**І. ЗНАЙОМСТВО З МОВОЮ *JAVA***

Що таке *Java*? Створюючи програми на більшості мов програмування, треба визначити, в якій операційній системі і на якому процесорі вони працюватимуть. Тільки визначивши це, можна долучити до програми виклик функцій з бібліотеки, призначеної для відповідної операційної системи. Наприклад: якщо розробляють програму для *Windows*, то можна використати бібліотеку *Microsoft Foundation Classes*; для роботи на платформі *Macintosh* − функції з *Mac OS Toolbox*. Після компіляції вихідних текстів отримуємо код, готовий до виконання на певному процесорі. Система *Windows* переважно працює на базі процесорів фірми *Intel*, комп’ютери *Macintosh* використовують процесори *Motorola 680x0* або *PowerPC*. Створюючи програми на *Java*, можна не замислюватися над тим, в якій операційній системі вони працюватимуть. *Java* має власний набір машинно-незалежних бібліотек, які називають пакетами. Щодо процесорів ситуація аналогічна. Компілятор *Java* не генерує безпосередньо інструкції процесору. Він створює проміжний код − байткод для віртуальної машини *Java* (*Java Virtual Machine* − *JVM*). Виходячи з того, що ядро віртуальної машини *Java* реалізовано практично для всіх типів комп’ютерів, вважатимемо файли байткодів незалежними від платформи.

**1.1. Створення та еволюція**

Започатковано мову *Java* у проекті фірми *Sun Microsystems* під назвою *Green* (1991). Головними розробниками першої робочої версії були Джеймс Гослінг (James Gosling), Патрік Ноутон (Patric Naughton), Кріс Ворс (Chris Warth), Ед Френк (Ed Frank) і Майкл Шерідан (Mike Sheridan). До 1995 року мову називали *Оак*, однак не пройшовши перевірку на допустимість торгової марки, було перейменовано на *Java*. Ідеєю створення нової мови програмування були не потреби *Internet*, а необхідність створення програмного забезпечення, яке не залежить від платформи (тобто архітектури) і використовується в побутових електронних приладах. Під час відпрацювання деталей *Java* виник вагомий чинник, який відіграв важливу роль щодо цієї мови. Таким чинником була всесвітня інформаційна служба *World Wide Web* (*WWW*), розквіт якої припадає на 1994−1995 р.

Ідеї створення ефективних і незалежних (працюючих на різноманітних процесорах під керівництвом різних операційних систем) програм такі ж давні, як і саме програмування, і завжди витіснялися нагальнішими проблемами. Якщо зважити на те, що більшість програмістів належить до одного з трьох кланів (*Intel*, *Macintosh*, *UNIX*), які безупинно змагаються між собою, то зрозумілим стає те, що нагальної потреби в переміщенні коду довший час не виникало. Проте з виникненням *Internet* і *WWW* проблема переміщення програм стала втричі гострішою. Різко змінилися акценти: від створення коду для вбудованих контролерів побутової техніки до програмування для *Internet*. Це і спричинило великий успіх мови *Java*.

**1.2. Від *С/С++* до *Java***

Після виникнення в 1979 р. мови *С++*, до якої було додано порівняно з мовою *С* об’єктно-орієнтовані засоби і збережено усі сильні сторони *С,* складається враження, що програмісти знайшли нарешті досконалу мову. Проте навіть могутня і популярна мова програмування має свої недоліки. Це важкі щодо розуміння і використання аспекти *С++*, пов’язані з керуванням пам’яттю і покажчиками. У *С++* також легко зберігається процедурний стиль програмування. Згідно з думкою П.Нортона, наступним кроком у логічному розвитку мов програмування є *Java*. Ця мова опирається на найпопулярнішу мову *С++* (розробники *Java* зробили це свідомо), проте цілковито відкидає поняття процедурного програмування і змушує підкорятися принципам об’єктноорієнтованого програмування (ООП). У мові *Java* відсутні покажчики, а керування пам’яттю відбувається автоматично.

Зважаючи на подібність мов *Java* та *С++*, дехто вважає, що *Java* − це "*Internet*-версія *С++*". Насправді це не так. Незважаючи на значний вплив *С++* на створення *Java*, щодо них не існує ніякої сумісності (ні зверху вниз, ні знизу вверх). Крім того, цю мову не розроблено з метою заміни *С++*. Мову *С++* застосовують для вирішення одних проблем, мову *Java* − для інших. Найважливіша з них − створення програм, які не залежать від платформ виконання.

**1.3. Девізи *Java***

Розробники *Java* створювали мову з метою втілення таких базових принципів:

* простота;
* безпека;
* перенесення;
* об’єктно-орієнтована направленість;
* стійкість щодо помилок;
* багатопоточність;
* незалежність від архітектури;
* інтерпретація;
* висока продуктивність;
* розподіленість; ◊ динамічність.

***Простота, інтерпретація, перенесення, незалежність від архітектури.*** До простих мов програмування належать ті, які працюють з інтерпретатором (наприклад, *Бейсік*). Перші персональні комп’ютери поставлялися з інтерпретатором *Бейсіка*. Сьогодні їхнє місце займають *HTML* і мови сценаріїв для *Web*. Вивчати і використовувати мови програмування, які компілюються, набагато складніше, ніж ті, які інтерпретуються. Тобто є мови програмування для професіоналів. Наприклад, *С++*, де використання покажчиків і керування пам’яттю є складними не тільки для початківців, але й для досвідчених програмістів. Одна стрічка програми, яка звертається до недозволеного місця в пам’яті, може спричинити до збоїв не тільки програми, але й комп’ютера загалом.

*Java* − це мова, програми якою компілюються та інтерпретуються, і водночас вона має просту структуру мови високого рівня. Написана програма компілюється в проміжну форму − *байткод*. Пізніше ця програма виконується, тобто інтерпретується виконавчим середовищем *Java*. Байткод дуже відрізняється від машинного коду, який є послідовністю нулів та одиничок. Байткод − це набір інструкцій, які подібні до команд *Асемблера*. Машинний код комп’ютер виконує безпосередньо, а байткод потрібно інтерпретувати. Тому машинний код можна використати тільки на конкретній платформі, для якої його скомпільовано. Байткод можна виконувати на довільній платформі, на якій є виконавче середовище *Java*. Саме ця можливість і робить програми на *Java* незалежними від архітектури. Так як байткоди є проміжною формою програми, то його інтерпретація вимагає незначних витрат.

Байткод створено для машини, яка реально не існує. Цю машину називають *віртуальною Java-машиною* (*JVM*), вона існує тільки в пам’яті комп’ютера. Створення компілятором *Java* байткоду для неіснуючої машини − це тільки половина процесу, який забезпечує незалежність від архітектури. Другу частину виконує інтерпретатор *Java*, який виконує роль посередника між віртуальною *Java*-машиною та реальним комп’ютером.

***Архітектура мови для розподіленого мережевого середовища.*** Головною вимогою щодо мови для роботи в розподіленому просторі комп’ютерів (наприклад, в *Internet*) − це можливість працювати на різнорідних і розподілених платформах.

Мова *Java* є пристосованою до перенесення завдяки підтримці стандартів IEEE для структур даних, наприклад, цілих чисел, чисел з плаваючою комою і рядків.

До мови *Java* зачислено безпосередньо підтримку таких розповсюджених протоколів як *FTP*, *HTTP*, що забезпечує сумісність під час роботи в мережі.

*Java* забезпечує розподілену роботу за допомогою механізму виклику віддалених методів (*RMI*), тобто дає змогу використовувати об’єкти, розташовані на локальних і віддалених машинах.

***Багатопоточність.*** У багатопоточних операційних системах для кожного застосування (процесу) надається окрема захищена область пам’яті, в якій зберігаються коди програми і дані. А час одного процесора квантується між цими процесами. З метою запуску процесу або переключення з одного на інший на рівні операційної системи необхідно виконати значний об’єм роботи. Тому для розробників прикладних програм спеціально створили "полегшену" версію системного процесу − потік. Найбільшою проблемою, пов’язаною з процесами і потоками, є їхнє функціонування під керівництвом конкретної операційної системи. Спеціалісти компанії *Sun* зробили потоки частиною мови програмування. Тому багатопоточне застосування, написане мовою *Java*, працюватиме і в операційних середовищах *Windows*, *Unix*, *MacOs*.

***Висока продуктивність.*** Інтерпретатор *Java* може виконувати байткоди зі швидкістю, яка наближається до швидкості виконання коду, відкомпільованого до машинного формату, що досягається завдяки використовування інтерпретатором багатьох потоків виконання. Наприклад, доки комп’ютер чекає на введення даних, фонові потоки можуть зайнятися очищенням пам’яті.

***Стійкість до помилок.*** *Java* − це мова строгого використання типів, що зумовлює зменшення числа помилок під час написання програми.

У мові Java відбувається автоматична перевірка виконання граничних умов під час роботи з масивами і стрічками, які в *Java* є класами.

У *Java* немає арифметики покажчиків, а керування пам’яттю здійснюється автоматично. Програмний код, написаний мовою *Java* не може зіслатися на пам’ять поза простором програми або зробити помилку внаслідок вивільнення пам’яті і тим самим вичерпати всю пам’ять.

Зазначимо, що у *Java* організовано процес автоматичного збирання сміття, тобто об’єктів, на які більше ніхто не вказує.

***Безпека.*** Функції забезпечення безпеки дуже важливі для розподілених мереж з безліччю вірусів, "троянських коней" і т. п. Для реалізації цієї мети розробники мови *Java* створили механізм, який отримав назву пісочниці (*sandbox*):

* перевірку на рівні *JVM*;
* захист на рівні мови;
* інтерфейс *Java Sucurity* (цифрового підпису).

**1.4. Структура мови Java**

***Аплети і застосування.*** Якщо б мову *Java* використовували для створення машинно незалежних застосувань, то і цього було б достатньо для її успіху в програмістів. Однак у 1993 р. компанія *Sun* звернула увагу на зростання популярності *Internet* і почала доробляти мову *Java* так, щоб написані на ній програми можна було запускати з *Web*-бровзерів. Відтоді самостійні *Java*-програми називають *застосуваннями* (applications), а програми, які виконуються під керівництвом інших програм ( переважно *Web*бровзерів), − *аплетами* (applets). Аплети необхідні у випадку, коли для створення потрібної *Web*-сторінки не вистачає можливостей HTML, мови сценаріїв, а також у випадку, коли необхідно забезпечити зворотній зв’язок з клієнтом.

***Простір імен.*** Під час написання складних програм інколи важко забезпечити унікальність імен змінних і класів. У мові *Java* використовують систему декількох рівнів вкладеності імен:

1. - простір імен пакета;
2. - простір імен одиниці компіляції (файл класу);
3. - простір імен типу (клас у класі);
4. - простір імен методу;
5. - простір імен локального блоку;
6. - простір імен вкладеного локального блоку.

За підтримку і перетворення просторів імен відповідає компілятор *Java*. Імена, пов’язані з кожним рівнем, відокремлюють від імен інших рівнів крапкою. Наприклад, Java.awt.BorderLayout.

***Файли програми.*** Файл з вихідним текстом програми мовою *Java* є звичайним текстовим файлом з розширенням .java. Після компіляції (за допомогою javac) вихідних текстів отримується по одному файлу для кожного класу, оголошеного в тексті програми. Імена файлів збігаються з іменами класів ( з урахуванням регістра) і мають розширення .class.

***Пакети.*** У багатьох мовах програмування набір зв’язаних класів або функцій називають бібліотекою. *Java* надає поняттю бібліотеки певний відтінок, використовуючи для описання набору зв’язаних класів термін *пакет.* Наприклад, базові функції *Java* розташовані в пакеті java.lang.

***Оператори імпорту.*** З метою використання класів з існуючих пакетів необхідно до тексту програми (першими) додати оператори *import java.awt.\*;*

Це означає, що всі класи пакета java.awt можна використовувати при написанні програмного коду безпосередньо без посилання на пакет.

***Оголошення класів.*** Усі класи в мові *Java* є похідними від системного класу *Object*. У *Java* допускається тільки одинарне наслідування, тобто в ієрархії перед класом є тільки один базовий клас.

***Оголошення інтерфейсу.*** Інтерфейсом у мові *Java* називають абстрактний клас. Його введено для реалізації наслідування від декількох класів.

**1.5. Мова *Java* − це не вдосконалена HTML**

Серед початківців розповсюджена хибна думка, що оскільки мову *Java* використовують для створення *Web*-сторінок, її можна вважати деяким вдосконаленням HTML (Hypertext Markup Language − мова розмітки гіпертексту). Насправді це не так. HТML є засобом логічної організації інформації і створення гіпертекстових зв’язків з відповідними даними. Вона дає змогу читати документи не тільки зверху вниз, а й у будь-якому іншому порядку, проте ніколи не була мовою програмування. Єдиний зв’язок між HТML і *Java* − це наявний в HТML дескриптор APPLET, за допомогою якого викликається для виконання аплет *Java*.

**1.6. Середовище розробки програм мовою *Java***

Компанія *JavaSoft*, створена розробниками мови *Java*, пропонує безкоштовно набір засобів для програмістів мовою *Java*

*JDK* (*Java Development Kit*) за адресою http:// java.sun.com/products/jdk/. *JDK* містить все необхідне для створення програм: базові функції мови, інтерфейс прикладного програмування (АРІ) з наборами пакетів й основні інструменти. Більшість версій *JDK* містять сім інструментів розробки на *Java*:

1. компілятор (javac);
2. генератор документації (javadoc);
3. генератор файлів заголовків і заглушок мови *С++* для *Java* (javah);
4. інтерпретатор (java);
5. програму перегляду аплетів (appletviewer); 6) реасемблер файлів класів; 7) відлагоджувач програм (jdb).

Сьогодні версії *JDK* існують для більшості операційних систем. Починаючи з версії *JDK 1.2* прийнята нова домовленість щодо імен. Загальну технологію називають *Java 2*, а засоби розробника *Java 2* − *SDK* (*Software Development Kit*).

Процес встановлення *SDK* налічує три етапи:

1. отримання SDK (переважно з *Internet*); 2) встановлення SDK; 3) перевірка конфігурації.

При перевірці конфігурації необхідно визначити правильне встановлення змінних оточення PATH, яка вказує на каталог з інструментами SDK, та CLASSPATH, яка задає шлях до каталогів класів *Java* (як готових, так і власних).

**1.7. Комерційні інтегровані середовища розробки програм мовою *Java***

Професійні програмісти, яким оплачують за кожну хвилину роботи, повинні працювати дуже продуктивно. Для підвищення продуктивності їхньої праці використовують IDE (*Integrated Development Environment*) − візуальне середовище розробки. Кожна з таких відомих компаній як *Microsoft,* *Symantec*, *Borland*, *Sun* пропонує свої IDE. Перелік компаній, які мають засоби розробки *Java*, можна знайти на сервері Yahoo за адресою http:// www.yahoo.com/Business\_and\_Economy/Companies/Software/Progra mming\_Tools/Laguages/Java/. При наявності великої кількості IDE відразу ж виникає запитання, яке середовище розробки найкраще? У 1997 році, згідно з опитуванням, яке здійснило об’єднання developer.com, складено такий список IDE:

1. Visual Cafe for Java від Symantec;
2. JBuilder від Borland;
3. VisualAge for Java від IBM;
4. CodeWarrior Professional від Metrowerks;
5. Visual J++ від Microsoft.

Інша компанія *Market Decisions, Inc.*, яка виконувала незалежні маркетингові дослідження, повідомила, що майже половина розробників *Java* використовують *Visual J++* від *Microsoft*. Детальну інформацію щодо продуктів *Java* компанії

*Microsoft* можна знайти за адресою http:// www.microsoft.com/visualj/. Зауважимо також, що компанія *Sun Microsystems* подала судовий позов на *Microsoft* з приводу використання логотипу *Java* на тій підставі, що *Visual J++* не підтримує стандарт *Java*.

Конкурент компанії *Microsoft* на ринку продуктів з *Java* компанія *Symantec* своїм важливим козирем вважає те, що її Visual Cafe цілковито відповідає стандартові *Java*. Компанія *Symantec* визначає *Visual J++* як дещо підправлене та адаптоване середовище *C++* від *Microsoft*.

Фірма *Borland* також намагається не відставати від конкурентів і пропонує своє середовище JBuilder. Детально познайомитися з JBuilder можна на Web-сервері за адресою http:// www.borland.com/jbuilder/.

**1.8. Перша проста програма мовою *Java***

З метою створення програми на *Java* необхідно інсталювати JDK, використовуючи *Internet*-адресу:

http://java.sun.com/products/jdk/. Далі виконати такі дії:

1. написати текст програми;
2. відкомпілювати усі класи за допомогою компілятора javac;
3. виконати програму за допомогою інтерпретатора java.

***1.8.1. Написання тексту програми***

Якщо не використовувати інтегроване середовище розробки, то для написання тексту програм мовою *Java* можна використовувати звичайний текстовий редактор. Наприклад, в ОС *Windows* – це *NotePad, WordPad.*

Для більшості мов програмування ім’я файла, який містить вихідний текст програми, може бути довільним. Для мови *Java* це не так. У *Java* вихідний код офіційно називається модулем компіляції (compilation unit). Він є текстовим файлом, який містить одне або більше визначень класів. Компілятор вимагає, щоб вихідний файл мав розширення .java, а його ім’я збігалося з іменем класу (з урахуванням регістру), в якому є метод *main()*.

Компанія *Sun* пропонує дотримуватися декількох домовленостей з приводу імен під час написання *Java*-програм:

1. В іменах класів можна використовувати як великі, так і малі букви. Перша буква має бути великою. Наприклад, класам бажано надавати імена *NativeHello* і *HelloWorld*, а не *nativeHello* і *helloWorld*.
2. В іменах методів також можна використовувати символи обох регістрів, однак перша буква має бути малою.

Наприклад, методу можна присвоїти ім’я *sayHello()*, а не *SayHello()* або *sayhello()*.

1. Для найменування властивостей використовують ті ж домовленості, що і для методів. Наприклад, властивість можна назвати *thePoint*, а не *ThePoint* чи *thepoint*.
2. Імена констант переважно пишуть великими буквами. Наприклад, *PI*, а не *pi*.
3. Імена методів доступу до властивостей розпочинають з *set* і *get*.
4. Якщо властивість має тип *bоolean*, то краще в ролі префікса в методі писати *is*, *has*.

Під час написання програм небажано використовувати відкриті властивості класу прямо, а тільки через методи доступу.

***Аналіз коду програми.*** Розглянемо текст загальноприйнятої першої програми, яка виводить текстовий рядок на екран.

1. */\* Проста програма мовою Java \*/*
2. *public class Example {*
3. *public static void main(String args [ ]){*
4. *System.out.println("Перша програма мовою Java");* 5  *}*

6 *}*

Нумерацію рядків наведено для зручності пояснення тексту програми, а не через необхідність. Незважаючи на те, що програма дуже коротка, вона складається з декількох ключових особливостей, визначальних для всіх програм мовою *Java*. Вона розпочинається з коментарію: текст, обрамлений /\*...\*/, у першому рядку. Цей тип коментарю називають багаторядковим. У мові *Java* є однорядкові коментарі, які розпочинаються з //.

Звернемо увагу на те, що в мові *Java* усі змінні і методи (у тім числі *main*) не можуть бути за межами класу. Тому другий рядок є оголошенням класу *Example.*

Усі *Java*-програми (крім аплетів) розпочинають свою роботу з виклику методу *main()* (рядок 3). Метод *main ()* необхідно оголосити як *public*, тому що його викликають кодом, визначеним за межами класу. Ключове слово *static* дає змогу викликати метод *main()* без обов’язкового створення екземпляра класу (об’єкта типу *Example)*. Це вимушений крок, оскільки *main()* викликає інтерпретатор *java* до створення будь-яких об’єктів. Ключове слово *void* повідомляє, що метод не повертає ніякого значення.

Необхідно пам’ятати, що в програмах мовою *Java* розрізняється регістр букв. Наприклад, метод *Main* буде відкомпільовано, проте інтерпретатор *java* його не знайде.

Складні програми можуть мати десятки класів, однак тільки один може (і повинен) мати метод *main()*. Виняток становлять аплети – їхній запуск здійснює *Web*-броузер за допомогою інших засобів.

У методі *main()* один параметр *String args[ ]* (*String* з великої букви тому, що це клас). У цей масив записуються параметри командного рядка, якщо їх задають під час запуску програми. В програмі з елементів масиву *args[0], args[1]* тощо можна зчитувати і використовувати параметри, задані у командному рядку. Кількість переданих до програми параметрів можна визначити за допомогою методу *args.length()*.

Четвертий рядок програми містить виклик методу *System.out.println()*. Інженери компанії *Sun* здійснили чималу підготовчу роботу і написали значну кількість кодів, які можна використовувати в своїх програмах. Ці коди розміщено в пакетах, імена яких починаються з *java.*, *sun.* і *javax*. Клас *System* розташовано в пакеті *java.lаng*, який автоматично імпортується в усі програми. Об’єкт *out* має тип *PrintStream* і метод *println()*. *Out* – це вихідний потік, який під’єднується до консолі. Оскільки сучасні операційні системи мають графічний віконний інтерфейс, консоль введення-виведення використовують, переважно, для простих демонстраційних програм. Оператор *System.out.println()* завершується крапкою з комою. Запам’ятаємо, що всі оператори в *Java* завершуються " ; ".

Усі блоки програм мовою *Java* обрамлюють фігурними дужками "{}". У наведеній програмі є тіло класу і тіло методу, тобто два вкладені блоки.

***1.8.2. Компіляція програми***

Після написання тексту програми його бажано розмістити в своєму робочому каталозі. Для компілятора це не має значення, проте зручно зберігати вихідний код мовою *Java* і файли класів в одному каталозі. Відкриваємо вікно командної оболонки. Переконайтесь, що поточним є ваш робочий каталог. Після цього необхідно задати команду:

*javaс Example.java*

Ім’я класу необхідно задавати з урахуванням регістра, незважаючи на те, що, наприклад, у *Windows* імена файлів не залежать від регістра.

***1.8.3. Запуск програми***

Компілятор створює файл *Example.class*, який містить байткод програми, тобто інструкції з виконання інтерпретатором java. У командній стрічці необхідно задати (поточним має бути робочий каталог з файлом класу) *java Example.class*

Розширення *.class* можна не задавати. Якщо файл буде знайдено, то інтерпретатор виконає код, який є в заданому класі, тобто виведе в командний рядок: "Перша програма мовою Java".

***1.8.4. Проблеми***

Під час написання і запуску *Java*-програми часто виникають такі проблеми:

1. Командна оболонка не може знайти файл компілятора javac. Необхідно перевірити, чи змінна PATH має значення підкаталога BIN каталога SDK.
2. Змінна CLASSPATH не вказує каталогів з класами.

**1.9. Перший аплет**

Мовою *Java* можна писати ще один тип програм: аплети (applets). Це невеликі програми, які розташовані на сервері, поєднаному з *Internet*. Аплети передаються мережею, автоматично встановлюються і запускаються як частина *Web*-документа.

Аплети принципово відрізняються від застосувань за структурою і деякими ключовими областями. В застосуванні може використовуватися або інтерфейс командного рядка (як в описаній вище першій програмі *Example*), або графічний інтерфейс користувача (GUI). В аплетах можна використовувати тільки GUI.

Створення програм з використанням GUI є набагато складнішим, ніж тих, які використовують командний рядок.

*Java* є об’єктно-орієнтованою мовою. Інженери компанії *Sun* створили завершений аплет, який можна використовувати в ролі заготовки. Все, що потрібно для написання програми-аплета, − це створити клас, який успадковує клас *Java.applet.Applet*, і перевизначити методи, які не задовольняють наших вимог. Нижче наведено код найпростішого аплета: *import java.applet.Applet; import java.awt.Graphics; public class HelloApplet extends Applet*

*{ public void paint(Graphics g)*

*{ g.drаwString ("Перший аплет", 20, 20); }*

*}*

Оператори *import* дають змогу імпортувати класи, і в тексті програми можна писати просто *Applet* і *Graphics* замість повного імені j*ava.applet.Applet* i *java.awt.Graphics*.

Написавши простий оператор

*HelloApplet extends Applet*, ми отримуємо готовий аплет, який успадковує всі змінні і методи базового класу *Applet*, створеного компанією *Sun*.

При створенні аплета для відображення інформації необхідно використовувати методи малювання, а не виведення в потік. Доступ до графічного простору, наданого броузером, здійснюється за допомогою об’єкта *Graphics*. Його необхідно передати в якості параметра методу *paint()*. Метод *paint()* викликається щоразу, коли вікно аплета повинно знову з’явитися на екрані.

Аплет компілюється так само, як і проста програма. Проте виконання аплета можна здійснити через:

1. *Java*-сумісний *Web*-бровзер (який підтримує тег APPLET);
2. Викликом *appletviewer* з SDK.

Для запуску аплета на виконання необхідно написати невеликий фрагмент коду HTML:

*< HTML>*

*<BODY>*

*<APPLET code ="HelloApplet.class" width="200" height="200"> </APPLET>*

*</BODY>*

*</HTML>*

Якщо цей код HTML розмістити у файлі RunApp.html, то аплет можна запустити командою *appletviewer RunApp.html*

з командного рядка, або шляхом відкривання цієї сторінки в *Java*сумісному *Web*-бровзері.