Редактор коду C#

Налаштування редактора MS Visual Studio

Практично будь-який аспект текстового і кодового редактора може бути налагоджений під будь-які ваші капризи. Ми знаємо із власного досвіду, що не існує навіть двох розробників з однаковим сприйняттям коду. Ви можете використати діалогове вікно Options (Tools | Options) для зміни фонового кольору редактора або кольору і шрифту різного тексту усередині редактора. Ви можете також включити нумерацію рядків і управляти відступами (табуляцією) і пробільними символами. Повний список налаштувань текстового редактора великий. Ви можете настроїти мову і специфічні для редактора опції.

На рис. 2 показано діалогове вікно Options для шрифтів і кольорів. Тут ви можете настроїти безліч елементів редактора (їх колір, шрифт і розмір шрифту), що відображаються.

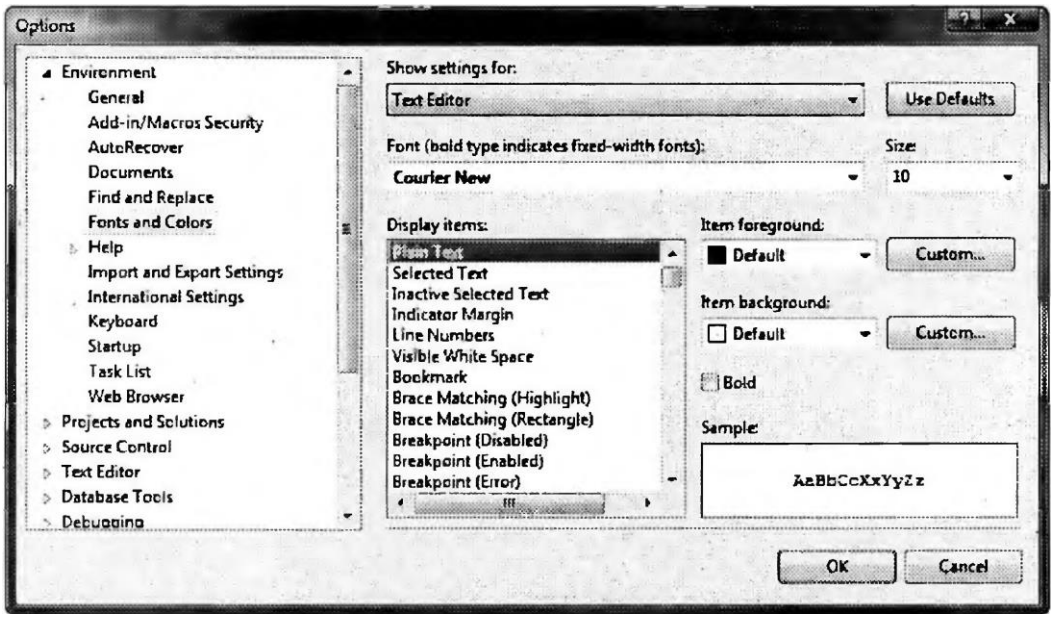

Рис. 2. Ділогове вікно з налаштуваннями шрифтів і кольорів

Якщо ви заглибитеся в діалогове вікно Options, то в дереві опцій ви наштовхнетеся на вузол Text Editor. Тут ви можете маніпулювати ще більшою кількістю налаштувань для текстового редактора. Наприклад, ви можете прибрати горизонтальні роздільники процедур в редакторові Visual Basic або відключити автоматичне переформатування коду редактором.

Що ще краще — ви можете управляти тим, як редактор автоматично форматує ваш код усередині редактора коду C#. Якщо ви хочете, щоб усі ваші фігурні дужки стояли на окремих рядках, або вважаєте за краще, щоб вони знаходилися на початку рядка, з яким починається блок коду, то можете настроїти це тут. На рис. 3 показані деякі опції форматування коду C# в редакторові.

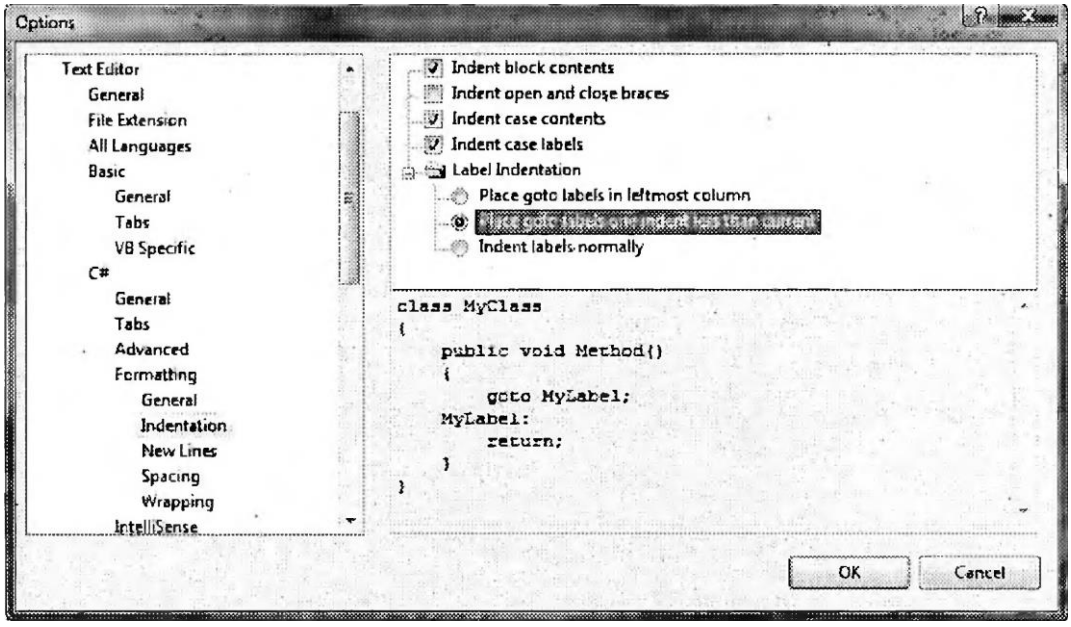

Рис. 3. Управління форматуванням коду

Засоби підвищення продуктивності [2]:

- смарт-теґі
- IntelliSense

Транслятор (компілятор)

Підготовлена за допомогою редактора коду програма запам'ятовується у вигляді одного або декількох файлів і надалі служить вхідною інформацією для транслятора.

Під транслятором (translator) зазвичай розуміють спеціальну програму, яка переводить текст програми в послідовність машинних команд. Транслятори мов високого рівня, таких як C, C++, Pascal і інших, називають зазвичай компіляторами (compiler).

Традиційно, компілятори були не інтерактивними (командними) трансляторами початкового коду. Проте, існує тенденція інтеграції компіляторів і редакторів в інтегровані середовища програмування. Такі як, наприклад, MS Visual Studio.

Компілятор .NET csc.exe

Компіляцію вихідного коду на C# можна виконати маючи тільки Microsoft .NET Framework, який включає компілятор командного рядка для C#, csc.exe (C-Sharp Compiler). Компілятор знаходиться в стандартній папці “C:\Windows\Microsoft.NET\Framework\v3.5” (дві останні папки шляху залежать від розрядності ОС і версії .NET).

Користуватись csc.exe для компіляції простої програми нескладно. Достатньо просто запустити компілятор командного рядка і передати йому ім’я вихідного файлу:

```
csc C:\First.cs
```

Компіляція коду дасть в результаті виконуваний файл First.exe, який можна запустити в командному рядку.

Компілятор csc має багато параметрів. Кожен параметр, що передається csc, повинен мати префікс у виді дефісу або косої риски. Повний список параметрів з короткими поясненнями можна отримати викликавши компілятор зі знаком питання

```
csc -?
```

Розглянемо лише окремі з параметрів (табл. 1).

При написанні навіть найпростішого застосування доводиться використати типи, визначені в окремих складках .NET. Для цього в командному рядку треба проінформувати компілятор csc.exe про те, в якій зборці містяться використовувані в програмі зовнішні простори імен (наприклад, System.Windows.Forms, визначено в

зборці System.Windows.Forms.dll). Посилання на зовнішню зборку вказується за допомогою прапора /reference(чи його скороченій версії /r) :

```
csc /r: System.Windows.Forms.dll TestApp.cs
```

Таблиця 1. Вихідні параметри, які може приймати компілятор C#

Параметр	Опис
/out	Цей параметр застосовується для вказівки імені створюваного складання За умовчанням складанню привласнюється те ж ім'я, що у вхідного файлу *. cs
/target: exe	Цей параметр дозволяє створювати виконуване консольне застосування. Складання такого типу генерується за умовчанням, тому при створенні подібного застосування цей параметр можна опускати (Коротка форма: /t: exe)
/target: library	Цей параметр дозволяє створювати однофайлове складання *.dll (Коротка форма: /t: library)
/target: module	Цей параметр дозволяє створювати модуль. Модулі є елементами багатофайлових складок (Коротка форма: /t: module)
/target: winexe	Хоча додатки з графічним призначеним для користувача інтерфейсом можна створювати із застосуванням параметра /target : exe, параметр /target : winexe дозволяє запобігти відкриттю вікна консолі під іншими вікнами (Коротка форма: /t: winexe)

Коли необхідно вказати csc.exe декілька зовнішніх складок, для цього треба просто перерахувати усі складки через крапку з комою. Нижче приведена команда, яка ілюструє перерахування безлічі складок :

```
csc /r: System.Windows.Forms.dll;System.Drawing.dll *.cs
```

У більшості випадків проекти формуються з декількох файлів *. cs для надання кодовій базі більшої гнучкості. Щоб скомпілювати усі файли початкового коду#, необхідно їх явно перерахувати як вхідні файли:

```
csc /r: System.Windows.Forms.dll TestApp.cs HelloMsg.cs
```

В якості альтернативного варіанту компілятор З# дозволяє використати груповий символ(*) для включення в поточне складання усіх файлів *.cs, які містяться в каталозі проекту :

```
csc /r: System.Windows.Forms.dll *.cs
```

У компіляторе C# підтримується використання так званих файлів (response files) у відповідь. У файлах у відповідь C# розміщуються усі інструкції, які повинні використовуватися в процесі компіляції поточного складання. Це дозволяє уникнути необхідності безліч разів вводити одні і ті ж параметри. За угодою ці файли мають розширення *.rsp (скорочення від response — відповідь).

Приклад файлу у відповідь на ім'я TestApp.rsp (коментарі в даному випадку позначаються символом #):

```
# Це файл у відповідь для прикладу
# TestApp.exe з глави 2.
# Посилання на зовнішні складки:
/r: System.Windows.Forms.dll
# Параметри виведення і що підлягають компиляции файли
# (тут використовується груповий символ):
/target: exe /out: TestApp.exe *.cs
```

Тепер усе застосування можна буде створити таким чином(зверніть увагу на застосування символу @):

```
csc @TestApp.rsp
```

У разі потреби допускається також вказувати і декілька відповідей *. rsp файлів в якості вхідних параметрів(наприклад, `esc @FirstFile.rsp @SecondFile.rsp @ThirdFile.rsp`).

Ще одним моментом, пов'язаним з файлами у відповідь, про яке необхідно згадати, являється те, що з компилятором C# асоційований файл у відповідь csc.rsp, який використовується за умовчанням і розміщений в тому ж самому каталозі, що і файл esc .exe (звичайно це C:\Windows\Microsoft.NET\Framework\<Версія>). У ньому за допомогою прапора /r : вказана безліч зборок .NET.

Для відключення функції автоматичного читання файлу esc. rsp вкажіть опцію /noconfig:

```
csc @TestApp.rsp /noconfig
```

Компілятор C/C++ cl.exe

[http://people.scs.carleton.ca/~dehne/projects/cpp-doc/compiler/vccl.html]

У підкаталозі BIN (тому, який VCVARS32 включає в PATH) є корисна утиліта, наш компілятор, під назвою CL.EXE спроектований, щоб складати програми з командного рядка. Його стандартний виклик має наступний формат:

```
CL option(s) file(s) @command_file /link link_options
```

де тільки обов'язковий параметр — файл(s), який має бути списком початкових кодових файлів і бібліотек, які ми хочемо скомпілювати. Типи файлів будуть розрізнятись згідно з їх розширеннями, отже:

с буде розглядатись як файли початкового коду мови C, і
cpp і cxx буде розглядатись файлами початкового коду мови C++.

якщо ми хочемо користуватися іншими розширеннями для наших початкових кодових файлів, нам доведеться передувати імені файлу /Tc для файлів на мові C і /Tp для файлів, написаних на мові C++ (без пробілу між цими ознаками і ім'ям файлу).

Параметр option(s) може піти змішано між файлами, або перед ними, або після них. Це параметри компіляції вибір і їх є багато, головним чином різні види оптимізацій, я не збираюся деталізувати їх (консультуйтеся з компілятором допомагають друкуванню CL /? для повного списку, або MSDN).

Command_file параметр використаний, щоб задати ім'я файлу, якому передує знак(@), і який містить інші параметри командного рядка. CL вставить вміст цього файлу в пункт командного рядка, де ви задаєте його ім'я нібито ви надрукували вміст файлу.

Нарешті /link використаний, щоб задати параметри для редактора зв'язків, який автоматично визивається CL, якщо не вказаний параметр /c. Він служить, щоб задати тип виконуваного файлу, який ми хочемо створити, розташування бібліотек і "інклуд-файлів" і інші параметри редактора зв'язків наприклад рівня відладки.

Отже, якщо ви хочете скомпілювати файл початкового коду C++ в програму, наприклад подібно до test.cpp:

```
#include <iostream>

int main()
{
    std::cout << "This is a native C++ program." << std::endl;
    return 0;
}
```

(і ви вже виконали VCVARS32.BAT). Досить написати в командному рядку:

```
CL test.cpp
```
щоб отримати файл test.exe.

Командні файли CL
[http://msdn.microsoft.com/ru-ru/library/x2khzsa1%28v=vs.110%29.aspx]

Командний файл — текстовий файл, який містить параметри і імена файлів, які ви повинні задавати в командному рядку, або задавати користуючись змінною середовища CL . CL приймає командний файл компілятора як аргумент в змінній середовища CL або в командному рядку. На відміну від командного рядка або змінної середовища CL, командний файл дозволяє вам користуватися кількома рядками параметрів і імен файлів.

Опції і імена файлів в командному файлі обробляються згідно з розташуванням імені командного файлу в змінній середовища CL або в командному рядку. Проте, якщо в командному файлі з'являється параметр /link,

всі опції в наступній частині до кінця рядка передаються до редактора зв'язків. Параметри в наступних рядках в командному файлі і параметри в командному рядку після виклику командного файлу знову приймаються як опції компілятора.

Командний файл не повинен містити команду CL. Кожний параметр повинен початися і закінчитися в тому ж рядку; ви не можете користуватися зворотним слешер (\), щоб комбінувати параметри з двох рядків.

Командний файл вказується знаком (@), за яким слідує ім'я файлу; ім'я файлу може задаватись абсолютним або відносним шляхом.

Наприклад, якщо наступна команда знаходиться у файлі з назвою RESP:

```
/Og /link LIBC.LIB
```

і ви конкретизуєте наступну CL команду:

```
CL /Ob2 @RESP MYAPP.C
```

команда до CL буде такою, як зазначено нижче:

```
CL /Ob2 /Og MYAPP.C /link LIBC.LIB
```

До класу трансляторів також відносяться

- препроцесори
- лінкувальники/завантажувачі. Вже на самому початку розвитку методів програмування став застосовуватися простий і ефективний прийом виділення часто використовуваних алгоритмів в самостійні програми, що дістали назву стандартних підпрограм. Прикладом можуть служити підпрограми обчислення елементарних функцій(синус, косинус та ін.), а також процедури обміну із зовнішніми облаштуваннями комп'ютера. Одного разу складені і такі, що відкомпілювалися, вони надалі можуть застосовуватися програмістами у своїх завданнях шляхом під'єднування їх до розробленого коду основного алгоритму. У систему засобів програмування входить програма, що називається *редактор зв'язків* (компонувальник, лінкувальник), яка забезпечує пошук допоміжних підпрограм в спеціальних бібліотеках програм і їх приєднання до основної програми користувача. Результатом роботи редактора зв'язків є повністю готовий до виконання двійковий код програми, що називається *завантажувальним модулем*.

Лінкувальник C/C++ link.exe

- генератори коду (за виключенням, можливо, об'єктно-орієнтованих засобів проектування, що підтримують зв'язок з вихідним кодом і мають тенденцію бути тісно інтегрованими з новим поколінням IDE). В деяких випадках генератори самостійно створюють програмний код, що виконує певні стандартні дії. Прикладом таких засобів може служити, наприклад, Microsoft Visual Studio, яка автоматично створює і заповнює полями клас "Форма" при створенні нами нового вікна і наповненні його різними компонентами.

Генератор образів в машинному коді Ngen.exe

Інтерпретатори

Ці інструменти забезпечують виконання програм за допомогою емуляції. Вони можуть підтримувати дії з конструювання програмного забезпечення, надаючи для виконання програм оточення, що більше контрольоване і піддається спостереженню, чим це зазвичай здатна зробити та або інша операційна система.

Хочеться відмітити певне "злиття", якщо так можна виразитися, між компіляторами і інтерпретаторами. Яскравим тому свідченням є використання так званої just-in-time компіляції – компіляції "на льоту", коли проміжний програмний код, у міру виконання або з випередженням (наприклад, в процесі запуску/завантаження програми) перетворюється в набір інструкцій, що виконуються безпосередньо засобами операційної системи, але під контролем середовища виконання, в першу чергу, з точки зору безпеки. Такого роду підхід став родоначальником ряду сучасних програмних платформ, наприклад, Java і .NET. На цьому фоні можливо об'єднати інтерпретатори з компіляторами і генераторами коду, як засоби безпосередньої підготовки(трансляції) початкового коду до виконання.

Відлагоджувачі

У систему засобів програмування входять також програми, що полегшують відлагодження (пошук помилок, зневадження). При усьому різноманітті реалізацій відлагоджувачів їх основні можливості полягають в так званому трасуванні роботи програми. Трасування – це відстежування (ведення протоколу) роботи програми. В процесі трасування програміст може простежити порядок виконання операторів, а також динаміку зміни значень змінних програми.

Відлагоджувач Visual Studio

Відлагоджувач, вбудований в Visual Studio, є одним з найбільших і складних інструментів інтегрованого середовища розробки.

Можливості Microsoft Visual Studio Debugger [5]:

- Повна символьна і сирцева інтеграція.
- Прив'язування до і відв'язування від процесів.
- Інтегроване зневадження програм, написаних на мовах .NET і природних мовах для Windows, (наприклад, виклики з C# в C++).
- Можливість зневадження з віддаленої машини.
- Повна підтримка C++, включаючи шаблони і стандартну бібліотеку.
- Зневадження веб-сервісів ASP.NET.
- Зневадження всередині коду DLL, якщо доступна символьна інформація відлагоджувача.
- Єдиний стандарт для просунутих можливостей точок зупину, включаючи умови, адреси, дані.

- Безліч способів представлення стану програм і даних, включаючи кілька вікон перегляду, потоки, стек викликів та модулі. Відображення використовуваної бібліотеки та користувацьких типів даних можна налаштовувати (наприклад, для показу вмісту контейнерного класу докладніше, ніж просто показувати його основну структуру)
- Скриптовість або можливість керувати за допомогою макросів або скриптів. Може застосовуватися будь-яка мова, яка може взаємодіяти з COM.
- Підтримка принципу «Виправ і продовжуй» (*Edit and continue*), що дозволяє змінювати сирцевий код і його перекомпіляцію без перезапуску програми (тільки для 32-бітних застосунків)
- Локальне та віддалене зневадження збережених процедур SQL на підтримуваних версіях Microsoft SQL Server.

Меню і панель інструментів [2]

Меню Debug і відповідна панель інструментів надають доступ до запуску сеансів відлагодження, покрокового проходження коду, управлінню точками останову, а також і до багатьох функціональних можливостей відлагодження в Visual Studio. Є два стани меню відлагодження: стан спокою (неактивне) і режим відлагодження. На рис. 4 показано меню в стані спокою.

У стані спокою меню Debug надає можливості для запуску сеансу відлагодження, прикріплення до процесу, що виконується, і для доступу до деяких з налагоджувальних вікон.

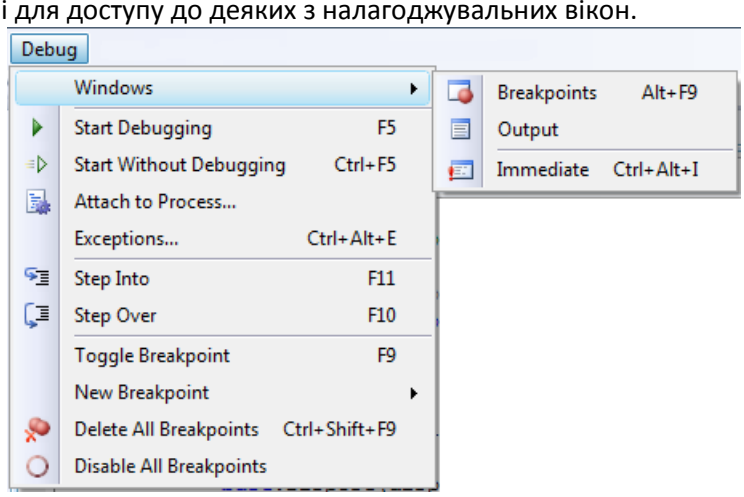

Рис. 4. Меню Debug в стані спокою

Коли відлагоджувач включений і ви працюєте в сеансі відлагодження, то стан меню Debug змінюється. Тепер в ньому є декілька додаткових опцій (окрім тих, які є в стані спокою). Ці опції включають: функції переміщення по коду, перезапуск сеансу і доступ до додаткових вікон відлагодження. На рис. 5 показано меню Debug під час сеансу відлагодження.

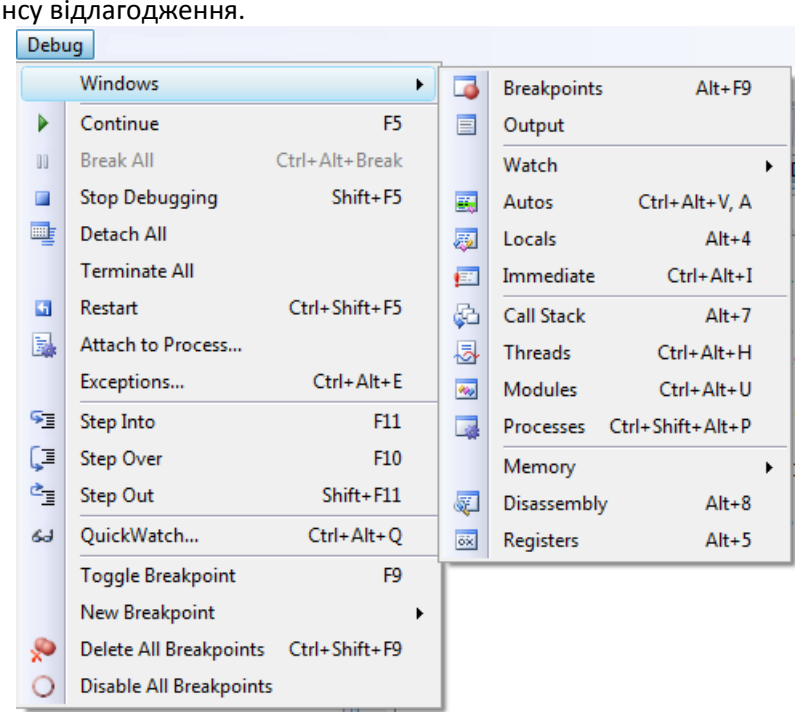

Рис. 5. Меню Debug під час сеансу відлагодження

На рис. 6 показана панель інструментів Debug. Панель інструментів Debug дає швидкий доступ до деяких ключових елементів меню Debug. Тут ви можете управляти вашим сеансом відлагодження. Наприклад, ви можете почати або продовжити сеанс відлагодження, зупинити сеанс, що виконується, виконати покроковий прохід коду і т. д.

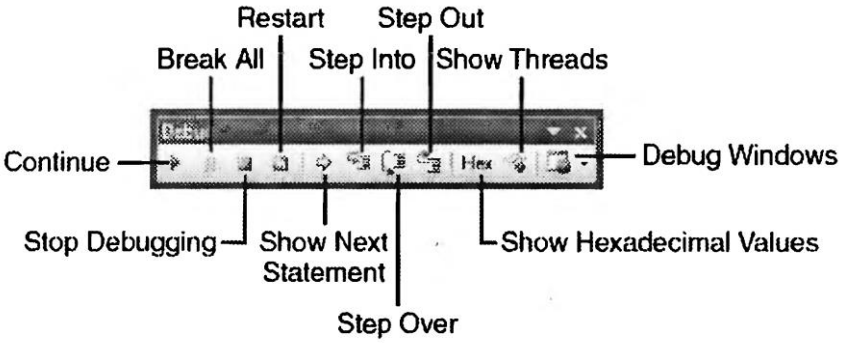

Рис. 6. Панель інструментів Debug

Вхід в код, вихід з коду і пропуск коду

Ймовірно, звичайнісінькою налагоджувальною операцією для розробника є покроковий прохід по рядках коду і вивчення даних, що видаються додатком і відлагоджувачем. Покрокове проходження саме в цьому і полягає: вивчення рядка, виконання рядка, вивчення результатів (а потім повторення цього процесу знову і знову).

Початок сеансу відлагодження (вхід в код)

Команда **Step Into** доступна в меню Debug і на панелі інструментів (ви можете також натиснути клавішу <F11>). З цією командою пов'язані два типи поведінки. Перший — це коли ви викликаєте цю команду для додатка, який в даний момент не виконується в режимі відлагодження. В цьому випадку додаток відкомпілюється і запуститься, і у вікні відлагодження ви отримаєте перший рядок для покрокового проходження коду. Це (по суті) вхід в код вашого застосування. На рис. 7 показаний додаток Windows Forms в режимі відлагодження (в результаті виклику Step Into).

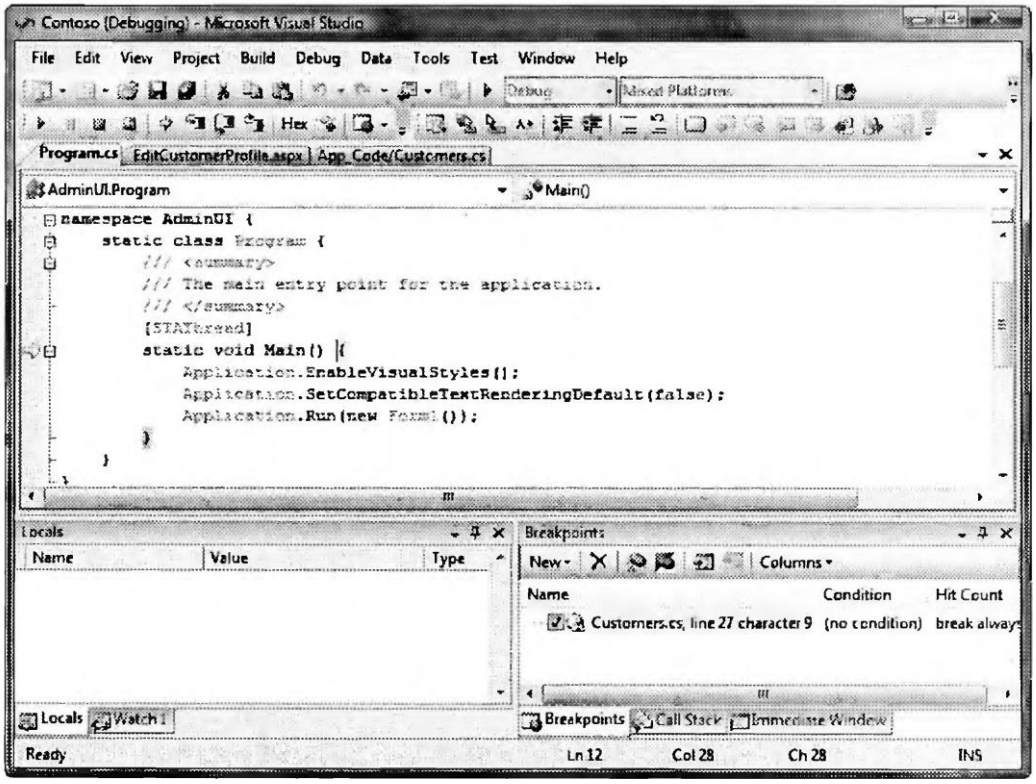

Рис. 7. Використання виклику Step Into для запуску відлагодження

Виклик команди **Step Over** (меню Debug, панель інструментів або клавіша <F10>) у той момент, коли ваше застосування знаходиться в стані спокою, приведе до того ж самого, що і виклик Step Into. Тобто ваше застосування відкомпілюється і буде запущене в сеансі відлагодження на перший рядок коду.

Однією із зручніших функціональних можливостей набору інструментів відлагодження є функція **Run to Cursor** (Виконати до курсора). Вона працює в повній відповідності зі своєю назвою. Ви встановлюєте курсор на деякий код і викликаєте цю команду. Додаток компілюється і виконується до тих пір, поки не доходить до того рядка, де знаходиться курсор. У цій точці відлагоджувач перериває додаток і видає вам цей рядок коду для покрокового проходження.

Ви можете також розпочати ваш сеанс відлагодження з вибору опції **Start Debugging** — зелена стрілка «плей» — в меню Debug або панелі інструментів (чи натиснення клавіші <F5>). При цьому почнеться сеанс відлагодження, але вихід в код не станеться (якщо тільки не станеться виключення або не попадеться точка останову).

Якщо ваше застосування виконується, і ви хочете увійти до режиму останову, то ви можете зробити це у будь-який час за допомогою команди **Break All** (Перервати все) з меню Debug або панелі інструментів, або за допомогою комбінації клавіш <Ctrl>+<Alt>+<Break>.

Функція Break All представлена на панелі інструментів значком з символом паузи. Натиснення цієї кнопки зупиняє ваше застосування на наступному рядку, що виконується, і дозволяє вам отримати інформацію з відлагоджувача. Команда Break All особливо корисна у тому випадку, коли вам треба перервати тривалий процес або цикл, який "підвісив" ваше застосування.

Проходження по коду

Команда **Step Into** (Вхід) — <F11> для C# — дозволяє вам просуватися за вашим кодом по одному рядку. Виклик цієї команди виконає поточний рядок коду і помістить ваш курсор на наступний виконуваний рядок. Важлива відмінність між Step Into і іншими схожими командами полягає в тому, як Step Into обробляє рядки коду, в яких містяться виклики методів. Якщо ви знаходитеся на рядку коду, яка викликає інший метод вашого рішення, то виклик Step Into перенесе вас на перший рядок цього методу (за умови, що у вас завантажені відповідні налагоджувальні символи).

Команда **Step Over** (Пропуск) — <F10> для C# — дозволяє вам зберігати фокус в поточній процедурі (не заходячи в методи, що викликаються нею). Тобто виклик Step Over приведе до виконання рядка за рядком, але не заведе вас у виклики функцій, конструктори або виклики властивостей.

Команда **Step Out** (Вихід), що викликається натисненням комбінації клавіш <Shift>+<F11> для C# — це ще один корисний інструмент. Він дозволяє вам дати вказівку відлагоджувачу закінчити виконання поточного методу (який ви відлагоджуєте) і повернутися в режим останову відразу після його завершення. Це дуже зручно тоді, коли ви повгрузали в довгому методі, який варто було б пропустити. Крім того, ви можете увійти до цієї функції для відлагодження тільки її частини, а потім вийти з неї.

Коли ви знаходитеся в сеансі відлагодження, то команда Start Debugging (чи Run) змінюється на Continue. Команда Continue доступна тоді, коли ви призупинили виконання на рядку коду у відлагоджувачі. Вона дає вам можливість продовжити виконання додатка без покрокового проходження по рядках.

Закінчення сеансу відлагодження

Ви можете закінчити сеанс відлагодження декількома способами. Один з самих часто використовуваних методів — це припинити виконання додатка. Це можна зробити за допомогою натиснення кнопки Close (чи x) вікна додатка Windows. При завершенні додатка станеться також і завершення сеансу відлагодження.

Є також пара способів і у вікні Debug. Команда **Terminate All** завершує усі процеси, до яких прикріплений відлагоджувач, і завершує сеанс відлагодження. Є також опція **Detach All**. Detach All просто відкріпляє відлагоджувач від усіх процесів, що виконуються, без їх завершення. Ця можливість корисна тоді, коли ви тимчасово прикріплялися до процесу, що виконується, відлагодили його і хочете залишити його працюючим.

Вказівка місць виходу в код

Ви управляєте відлагоджувачем за допомогою точок останову і точок відстежування. З їх допомогою ви можете вказати відлагоджувачу, де вам треба вийти в код або отримати інформацію про ваше застосування. Точки останову дозволяють вам вказати, коли відлагоджувач повинен зупинитися на певному рядку коду. Точки відстежування були введені в Visual Studio 2005. Це такий тип точки останову, який дозволяє вам виконати певну дію тоді, коли досягнутий вказаний вами рядок коду. Зазвичай при цьому відбувається виведення даних про ваше застосування у вікно виведення.

Налаштування точки останову

Самий часто використовуваний спосіб налаштування точки останову: спочатку потрібно знайти рядок кода, на якому ви хочете зупинити відлагоджувач, потім ви клацаєте по цьому рядку в полі індикаторів редактора коду. При цьому в полі індикаторів з'являється червоний кружок, і рядок коду виділяється червоним кольором.

Є ще декілька додаткових способів налаштування точок останову. Наприклад, ви можете клацнути по рядку коду правою кнопкою миші і вибрати пункт **New Breakpoint** в контекстному меню Breakpoint. Ви можете також вибрати команду New Breakpoint в меню Debug (чи натиснути <Ctrl>+<D>, <N> у C# або <Ctrl>+<B> у VC++). Цей варіант активує діалогове вікно New Breakpoint, в якому ви можете настроїти точку останову функції.

Вікно точок останову Breakpoints

Вікно Breakpoints в Visual Studio дає зручний спосіб організації і управління безліччю умов, по яких ви хочете вийти у відлагоджувач. Ви дістаєте доступ до цього вікна через меню Debug або панель інструментів (чи за допомогою натиснення <Ctrl>+<D>, <B> у C# або <Alt>+<F9> у VC++). На рис. 8 показано вікно Breakpoints усередині Visual Studio.

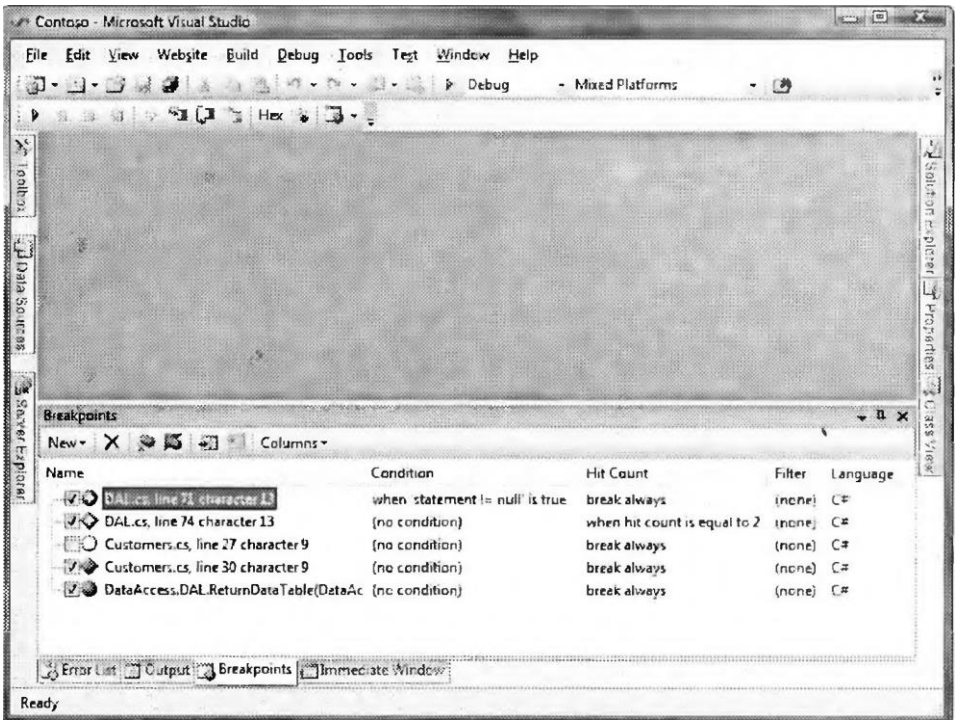

Рис. 8 Вікно Breakpoints

Вікно Breakpoints дає доступ до кожної точки останову. Ви можете деактивувати окрему точку останову за допомогою зняття галочки біля точки останову в списку. Ви можете налаштувати сукупність властивостей і умов, пов'язаних з точкою останову.

Переривання на основі умов

Часто однієї установки простої точки останову буває недостатньо (або вона неефективна). Наприклад, якщо ви шукаєте виконання у вашому коді певної умови (яке, можливо, викликає винятковий стан), то вам краще робити останов по цій умові. Це заощадить час на постійні входи у функції, при яких ви тільки вивчаєте декілька елементів даних і бачите, що ваша умова не виконана.

Є п'ять типів умов, які ви можете додати до точки останову: **Location**, **Condition**, **Hit Count**, **Filter** і **When Hit**. Ви додаєте умову потрібного типу до точки останову у вікні Breakpoints за допомогою контекстного меню для цієї точки останову.

Налаштування умови точки останову

Умова точки останову дозволяє вам вийти у відлагоджувач (чи виконати деяку дію у разі точки відстежування) тоді, коли деяка умова буде або виконано, або зміниться значення виразу.

Для налаштування умови виділіть точку останову, для якої ви хочете додати умову. Потім виберіть пункт Condition з контекстного меню (через клацання правою кнопкою миші). Це активує діалогове вікно **Breakpoint Condition** (рис. 9).

При налаштуванні умови у вас є два варіанти: **Is true** і **Has changed**. Варіант Is true дозволяє вам настроїти логічну умову, при виконанні якого станеться вихід з відлагоджувача на відповідний рядок коду.

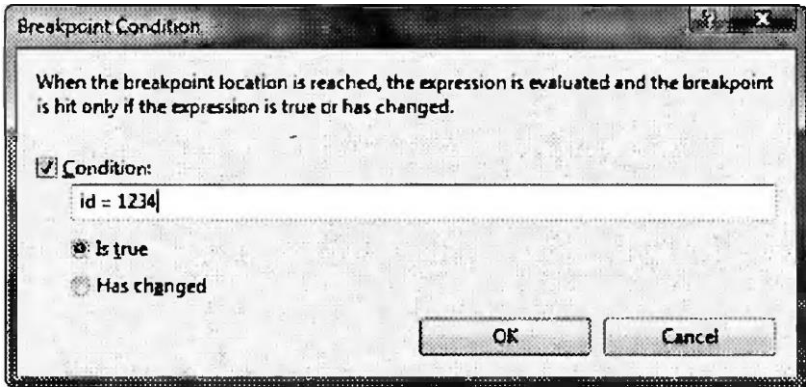

Рис. 9. Діалогове вікно Breakpoint Condition

Налаштування фільтру точки останову

Фільтри точок останову дозволяють вам вказати конкретний комп'ютер, процес або потік, в якому повинен статися останов. Наприклад, якщо ваша помилка відбувається тільки на певному комп'ютері або в певному процесі, то ви можете відлагоджувати саме цю умову (за допомогою фільтру). Фільтри найбільш корисні в складних сценаріях відлагодження, коли ваше застосування має яскраво виражений розподілений характер.

Використання з точкою останову лічильника кількості попадань

Використовуючи команду **Hit Count**, ви повідомляєте відлагоджувач, що хочете перервати виконання тоді, коли цей рядок коду виконається певна кількість разів. Зазвичай можна знайти зручнішу умову останову, чим Hit Count. Проте ця функція корисна в тих випадках, коли ви не можете вказати реальну умову, але знаєте, що коли ви проходите через функцію певну кількість разів, то починаються проблеми. Крім того, опція Hit Count може бути кориснішою в сценаріях з точками відстежування, коли ви видаєте дані про те, що відбувається у вашому коді. Можливо, вам зручно видаватиме ці дані тільки час від часу.

*Точки відстежування (опція **When Hit**)*

Точки відстежування дозволяють вам видати дані у вікно Output або виконати макрос для Visual Studio у тому випадку, коли зустрілася певна точка останову. Після цього ви можете вийти у відлагоджувач (як і у випадку із звичайною точкою останову), обробити іншу умову або просто продовжити виконання додатка. Ця можливість може бути дуже корисною тоді, коли ви хочете підтримувати журналювання усього того, що відбувається у вашому застосуванні при відладці. Потім ви можете проглянути цей журнал, щоб отримати цінну інформацію про конкретні умови і порядок виконання (коли відбувається виключення).

Ви можете настроїти точки відстежування явним чином (за допомогою клацання правою кнопкою миші по рядку коду і подальшого вибору пункту **Insert Tracepoint** в меню Breakpoint). Крім того, вибір команди When Hit з контекстного меню (див. рис. 8) точки останову у вікні Breakpoints активує діалогове вікно точки відстежування When Breakpoint Is Hit (рис. 10).

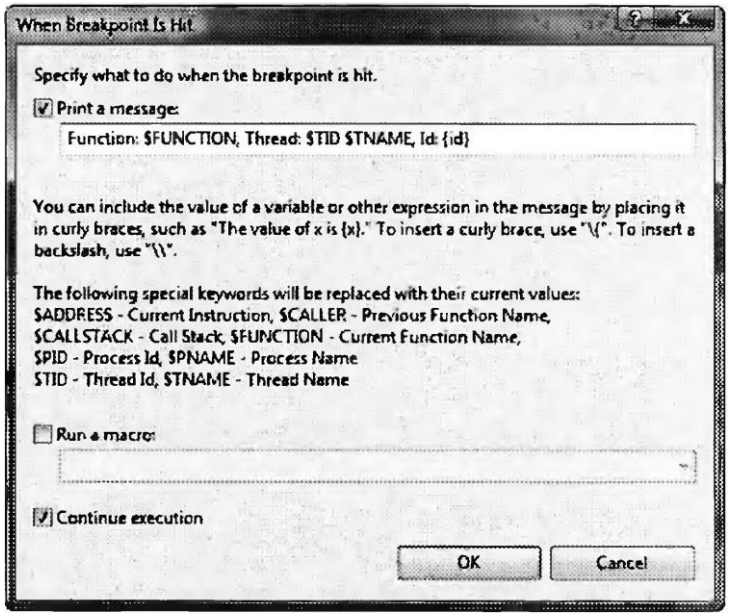

Рис. 10. Налаштування точки відстежування

Перегляд даних у відлагоджувачі. Спостереження за змінними

Звичайною справою в сеансі відлагодження є спостереження за значеннями типів вашого застосування. Для цього існує багато вікон. Два найочевидніших — це вікна **Locals** і **Autos**.

Вікно Locals

Вікно **Locals** показує усі змінні і їх значення для поточної зони видимості відлагоджувача. Це дає вам уявлення про все, що є в поточному методі, який виконується. Змінні в цьому вікні автоматично настроюються відлагоджувачем. Вони організовані в список по імені в алфавітному порядку. Крім того, ієрархія також показана. Наприклад, якщо ця змінна має відношення до об'єктного типу, то члени цього об'єкту перераховані усередині змінної (у вигляді деревовидної структури).

Вікно Autos

Дуже часто перегляд усіх локальних змінних дає надто багато інформації, щоб в ній можна було розібратися. Так може відбуватися тоді, коли в зоні видимості цього процесу або функції знаходиться надто багато всього. Для того, щоб,побачити значення, пов'язані з тим рядком коду, на яку ви дивитеся, ви можете використати вікно **Autos**. Це вікно показує значення усіх змінних і виразів, наявних в поточному рядку коду, що виконується, або в попередньому рядку коду. Це дозволяє вам дійсно зосередитися тільки на значеннях, які ви в даний момент відлагоджуєте.

Вікна Watch

Вікна контрольних значень в Visual Studio дозволяють вам настроювати власний список змінних і виразів, за якими ви хочете спостерігати. Саме тут ви визначаєте ті елементи, які вас цікавлять. Вікна **Watch** виглядають і поводяться точно так, як і вікна Locals і Autos. Крім того, ті елементи, які ви розміщуєте у вікнах Watch, зберігаються між сеансами відлагодження.

Ви дістаєте доступ до вікон Watch з меню або панелі інструментів Debug. Чотири вікна Watch мають імена Watch 1, Watch 2, Watch 3 і Watch 4. Чотири вікна Watch дозволяють вам настроїти чотири списки елементів, за якими ви хочете спостерігати. Ця можливість може бути особливо корисна у тому випадку, коли кожен список відноситься до окремої зони видимості вашого застосування.

Змінну або вираження у вікно Watch ви додаєте або з редактора коду, або з вікна QuickWatch. Якщо ви знаходитеся в редакторові коду, то виділяєте змінну (чи вираз), клацаєте по ній правою кнопкою миші і вибираєте пункт Add Watch. При цьому виділена змінна (чи вираження) буде поміщена у вікно Watch. Ви можете також перетягнути виділений елемент у вікно Watch.

DataTips

DataTips (спливаючі підказки даних) дозволяють вам виділити змінну або вираз у вашому редакторові коду і отримати інформацію спостереження прямо в редакторові. Ця функція добре відповідає методу роботи розробника. Наприклад, якщо ви дивитеся на рядок коду, то можете виділити щось в цьому рядку для обчислення. Ви могли б зробити це і за допомогою створення QuickWatch. Проте ви можете також просто навести курсор на деякий елемент, і його дані розгорнуться в DataTip.

Відлагоджувач командного рядку cordbg.exe [6]

…

Інтегровані середовища розробки [7]

Інтегроване середовище розробки, ICP (IDE, Integrated development environment або Integrated debugging environment) – система програмних засобів, яка використовується програмістом для розробки програмного забезпечення.

Зазвичай ICP об’єднують множину інструментів з широким набором функцій для автоматизації і підвищення продуктивності праці програміста:

- текстовий редактор;
- компілятор/інтерпретатор і засоби автоматизації збирання;
- зневаджувач.

ICP також може включати:

- засоби інтеграції з системами управління версіями;
- інструменти конструювання графічного інтерфейсу;
- браузер класів;
- інспектор об’єктів;

- візуальний конструктор (діаграму) класів;
- інтеграцію з веб-серверами (або власні веб-сервери).

ІСР розробляються, щоб максимізувати продуктивність програміста, надаючи набір зв'язаних інструментів об'єднаних єдиним інтерфейсом користувача. Це повинно означати, що програмістові необхідно робити менше переключень між різними режимами роботи в порівнянні з використанням дискретних програмам розробки. Проте, оскільки ІСР – досить складний пласт програмного забезпечення за своєю природою, це підвищення продуктивності відбувається тільки після довгого повчального процесу.

Зазвичай ІСР присвячене специфічній мові програмування, включаючи більшість особливостей, які ближче всього відповідають парадигмам програмування певної мови. Однак, деякі ІСР багатомовні, як наприклад Eclipce, ActiveState Komodo, IntelliJ IDEA, Oracle JDeveloper, останні версії NetBeans, Microsoft Visual Studio, Genuitec MyEclipse, WinDev і Xcode.

ІСР зазвичай являють собою єдину програму, в якій виконується уся розробка. Ця програма зазвичай забезпечує багато функцій для написання, модифікації, компілювання, розгортання і відлагодження програмного забезпечення. Мета – абстрагувати (спростити) процес конфігурування необхідний, щоб об'єднати утиліти командного рядка в єдиний інструмент, що теоретично скорочує час, необхідний на вивчення мови, і підвищує продуктивність розробника. Вважається також, що тісне інтегрування робіт при розробці програмного забезпечення може також збільшити продуктивність. Наприклад, код може бути аналізований в процесі написання, забезпечуючи миттєвий зворотну реакцію на синтаксичні помилки. В той час, коли сучасні ІСР графічні, ІСР які використовувались до появи віконних систем (як наприклад Microsoft Windows або X Windows (X11)) були текстовими з функціональними клавішами або гарячими клавішами, щоб виконувати різні завдання (Turbo Pascal наприклад). Все це контрастує з розробкою програмного забезпечення, користуючись незв'язаними інструментами, як наприклад vi, GCC або make.

Історія

Виникнення ІСР стало можливим з появою можливості роботи через консоль або термінал. Більш ранні системи не могли підтримувати ІСР, оскільки програми готувались з використанням блок-схем і вводились за допомогою перфокарт (чи перфострічки, і т.п.) перед тим як потрапити на вхід компілятора компілятора. Dartmouth BASIC (розроблена в Dartmouth College, Гановер, Нью-Гемпшир, США) була першою мовою, яка створена з ІСР і також першою мовою, призначеною для інтерактивного використання, сидячи перед консоллю або терміналом (перша інтерактивна версія Dartmouth BASIC стала доступною для користувачів в червні 1964 р.). Її ІСР (частина системи розділення часу Dartmouth, Dartmouth Time Sharing System) було командним, і тому не нагадувало більшість керованих за допомогою меню, графічних ІСР, що переважають сьогодні.

Будь-який рядок надрукований користувачем, і розпочинався з номера рядка, додавався до програми, замінюючи будь-який збережений раніше рядок з тим же номером; все інше вважалося командою DTSS і відразу виконувалось виконувався. Рядки, які складалися виключно з номера рядка, не зберігались але стирали будь-який раніше збережений рядок з тим же номером. Цей метод редагування був необхідним, оскільки в якості терміналів для Dartmouth Timesharing System використовували телетайпи (*teletype*, TTY).

Проте ІСР вперше Dartmouth BASIC об'єднало редагування, управління файлами, компіляцію, відлагодження і виконання аналогічно з сучасними ІСР.

Maestro I – продукт Softlab Munich, був першим всесвітньо розповсюдженим інтегрованим середовищем розробки для програмного забезпечення, представленим в 1975 р. Maestro I був встановлений для 22,000 програмістів у всьому світі. Maestro I був можливо світовим лідером в цьому полі впродовж 1970-х і 1980-х.

Один з перших ІСР, що підтримувало ідею плагінів, що підключаються, був Softbench.

Вдношення до ІСР на різних обчислювальних платформах

Багато програмістів *Unix* стверджують, що, традиційні POSIX інструменти командного рядка складають закінчене середовище розробки, хоч і з іншим стилем інтерфейсу і під керуванням Unix. Багато програмістів все ще користуються makefiles і їх похідні. Також, багато програмістів Unix користуються Emacs чи Vim, які повністю підтримують багато із стандартних інструментів Unix для побудови ПЗ. Data Display Debugger використовується як потужний графічний інтерфейс для багатьох стандартних текстових зневаджувальних інструментів.

І все ж існує багато прикладів ІСР під *Unix*: Anjuta, NetBeans, Geany, KDevelop, Qt Creator, Code::Blocks.

На різних платформах *Microsoft Windows*, інструменти розробки командного рядка рідко використовуються. Відповідно, є багато комерційних і некомерційних рішень, проте кожен має свій відмінний дизайн зазвичай, створюючи несумісності. Найголовніші продавці компіляторів для Windows також

пропонують безкоштовні копії їх інструментів командного рядка, у тому числі Microsoft (Visual C++, Platform SDK, .NET Framework SDK, nmake), Embarcadero Technologies (компілятор bcc32, make).

Додатково на багатьох платформах, у тому числі Windows, доступні безкоштовні програмні засоби GNU (GNU Compiler Collection (gcc), GNU Debugger (gdb), GNU make).

ІСР завжди були популярні на *Mac OS*, Apple Macintosh, ще з часів Macintosh Programmer's Workshop, Turbo Pascal, THINK Pascal and THINK C в середині 1980-их. Зараз Mac OS X програмісти можуть вибирати з обмеженого набору ІСР, включаючи рідні ІСР подібно до Xcode, старшим ІСР подібно до CodeWarrior, новішим ІСР подібно до MyEclipse, і загальнодоступних інструментів, як наприклад Eclipse і Netbeans. ActiveState Komodo – комерційне ІСР, підтримуване на Mac OS.

Нещодавно, з появою хмарних обчислень, ІСР почали переміщення в он-лайн і працювати в веб-браузерах; один приклад такого ІСР – Cloud9 IDE (standard – безкоштовне, premium), CodeRun IDE (безкоштовне).

Eclipce [8,9]

Eclipse являється open - source інтегрованим середовищем розробки(IDE). Спочатку, Eclipse призначався для використання Java- розробниками, проте, оскільки користувачі можуть розширити свої можливості за рахунок установки численних плагінів, Eclipse широко використовується професійними розробниками різних напрямів. Наприклад, є плагіни для C і C++ (CDT- проект), Perl, PHP, ColdFusion, Ruby, Python і C #. В силу безкоштовності і високої якості, Eclipse у багатьох організаціях є корпоративним стандартом для розробки додатків.

Історія Eclipse

Розробка Eclipse почалася в квітні 1999 року спільними зусиллями компаній OTI і IBM. Спочатку Eclipse розроблялася як наступник середовища розробки IBM VisualAge, в якості корпоративного стандарту IDE для розробки на різних мовах під платформи IBM. У жовтні 2001 року вийшов релиз 1.0, а місяцем пізніше IBM повністю відкрила початковий код Eclipse. Тоді ж була утворена рада керівників (Board of Stewards) Eclipse, покликана забезпечити розвиток проекту і створити навколо нього співтовариство розробників незалежне від IBM. До кінця 2002 роки в раду входило близько тридцяти компаній-учасників. Проте, підтримку Eclipse з боку великих компаній-розробників ПЗ істотно обмежував той факт, що увесь проект і, найголовніше, перспективи його розвитку, міцно асоціювалися з однією компанією - IBM. Дійсно, вклад IBM в створення Eclipse був настільки великий, що увесь проект неминуче сприймався як її власна розробка. Проте головним завданням самої IBM, що зробило ставку на технологію Java (а не .NET), було добитися широкого поширення якісних продуктів для розробників на цій платформі. Щоб завоювати довіру великих постачальників ПЗ, на початку 2004 року Eclipse був перетворений в некомерційну асоціацію, що гарантує, що усі технології і початковий код Eclipse залишаться відкритими і безкоштовними.

У основу Eclipse 3.0 (2003 рік) були покладені специфікації сервісної платформи OSGi (специфікація модульної системи і службова платформа для Java). З версії 3.0 Eclipse перестав бути монолітною IDE, що підтримує розширення, а сам став набором розширень.

Eclipse написана на Java, тому є незалежним продуктом, за винятком бібліотеки SWT (Standard Widget Toolkit, бібліотека з відкритим початковим кодом для розробки графічних інтерфейсів користувача на мові Java), яка розробляється для усіх поширених платформ. Бібліотека SWT використовується замість стандартної для Java бібліотеки Swing (Sun Microsystems). Вона повністю спирається на платформу(операційну систему), що пролягає нижче, що забезпечує швидкість і натуральний зовнішній вигляд призначеного для користувача інтерфейсу, але іноді викликає на різних платформах проблеми сумісності і стійкості додатків.

Опис Eclipse

Зараз Eclipse - це щось більше, ніж просто IDE, і до того не обмежений рамками Java. Самі творці Eclipse називають його платформою для розробки інтегрованих застосувань. Eclipse завоював популярність саме як платформа для розробки розширень : будь-який розробник може розширити Eclipse своїми модулями. Існують Java Development Tools(JDT), C/C++ Development Tools(CDT), що розробляються інженерами QNX спільно з IBM, і засоби для мов Ada (GNATbench, Hibachi), COBOL, FORTRAN, PHP і ін. від різних розробників. Безліч розширень доповнює середовище Eclipse менеджерами для роботи з базами даних, серверами додатків та ін.

Середовище Eclipse надає усі стандартні функції, які очікуються від професійного редактора коду, а також багато що інше. Серед підтримуваних функцій:

- рефакторинг;
- розширений пошук;
- розширений інструмент для порівняння;
- швидка навігація;

- інтеграція з системами управління версіями і завданнями(помилками);
- збереження дій(ви можете призначити дії, які выпонятся у файлі кожного разу, коли Ви його зберігаєте).

Розробка Eclipse ведеться у рамках декількох проектів, серед яких треба згадати Eclipse, Eclipse Tools, Eclipse Modeling Project. Проект Eclipse включає платформу для розробки додатків, IDE для Java, побудовану на її основі, а також засоби, необхідні розробникові додатків. У сам проект Eclipse входять декілька підпроектів, що розробляються більш менш незалежно один від одного. Серед основних підпроектів: Platform, JDT (Java development tools) і PDE (Plug - in development environment). Platform надає базові сервіси, JDT дозволяє розробляти додатки Java, а PDE - нові компоненти Eclipse.

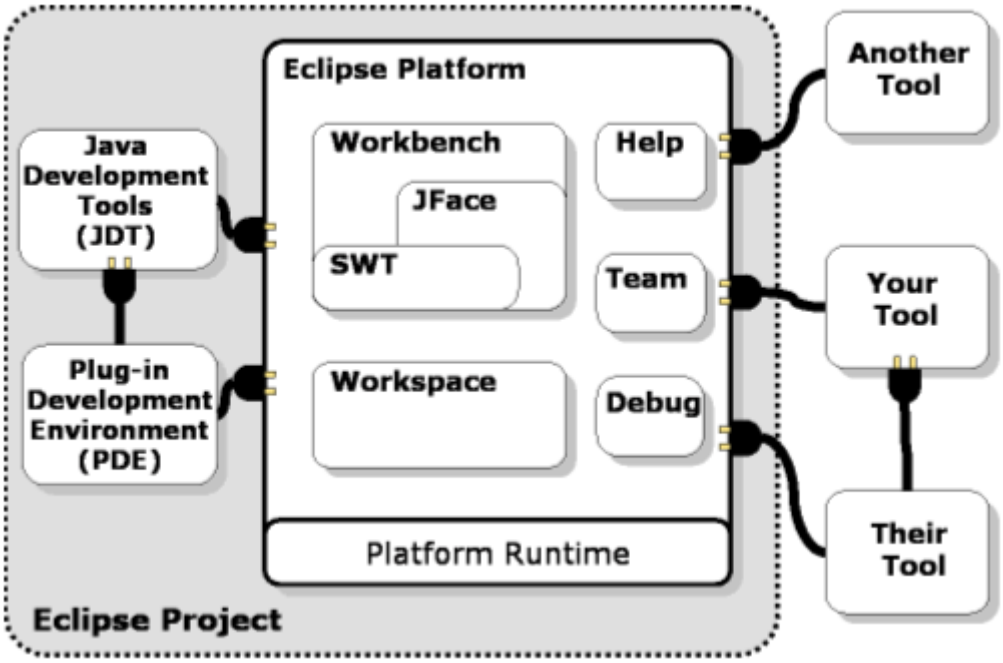

Рис. 11. Архітектура Eclipse.

Platform є ядром Eclipse. Сама по собі ця підсистема не містить особливо корисної для користувачів функціональності, але без неї неможлива робота інших підпроектів Eclipse. Сервіси, які забезпечує Platform, дозволяють розробникам визначати видимі користувачеві артефакти, створювати призначені для користувача інтерфейси, працювати з системами контролю версій, середовищами відладки і довідковою системою. Відповідними компонентами Platform, що реалізовують ці сервіси, є Workspace(управління контентом), Workbench(базовий призначений для користувача інтерфейс Eclipse), Team, Debug і Help.

Компонент **Workspace** визначає основні об'єкти, з якими можуть працювати користувачі і додатки Eclipse, структурує ці об'єкти і надає можливість управління ними. Об'єкти, які містить Workspace, збирально називаються ресурсами і розділяються на три типи: проекти, теки і файли. Плагін Resources надає API для створення, навігації і маніпуляції ресурсами в Workspace. Workbench використовує ці API для представлення цієї функціональності користувачеві.

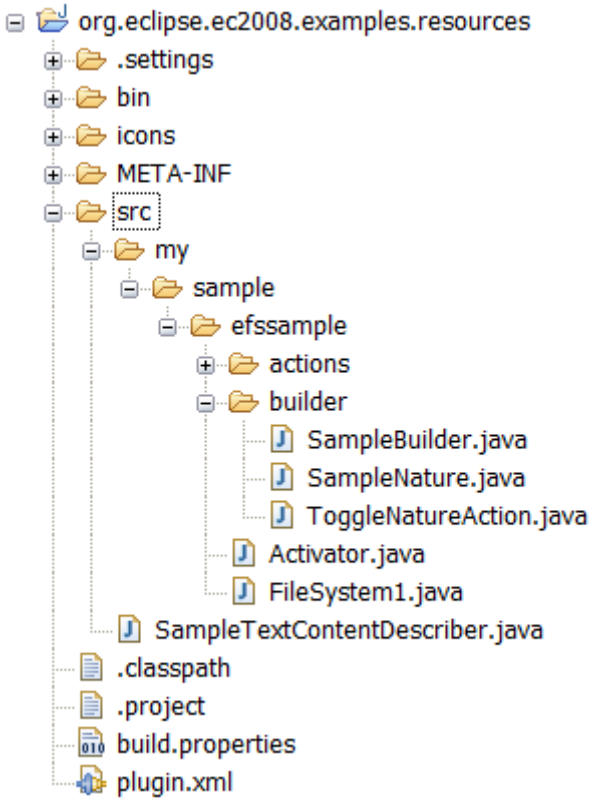

Рис. 12. Ресурси Workspace

Компонент **Workbench** визначає базовий призначений для користувача інтерфейс(UI) Eclipse, а також надає іншим компонентам засобу для створення своїх власних інтерфейсів. Основні об'єкти інтерфейсу, з якими працює Workbench, - це *редактори(editors), види(views) і перспективи(*perspectives).

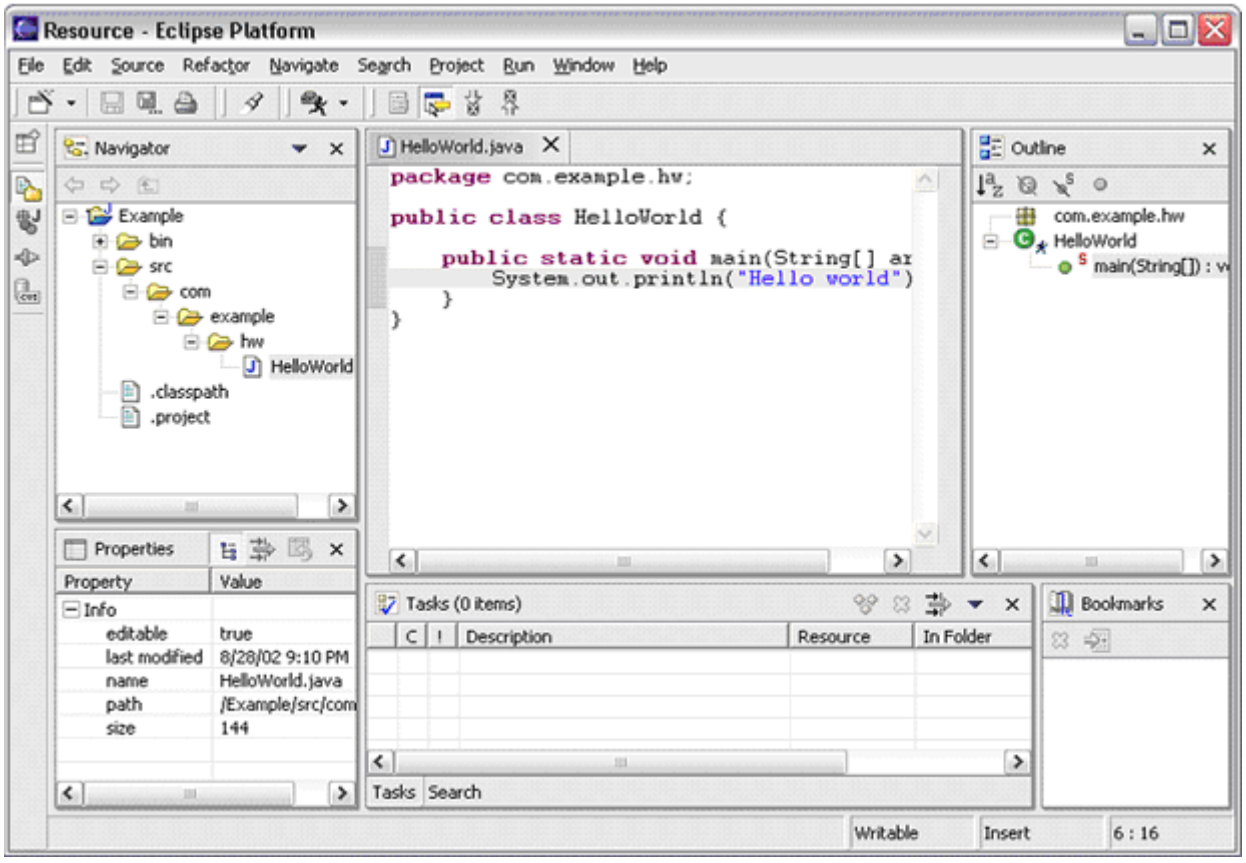

Рис. 13. Eclipse Workbench.

Редактори в Eclipse цілком відповідають інтуїтивному уявленню про те, як повинен працювати редактор, включаючи підсвічування синтаксису, контекстне автозавершення і підтримку шаблонів. Призначення

редакторів полягає в наданні можливості перегляду і редагування файлів. Візуальне представлення утримуваного файлу залежить від його типу і може мінятися від традиційного текстового редактора з підсвічуванням синтаксису для сценаріїв Ant до графічного редактора для зображень у форматі gif.

Види, на відміну від редакторів, не обов'язково пов'язані з яким-небудь конкретним ресурсом. Вони можуть бути використані для відображення утримуваного активного в даний момент редактора(на кшталт виду Outline), елементу, вибраного в іншому виді(історія версій ресурсу в CVS), а також даних, отриманих будь-яким іншим шляхом(таких, як звіт про виконання тестів JUnit).

Перспективою, в термінології Eclipse, називається конфігурація платформи, що відповідає певній ролі. Функціональність, необхідна користувачеві, що виступає в цій ролі, визначає набір і розташування вікон, меню, гарячих клавіш, і інших елементів інтерфейсу для цієї перспективи. Наприклад, в перспективі для відладки є присутніми види з поточним стеком, значеннями локальних змінних і списком точок останову, а в перспективі для синхронізації утримуваного проекту по CVS - види зі списками доступних серверів і історією змін у вибраного файлі.

Фундаментом, на якому побудовані усі графічні засоби Eclipse є компонент **SWT** , що входить в Workbench (Standard widget toolkit). SWT є основним засобом створення призначеного для користувача інтерфейсу для додатків на основі Eclipse. Оскільки для промальовування елементів інтерфейсу SWT максимально використовує засоби віконного менеджера операційної системи (Win32, GTK, Motif), додатки, побудовані на базі SWT, візуально не відрізняються від "рідних" застосувань, розроблених спеціально для конкретної операційної системи. При цьому вони зберігають сумісність з усіма платформами, підтримуваними Eclipse. Використання під кожною ОС стандартних елементів управління дозволяє істотно поліпшити продуктивність і уникнути запізнювання реакції на події в складних, багатовіконних інтерфейсах.

Eclipse JDT (Java Development Tools) – другий з основних підпроектів, складових Eclipse SDK. Це і є "IDE для Java", що містить инкрементальный компілятор Java, редактори, засоби для відладки, тестування, навігація і рефакторинга початкового коду. Оскільки Eclipse націлений на групову розробку сюди входить і інтеграція з системами управління версіями - CVS, GIT в основному постачанні, для інших систем(наприклад, Subversion, MS SourceSafe) існують плагіни. Є присутньою також підтримка зв'язку між IDE і системою управління завданнями(помилками). У основному постачанні включена підтримка трекера помилок Bugzilla, також є безліч розширень для підтримки інших трекерів(Trac, Jira та ін.).

Важливо розуміти, що JDT побудований виключно на основі підпроекту Platform і не використовує яких-небудь закритих або недокументованих інтерфейсів. Це означає, що за наявності бажання і ресурсів можна розробити компонент, функціонально аналогічний JDT, для будь-якої іншої мови - від 3++ до Python, і тим самим перетворити Eclipse, наприклад, "IDE для Python".

Ви можете повністю виключити JDT з конфігурації Eclipse і встановити тільки свої компоненти, так що користувачам вашого продукту ніщо не нагадуватиме про Java. Єдиний компонент із стандартної конфігурації, який є обов'язковим для функціонування додатків, – це Platform.

Одним з рішень, прийнятих розробниками Eclipse, була автоматична інкрементальна компіляція проекту після кожної зміни, зробленої розробником. Кожного разу, коли, працюючи в Eclipse, ви зберігаєте змінений початковий файл на диску, інкрементальний білдер аналізує зроблені зміни і запускає компіляцію не лише збереженого файлу, але і усіх залежних від нього компонентів.

Що дає такий підхід, окрім, зрозуміло, можливості оперативного перегляду і виправлення помилок компіляції? При першій компіляції проекту в пам'яті будується дерево залежностей файлів в проекті, що дозволяє надалі робити компіляцію дуже швидко, оскільки немає необхідності аналізувати усі початкові файли в проекті. Потрібно помітити, що дерево залежностей зберігається на диску, так що після перезапуску Eclipse немає необхідності проводити аналіз залежностей в проектах з нуля.

Інформація, зібрана в процесі компіляції проекту, також використовується для виконання рефакторинга і спеціалізованого Java- пошуку(посилань на декларації, оголошення класів і тому подібне). Саме доступність цієї інформації дозволяє запускати рефакторинг і пошук без затримок, які були б потрібні для її збору у разі відсутності автоматичної компіляції. (Рефакторинг, і пошук працюватимуть і для проектів, що не відкомпілювалися, проте в цьому випадку точність отриманих результатів не гарантується.)

Дистрибутив, що об'єднує разом інструменти Eclipse з початковими кодами і призначеною для користувача документацією, називається Eclipse SDK. Eclipse SDK містить усе необхідне для розробників Java: інкрементальний компілятор, редактор Java, відладчик, підтримку автоматичного рефакторинга і навігація за початковим кодом. Для роботи Eclipse SDK потрібна JRE(Java runtime environment) операційна система Windows, Linux, Mac OS X.

У проекті **Eclipse Tools** зібрані додатки, створені на базі Eclipse, такі як CDT (C/C++ Development Tooling, IDE для C і C++).

Проект **Eclipse Modeling Project** включає такі підпроекти як EMF (Eclipse Modeling Framework), середовище моделювання Eclipse - засіб для створення моделей і генерації коду для побудови інструментів і інших застосувань, що базуються на структурованій моделі даних, із специфікації моделі, прописаної в XMI. MDT (Model Development Tools) включає набір інструментів для створення моделей, визнаних стандартом в індустрії програмної інженерії. У тому числі плагін UML2, для роботи з моделями UML. Є, також проект GMT (Generative Modeling Technologies), який є «інкубатором», що займається розвитком ряду прототипів у напрямі Model Driven Engineering (MDE), розробки на основі моделей.

Усі додатки Eclipse використовують загальні концепції, інтерфейси і технічні рішення, що дозволяє говорити про їх інтеграцію.

NetBeans [11]

NetBeans IDE — вільне інтегроване середовище розробки (IDE) для мов програмування Java, JavaFX, C/C++, PHP, JavaScript, Python, Groovy. Середовище може бути встановлене і для підтримки окремих мов, і у повній конфігурації. Середовище розробки NetBeans за умовчанням підтримує розробку для платформ J2SE і J2EE.

Проект NetBeans IDE підтримувався і спонсорувався фірмою Sun Microsystems і після придбання Sun — Oracle, проте розробка NetBeans ведеться незалежно співтовариством розробників (NetBeans Community) і компанією NetBeans.Org.

За якістю і можливостям останні версії NetBeans IDE змагається з найкращим інтегрованими середовищами розробки для мови Java, підтримуючи рефакторинг, профілювання, виділення синтаксичних конструкцій кольором, автодоповнення мовних конструкцій на льоту, шаблони коду та інше.

NetBeans IDE доступна для платформ Microsoft Windows, GNU/Linux, FreeBSD, і Solaris (як SPARC, так x86). Для інших платформ доступна можливість зібрати NetBeans самостійно із сирцевих текстів.

Розробка середовища NetBeans почалася в 1996 під назвою Xelfi (гра букв на основі Delphi)[2][3], як проект студентів зі створення Java IDE під керівництвом факультету математики і фізики Карлова Університету в Празі. У 1997 році Роман Станек сформував компанію навколо проекту і став випускати комерційні версії середовища NetBeans до передачі всіх прав на IDE корпорації Sun Microsystems в 1999 році. Sun відкрила сирцеві коди середовища розробки NetBeans IDE в червні наступного року. Відтоді спільнота NetBeans постійно розвивається і росте завдяки людям і компаніям, що використовують і підтримує проект.

IntelliJ IDEA [12]

IntelliJ IDEA — комерційне IDE для Java від компанії JetBrains.

Перша версія IntelliJ IDEA з'явилася у січні 2001 року й швидко здобула популярність, як перша Java IDE із широким набором інтегрованих інструментів для рефакторингу, що дозволяла програмістам швидко реорганізовувати вихідні тексти програм. Дизайн середовища орієнтовано на продуктивність праці програмістів. З версії 9.0 є безплатний варіант Community Edition з відкритими кодами.

MS Visual Studio [2]

Microsoft Visual Studio — серія продуктів фірми Майкрософт, які включають інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів. Ці продукти дозволяють розробляти як консольні програми, так і програми з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-застосунки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight [14].

Visual Studio 97 — була першою спробою Microsoft створити єдине середовище для розробки на різних мовах програмування: Visual C++, Visual J++, Visual InterDev, і MSDN використовували одне середовище, зване Developer Studio. Visual Basic і Visual FoxPro використовували окремі середовища для розробки [14].

Visual Studio включає один або декілька з наступних компонентів [14]:

- Visual Basic .NET, а до його появи — Visual Basic
- Visual C++
- Visual C#
- Visual J#
- Visual F# (входить до складу Visual Studio 2010);
- Visual Studio Debugger

Багато варіантів постачання також включають:

- Microsoft SQL Server або Microsoft SQL Server Express

Огляд Visual Studio 2008

Вікна Visual Studio можна розділити на дві основні групи. Вікна *документів*, які зазвичай розміщуються в центрі ICP і містять редактори, дизайнери, web сторінки. *Інструментальні* вікна, які надають програмісту допоміжні функції. Інструментальні вікна включають Solution Explorer, the Class View та вікно Properties. Інструментальні вікна відрізняються від редакторів та дизайнерів тим, що прикріплюються до сторін ICP [15].

Програмні бібліотеки

Обчислювальні платформи

На даний момент найактивніше розвиваються дві конкуруючі лінії технологій створення ПО на основі компонентів обчислювальної платформи – технології Java і .NET. Розглянемо декілька елементів цих технологій, що є ключовими в створенні широко затребуваного нині і досить складного розподіленого програмного забезпечення (Web-застосування).

Java [12]

Технології Java компанії Sun Microsystems (нині належить Oracle) є набором стандартів, інструментів і бібліотек, призначених для розробки застосувань різних типів і пов'язаних один з одним використанням мови програмування Java. Торгова марка Java належить компанії Sun Microsystems, і ця компанія багато в чому визначає розвиток технологій Java, але в нім активно беруть участь і інші гравці — IBM, Intel, Oracle, Hewlett-Packard, SAP, Bea і ін.

Розробка Java почалась в 1990 році, перша версія Java була вперше випущена компанією Sun Microsystems в 1995 році.

Java включає декілька основних сімейств технологій:

• платформа Java Platform Standard Edition (Java SE або раніше J2SE). Призначена для розробки звичайних, в основному розрахованих на одного користувача застосувань або застосувань в масштабах малого підприємства;

• платформа Java Platform Enterprise Edition (Java SE або раніше J2EE). Призначена для розробки розподілених Web-застосувань рівня середніх і крупних підприємств;

• платформа Java Platform Micro Edition (Java ME або раніше J2ME). Призначена для розробки вбудованих застосувань, працюючих на обмежених ресурсах, в основному в мобільних телефонах і комп'ютеризованих побутових пристроях;

• платформа JavaFX. Наступний крок в розвитку Java в якості Rich Client Platform. Призначена для створення графічних інтерфейсів Rich Internet Applications (RIAs) корпоративних застосувань і бізнесу, які можуть запускатись як на персональних комп’ютерах, так і на мобільних пристроях.

• платформа Java Card. Призначена для розробки ПЗ, що управляє функціонуванням цифрових карт. Ресурси, наявні у розпорядженні такого ПО, обмежені найбільшою мірою.

Ідеї, закладені в концепцію і різні реалізації середовища віртуальної машини Java, надихнули безліч ентузіастів на розширення переліку мов, які могли б бути використані для створення програм, що виконуються на віртуальній машині. Ці ідеї знайшли також вираження в специфікації загальномовної інфраструктури CLI, закладеної в основу платформи .NET компанією Microsoft.

.NET [12, 13]

Вважається, що платформа .NET Framework стала відповіддю компанії Microsoft на велику популярність, що набрала на той час, платформа Java. .NET є схожим набором стандартів, інструментів і бібліотек, але розробка застосувань у рамках .NET можлива з використанням різних мов програмування. Деякою відмінністю від Java також є те, що код (на проміжній мові) в .NET не інтерпретується, а завжди виконується в режимі динамічної компіляції (JIT).

Розробка платформи почалася в другій половині 1990-х. Офіційно про розробку нової технології було оголошено 13 січня 2000 року, в день коли Біл Гейтс офіційно оголосив про передачу поста глави Microsoft Стіву Баллмеру. Цього дня керівництвом корпорації була озвучена нова стратегія компанії, що дістала назву Next Generation Windows Services (скор. NGWS, Нове покоління служб Windows). Нова стратегія повинна була об'єднати в єдиний набір існуючі і майбутні розробки Microsoft для надання можливості користувачам працювати зі Всесвітньою павутиною з безпровідних пристроїв, що мають доступ в Інтернет, як із стаціонарних комп'ютерів.

.NET Framework не містить особливих вказівок стосовно типів застосувань, які можуть створюватися за допомогою цієї платформи. Пояснюється це тим, що ніяких обмежень подібного роду просто не існує: .NET

Framework дозволяє створювати застосування для Windows, Web-застосування, застосування типу Web-служб і практично усі інші типи застосувань, які тільки можна собі уявити [13].

Окрім цього ще існує і версія Microsoft **.NET Compact Framework**, яка, по суті, є усіченим варіантом повної версії .NET Framework і може застосовуватися на пристроях типу персональних цифрових помічників (personal digital assistant - PDA) і навіть в деяких смартфонах [13].

Доступна також **.NET Micro Framework** – це реалізація платформи Microsoft .NET для вбудовуваного застосування в 32- і 64-розрядних мікроконтроллерах. Нині реалізована на мікроконтроллерах з архітектурою ARM7, ARM9 і Blackfin. Не вимагає наявності ОС.

Одним з головних стимулів до застосування платформи .NET Framework являється можливість використати її в якості засобу інтеграції різних операційних систем. Компанія Microsoft ініціювала прийняття стандартів, що описують її окремі елементи (на жаль, поки не всі), і вона ж є основним постачальником реалізацій цієї платформи і інструментів розробки. Завдяки наявності стандартів можлива незалежна реалізація .NET (наприклад, така реалізація розроблена у рамках проекту Mono), але, в силу молодості платформи (спочатку) і побоювань з приводу монопольного впливу Microsoft на її подальший розвиток, реалізації .NET не від Microsoft використовуються досить рідко.

Основою .NET являються віртуальна машина для проміжної мови (Intermediate Language — IL, іноді зустрічається скорочення Microsoft IL – MSIL, пізніше прийняли назву Common Intermediate Language – CIL), в яку транслюються усі .NET-програми, що також називається загальним середовищем виконання (Common Language Runtime — CLR), і загальна бібліотека класів (.NET Framework class library, FCL), доступна з усіх .NET-застосувань.

Проміжна мова є повноцінною мовою програмування, але вона не призначена для використання людьми. Розробка у рамках .NET ведеться на одній з мов, для яких є транслятор, в проміжну мову — Visual Basic.NET, C++, C# і ін. Проте різні мови досить сильно відрізняються одна від одної, і щоб гарантувати можливість з однієї мови працювати з компонентами, написаними на іншій мові, необхідно при розробці цих компонентів дотримуватися загальних правил (Common Language Specifications — CLS), які визначають, якими конструкціями можна користуватися в усіх .NET-мовах без втрати можливості взаємодії між результатами. Найбільш близький до проміжної мови C# — ця мова була спеціально розроблена разом з платформою .NET.

Джерела

1. Современная система разработки ПО. Основные компоненты, их функции и использование. //www.software.unn.ac.ru.

2. Пауэрс, Л. Microsoft Visual Studio 2008 / Л. Пауэрс, М. Снелл: Пер. с англ. — СПб.: БХВ-Петербург, 2009. – 1200 с.

3. Программная инженерия. Инструменты и методы программной инженерии (Software Engineering Tools and Methods). – Пер. SWEBOK, 2004 Сергей Орлик.

4. http://uk.wikipedia.org/wiki/Notepad++.

5. http://uk.wikipedia.org/wiki/Microsoft_Visual_Studio_Debugger.

6. Троелсен Эндрю. Язык программирования C# 2005 и платформа .NET 2.0.

7. http://en.wikipedia.org/wiki/Integrated_development_environment (http://ru.wikipedia.org/wiki/Интегрированная_среда_разработки).

8. http://ru.wikipedia.org/wiki/Eclipse_(среда_разработки).

9. Проект Eclipse // http://www.rsdn.ru/.

10. http://uk.wikipedia.org/wiki/NetBeans.

11. http://uk.wikipedia.org/wiki/IntelliJ_IDEA.

12. Гагарина Л. Г., Кокорева Е. В., Виснадул Б. Д. Технология разработки программного обеспечения: учебное пособие / под ред. Л. Г Гагариной. — М: ИД «ФОРУМ»: ИНФРА-М, 2008. — 400 с: ил. — (Высшее образование).

13. Уотсон, Карл и, Нейгел, Кристиан, Педерсен, Якоб Хаммер, Рид, Джон Д., Скиннер, Морган, Уайт, Эрик. Visual C# 2008: базовый курс. : Пер. с англ. - М. : ООО "И.Д. Вильяме", 2009. - 1216 с.: ил. — Парал. тит. англ.

14. http://uk.wikipedia.org/wiki/Microsoft_Visual_Studio

15. Craig Skibo, Marc Young, Brian Johnson. Working with Microsoft Visual Studio 2005. – Microsoft Press, 2006. – 280 с.