**Лекція № 1. Вступ до дисципліни. Поняття IDE**

Процес створення програми, незалежно від обраної мови програмування, складається, як мінімум, із двох етапів: написання власне тексту програми і її компіляція (трансляція). Текст (лістинг) програми може бути написаний практично в будь-якому текстовому редакторі і збережений у файлі, розширення якого залежить від використаної мови програмування. Так, для C++ необхідно створювати файли із розширенням .cpp, для Pascal – .pas, а для Java – .java. При цьому слід пам’ятати, що такий текст програми не може бути безпосередньо виконаний комп’ютером, оскільки процесор «вміє» виконувати лише програми, написані у машинних кодах. Безпосереднє ж створення програми за допомогою машинних кодів, хоч теоретично і можливе, але занадто трудомістке і незручне, через що, власне, і було розроблено численні мови програмування високого рівня. Для перетворення програми, написаної такою мовою, у вигляд, доступний для виконання комп’ютером, можливі два підходи. Перший з них – це ***компіляція.***

***Компіляція*** – це процес перетворення високорівневого початкового тексту програми в еквівалентний текст програми, але вже низькорівневою мовою. Компілятор – програма, яка виконує таке перетворення. Для різних мов програмування існують свої компілятори, що виконують дану роботу; наприклад, для C++ створені такі компілятори, як g++ (поставляється із більшістю \*nix-дистрибутивів), MinGW (версія g++ для Windows), Borland C++ Compiler, Microsoft Visual C++ Compiler, і т.д. Результатом виконання компіляції є виконуваний файл, що містить набір машинних кодів, виконуваних процесором. Такий підхід дозволяє максимально прискорити процес виконання програми при її запуску користувачем, адже машинні коди не потребують додаткового перетворення. Однак мінусом подібного підходу є прив’язка виконуваного файлу до конкретного набору команд процесора, що залежить від його архітектури. Таким чином, програма, скомпільована під певну архітектуру, працюватиме винятково на процесорах цієї архітектури, а це призводить до проблем із переносимістю програм, адже під кожний тип процесора (і кожну ОС) потрібно заново виконувати процес компіляції вихідного тексту програми.

Другий підхід – здійснення ***інтерпретації,*** тобто перетворення тексту програми у машинні коди безпосередньо перед її виконанням, «на льоту». В такому випадку для виконання програми на комп’ютері повинна бути встановлена т.зв. *віртуальна машина*, яка виконує подібне перетворення. Під кожну апаратну платформу достатньо створити свою віртуальну машину, яка буде працювати із одним і тим же стандартом вхідної мови, генеруючи машинний код, відразу орієнтований на обрану платформу. Зазвичай, щоправда, безпосередній текст програми перетворюється у деяку проміжну форму – байт-код, що може бути виконаний віртуальною машиною, а він уже, в свою чергу, і інтерпретується в машинні коди при запуску програми. Перевагою інтерпретації є легка переносимість програм, що важливо у випадку, якщо програма має запускатися на великій кількості різноманітних апаратних платформ, що актуально, наприклад, для вбудованих систем. Недоліком же такого підходу є, очевидно, сповільнення виконання програм через необхідність виконання трансляції в машинні коди перед кожним запуском. Інтерпретацію використовують такі програмні платформи, як, зокрема, Java та .NET. Також інтерпретація виправдана у web-технологіях; у цьому випадку роль інтерпретатора виконує браузер, а програма передається у вигляді власне лістингу, а не байт-коду. Зокрема, саме так працюють програмні вставки, написані на JavaScript.

Незважаючи на підхід до виконання програми, цей процес зазвичай є досить громіздким з точки зору програміста. Розглянемо для прикладу процес компіляції програми, що була попередньо написана у текстовому редакторі, за допомогою g++. Для цього потрібно в терміналі написати наступну команду:

g++ *filename* –o *execname*

де *filename –* ім’я файлу з лістингом, *execname –* ім’я скомпільованого файлу.

Якщо у процесі компіляції станеться помилка (наприклад, лістинг містить помилкові фрагменти коду), то програміст повинен відкрити файл лістингу в текстовому редакторі, внести необхідні правки, і повторити процес компіляції заново. Очевидно, що цей процес доволі незручний, тому для його автоматизації було розроблено спеціальний клас програмних продуктів – середовища розробки програмного забезпечення.

***Середовище розробки програмного забезпечення*** – сукупність програмних засобів, використовувана програмістами для розробки програмного забезпечення. Просте середовище розробки включає в себе *редактор тексту*, *компілятор* та/або *інтерпретатор*, *засоби автоматизації збірки* і *відладчик*. Коли ці компоненти зібрані в єдиний програмний комплекс, говорять про ***інтегроване середовище розробки*** (Integrated development environment – ***IDE***). Таке середовище представлене ​​однією програмою, не виходячи з якої, можна виробляти весь цикл розробки. До складу комплексу, крім перерахованих вище компонент, можуть входити *засоби управління проектами*, *система керування версіями*, різноманітні інструменти для спрощення розробки інтерфейсу користувача (GUI), стандартні заготовки («майстри»), що спрощують розробку стандартних завдань, засоби автоматизованого тестування програмних продуктів, та ін. Сучасні середовища розробки, що підтримують об'єктно-орієнтовану розробку ПЗ, також включають *браузер класів*, *інспектор об'єктів* і *діаграму ієрархії класів*. Якщо IDE включає в себе можливість візуального редагування інтерфейсу програми, вона називається *середовищем візуальної розробки*.

Найчастіше IDE призначені для декількох мов програмування, як, наприклад, Eclipse, NetBeans, Code::Blocks, Qt Creator, Geany, Embarcadero RAD Studio, Xcode, Microsoft Visual Studio, хоча існують середовища, що спеціалізуються на використанні лише однієї мови програмування, наприклад, Dev C++, Borland C++ Builder, Borland Delphi, Lazarus, Visual Basic.

Серед IDE є як платні продукти (Borland Delphi, Microsoft Visual Studio, IntelliJ IDEA), так і системи, що поширюються вільно (Eclipse, Code::Blocks, Geany, Dev C++, та ін.).

***Історія появи і розвитку IDE***

Історія розвитку операційних середовищ безпосередньо пов'язана з історією розвитку обчислювальної техніки та базового програмного забезпечення - операційних систем. Вся історія обчислювальної налічує трохи більше 60 років.

Кардинальні зміни у сфері технологій були пов'язані з використанням алфавітно-цифрових, а потім і графічних дисплеїв, розробкою операційної IBM OS / 360, винаходом мікропроцесорів, появі персональних ЕОМ, поширенням Інтернету.

Змінювався спосіб спілкування програміста з ЕОМ. Перші комп'ютери взагалі не мали базового програмного забезпечення. Програма могла вводитися вручну з пульта, а згодом за допомогою перфострічок і перфокарт. Перші термінали з електронно-променевими трубками (дисплеями) були призначені не для програмістів, а для операторів ЕОМ. Програмісти працювали з ЕОМ опосередковано, здаючи на обчислювальний центр колоди перфокарт. Їх основною мовою опису технологічних процесів була мова управління завданнями (в ОС EC - це JCL, Job Control Language).

Оператор ЕОМ – це була окрема спеціальність, яка в даний час практично не зустрічається. Оператор виконував функції управління багатозадачною операційною системою, міг впливати на хід виконання завдань, їх запуск і завершення, відповідати на запити, підготовляти пристрої введення-виведення. Поняття операційного середовища виникло саме в той час. Під операційним середовищем розумілася сукупність комп'ютерних програм, що забезпечує оператору можливість управляти обчислювальними процесами і файлами.

Початок 70-х років – час масового переходу на мови програмування, поява операційних систем першого покоління. У міру зростання потужності ЕОМ став можливим колективний доступ до її ресурсів, і дисплей став інструментом програміста.

Незважаючи на розвиток способів спілкування з ЕОМ, для програміста незмінним залишався основний цикл розробки й налагодження програм: редагування програми – компіляція – збірка – виконання програми – аналіз помилок – редагування. Цей цикл завжди треба мати на увазі при оцінці ефективності операційного середовища.

***Проблема вибору підходящого середовища***

Зазвичай вибирають ту модель і ті мови програмування, які добре знають члени колективу розробників. Обирати нову технологію, яку належить освоювати в процесі розробки – ризик провалити проект.

Будь-яке середовище дозволяє робити настроювання та адаптацію під ті чи інші вимоги: зміна інтерфейсу, режимів роботи, призначення гарячих клавіш, установка додаткових коштів («плагінів») і т.п. В арсеналі кожного досвідченого програміста є свої прийоми розробки, власні допоміжні засоби. Він має власні смаки та уподобання. Використовуючи настройки, програміст може зробити роботу в середовищі більш зручною для себе і, тим самим, більш ефективною. Він ніби проектує свою модель розробки на модель середовища. Це особливо важливо, коли серед інструментів є програми з відмінним інтерфейсом (наприклад, різне призначення гарячих клавіш).

Оцінюючи якість і ефективність операційного середовища, необхідно звертати увагу на інтерфейси інструментальних засобів. Сучасний підхід до інтерфейсу був закладений в стандарті фірми IBM. Цей стандарт втілився в багатьох програмних засобах, як інструментальних, так і прикладних. Стандарт визначає віконний інтерфейс, правила розміщення у вікнах елементів управління, призначення гарячих клавіш і навіть рекомендує колірні схеми інтерфейсу. Звичайно, в рамках цього стандарту можна реалізувати велику кількість різноманітних інтерфейсів. Розвиток і нововведення в інтерфейсах можна простежити на лінійках продуктів типу MS Office. Одним з важливих і необхідних властивостей середовища є можливість настройки. До інтерфейсу також відноситься зауваження про надмірності і необхідності мінімізації можливостей.

Технологія програмування багато в чому визначається мовою програмування, якою пишуться програми. У мові можуть бути закладені засоби, що впливають на технологічність і архітектуру розроблюваної системи (наприклад, об'єктно-орієнтованість, модульність і т.п.).

***Деякі популярні IDE***

***Code::Blocks***

IDE Code::Blocks – це вільне кросплатформенне середовище розробки програмного забезпечення. Маючи відкриту архітектуру, воно може маштабуватись за рахунок додаткових модулів. Дане IDE підтримує мови програмування С, С++, Fortran. Існують версії Code::Blocks для Windows, Linux и Mac OS X, що дозволяє працювати в цьому середовищі з використанням будь-якої з цих ОС.

Основні особливості Code::Blocks:

* Безкоштовність (поширюється за ліцензією GPLv3);
* Підтримка декількох компіляторів:
  + MinGW / GCC C/C++;
  + MS Visual C ++;
  + Digital Mars C/C++;
  + Borland C ++ 5.5, та ін.
* Дуже швидка система компіляції;
* Можливість імпорту проектів із MS Visual Studio та Dev-C ++$

Можливості інтерфейсу Code::Blocks:

* Підсвічування синтаксису;
* Згортання блоків коду;
* Автодоповнення коду;
* Браузер класів;
* Підтримка плагінів Devpack (від Dev-C++).

***Dev-C++***

Dev-C++ – інтегроване середовище розробки на C і C++, повнофункціональна C++ IDE.

Можливості Dev C ++:

* Зручний редактор з підсвічуванням синтаксису, нумерацією рядків, автовідступ і т.д.
* Можливість автозавершення коду для зручності роботи і підвищення продуктивності.
* Заготовки коду та шаблони для вставки.
* Використання закладок в редакторі для швидкого переміщення по коду.
* Експорт вихідних файлів або цілого проекту в HTML або RTF для публікації вихідних кодів на своєму веб сайті.
* Вбудований менеджер проектів.
* Імпорт проектів з MS Visual C ++.
* Можливість налаштування асоціації файлів з розширення - c, cpp, h і т.д.
* У навігаторі класів два варіанти огляду – перегляд функцій, класів та їх членів як для всього проекту, так і для поточного редагованого файлу.
* Гнучке налаштування робочого середовища, редактора і компілятора, велику кількість різних опцій.
* Використовується Mingw GCC компілятор, може працювати з будь-яким компілятором GCC.
* Можливість налагодження проекту - вбудований дебагер GDB.
* Існує портабельна версія програми, яка не потребує установки.
* Багатомовний користувальницький інтерфейс з підтримкою російської та української мов.

***Eclipse***

Eclipse є безкоштовною програмною платформою з відкритим вихідним кодом, що контролюється організацією Eclipse Foundation. Написана на мові програмування Java і основною метою її створення є підвищення продуктивності процесу розробки програмного забезпечення.

Претендує на статус найбільш популярної Java IDE і є єдиним конкурентом такої потужної платформи як NetBeans. Але, на відміну від NetBeans, який для створення елементів інтерфейсу користувача використовує платформо-незалежну бібліотеку Swing, в Eclipse використовується платформо-залежна бібліотека SWT – Standard Widget Toolkit.

IDE, розроблені на базі платформи Eclipse, застосовуються для створення програмного забезпечення на різних мовах програмування, тому що Eclipse є платформою для розробки будь-яких інтегрованих середовищ програмування і розширень для себе ж, за принципом "Додатки для Eclipse розробляються в самій Eclipse".

Особливості платформи Eclipse:

* Кросплатформенність – працює під операційними системами Windows, Linux, Solaris і Mac OS X.
* Використовуючи Eclipse, можна програмувати на безлічі мов, таких як Java, C і C ++, PHP, Perl, Python, Cobol та інших.
* Є фреймворком для розробки інших інструментів і пропонує великий набір API для створення модулів.
* Використовуючи підхід RCP (Rich Client Platform), Eclipse є інструментом для створення практично будь-якого клієнтського програмного забезпечення.
* Гнучкість і розширюваність досягається завдяки модульності платформи.

*Архітектура платформи Eclipse*

Основним елементом є виконуюче середовище – Eclipse Runtime, в якій виконуються коди розширень і модулів. Вона забезпечує всю базову функціональність платформи – управління розширеннями і оновленнями, взаємодія з операційною системою, забезпечення роботи системи допомоги.

Наступним елементом є власне IDE – вона відповідає за управління основними елементами програми, їх розташуванням і настройками, управління проектами, налагодження і складання проектів, пошук по файлах і командну розробку.

У стандартну поставку Eclipse SDK включені два плагіни – Java Development Tools або JDT, і Plug-in Developer Environment або PDE, таким чином ми отримуємо повністю готову IDE для Java-програмування і для розробки розширень для Eclipse.

Приклади інших спеціалізованих збірок Eclipse:

* Eclipse IDE for Java Developers - середовище розробки мовою Java.
* Eclipse IDE for C/C++ Developers (CDT) – функціональна IDE для програмування на C і C++.
* Eclipse IDE for JavaScript Web Developers – IDE для розробки веб-додатків з використанням HTML, XML, JavaScript і CSS.