ОСНОВНІ ПОНЯТТЯ ПРОГРАМУВАННЯ

Програмування – розробка програм за допомогою мов програмування.

Мова програмування – це формальна система знаків, що призначена для написання програм, зрозуміла для виконавця (комп'ютера).

Програма (program) - впорядкована послідовність команд комп'ютера для розв'язання задачі.

Програмне забезпечення (software) - сукупність програм обробки даних та необхідних для їх експлуатації документів. Програми призначені для машинної реалізації завдань (Задач). Терміни завдання і застосування (програма) мають дуже широке вживання в контексті інформатики і програмного забезпечення.

Завдання або задача (task) - проблема, що підлягає вирішенню.

Застосування або програма (application) - програмна реалізація на комп'ютері рішення задачі.

Розробка програмного забезпечення (software development) - це рід діяльності (професія) і процес, спрямований на створення і підтримку працездатності, якості та надійності програмного забезпечення, використовуючи технології, методологію та практики з інформатики, управління проектами, математики, інженерії та інших сфер знань.

Алгоритм (Algorithmi, від імені перського математика IX ст. аль-Хорезмі) — система правил виконання обчислювального процесу, що обов'язково приводить до розв'язання певного класу задач після скінченного числа операцій. При написанні комп'ютерних програм алгоритм описує логічну послідовність операцій. Для візуального зображення алгоритмів часто використовують блок-схеми. Засоби швидкої розробки додатків (RAD) дозволяють створити програмний продукт, що складається з ряду застосувань, що дозволяють користувачам вводити дані в таблиці або редагувати вже існуючі дані, аналізувати введені дані і представляти їх у більш зручному для сприйняття вигляді - графіків, зведених таблиць або звітів (у тому числі у вигляді "паперових" документів).

ПАРАДИГМИ ТА МОВИ ПРОГРАМУВАННЯ

Усе програмування прийнято поділяти на два основних види:

- Декларативне.
- Імперативне.

Декларативне програмування - термін з двома різними значеннями. Згідно першому визначенню, програма «декларативна», якщо вона описує щось, а не як його створити. Наприклад, веб-сторінки на HTML декларативні, оскільки вони описують що повинна містити сторінка, а не як відображати сторінку на екрані. Цей підхід відрізняється від мов імперативного програмування, що вимагають від програміста вказувати алгоритм для виконання.

Згідно другому визначенню, програма «декларативна», якщо вона написана на виключно функціональній, логічній або константній мові програмування.

Імперативне програмування — парадигма програмування, згідно з якою описується процес отримання результатів як послідовність інструкцій зміни стану програми. Подібно до того, як з допомогою наказового способу в мовознавстві перелічується послідовність дій, що необхідно виконати, імперативні програми є послідовністю операцій комп'ютеру для виконання.

Парадигма програмування — це спосіб мислення розробника програми. Мова програмування може підтримувати або не підтримувати ту чи іншу парадигму. В першому випадку застосування парадигми стає зручним, тобто простим, безпечним і ефективним. Ми розглянемо три основних наказових парадигми — процедурне, об'єктне (модульне) і об'єктно-орієнтовне (ієрархічне) програмування.

ПРОЦЕДУРНЕ ПРОГРАМУВАННЯ

Процедурне програмування подає програму у вигляді набору алгоритмів, для оформлення яких можуть застосовуватися іменовані програмні блоки — процедури і функції. В останньому випадку передбачається наявність механізмів передачі параметрів і поверненні результату.

Спочатку процедурне програмування користувалося довільними засобами керування, в тому числі, переходом за міткою — одним з найбільш вживаних операторів керування в Фортрані.

1968 році голландський вчений Е. Дейкстра вперше звернув увагу на проблеми, що виникають у програмах з неконтрольованими переходами, в 1970 році проголосив новий напрямок, який він назвав структур(ова)ним програмуванням.

Структурне програмування — це варіант процедурного, що вживає три типи структур керування: послідовне виконання дій, розгалуження і цикл. Не дивно, що Фортран не підтримував цю парадигму — в наборі його засобів не було циклів за умовами. Починаючи з Алголу, а особливо Паскалі, цикли стають основним засобом організації обчислень в програмі.

Автор Паскалю, професор Н. Вірт, відібрав до створюваної ним мови програмування лише прості в поясненні і легкі в реалізації конструкції. Завдяки сильній типізації програми в Паскалі відзначаються високою надійністю, вони мобільні завдяки закладеній в них концепції Паскаль-машини, їх легко читати і розуміти завдяки дисципліні програмування, продиктованої вжитою парадигмою.

Але разом з цим застосування Паскалю гальмувалося саме складністю виходу за межі віртуальної машини, потребою ефективного використання наявної апаратури. Головним критерієм, вжитим Б.Кернiганом і Д.Річі до створеної ними мови С, стала саме гнучкість використання особливостей конкретної апаратури і ефективність виконання програм.

МОДУЛЬНЕ ПРОГРАМУВАННЯ

Процедурна парадигма віддала належне алгоритмічній компоненті програмування. Але з ростом обсягу програм і складності даних з'явилася нова проблема структурної організації даних, найбільш ємко висловлена Віртовською формулою "алгоритми + структури даних = програми".

Поняття модуля як абстракції даних було вперше запропоноване Парнасом у 1972 році, правда на той час уже існувала мова програмування Симула 67, в якій використовувалася парадигма об'єктів. У найбільш повному виді поняття абстракції даних було реалізоване в мові програмування Модула-2.

Головна ідея полягає в забезпеченні доступу до даних, не залежному від їх конкретного представлення. Самі дані і програми їх обробки вбудовуються (інкапсулюються) в окремій одиниці програми.

ОБ'ЄКТНО-ОРІЄНТОВНЕ ПРОГРАМУВАННЯ

Об'єктно-орієнтована парадигма наділила класи ієрархією. Об'єктно-орієнтоване програмування за метафорою Б.Страуструпа, автора С++ — однієї з найпопулярніших мов об'єктно-орієнтованого програмування, — це високоінтелектуальний синонім доброго програмування. Дійсно, нові парадигми програмування з'являються не так часто, не частіше однієї десятиліття. Той факт, що об'єктно-орієнтована парадигма успішно використовується протягом 20 років, сам по собі служить вагомим підтвердженням її життєздатності.

Алгоритми, реалізовані в процедурному програмуванні, надто конкретні. Будь-яка модифікація — це вже новий алгоритм і таким чином кількість процедур і функцій, що знаходяться у використанні, надмірно зростає. Модульне програмування групує алгоритми в модулі, одночасно інкапсулюючи структури даних. Тепер залишається зробити наступний крок — побудувати ієрархію модулів або класів.

Таких ієрархій може бути дві. Перша з них — бути частиною чогось. Наприклад, грань є частиною многогранника, ребро — частиною грані, вершина — частиною ребра. Інша ієрархія — бути узагальненням або конкретизацією. Наприклад, овал і многокутник служать конкретизацією плоскої фігури, коло — конкретизацією овалу, чотирикутник — конкретизацією многокутника, подальшими конкретизаціями чотирикутника можуть служити паралелограм, прямокутник, ромб, квадрат. Той факт, що квадрат, ромб, прямокутник є повноцінними паралелограмами дозволяє їм користуватися усіма програмними засобами, створеними для паралелограма, паралелограм в свою чергу є повноцінним чотирикутником і так далі. Цей принцип, відомий під назвою reusable — знову вживаний — став одним з найважливіших досягнень об'єктно-орієнтованої парадигми. Знову вживаючи вже існуюче програмне забезпечення в більш конкретизованих умовах, ми дописуємо лише ту його частину, яка стосується особливостей наявної конкретизації. Цей принцип дістав назву programming by difference або дописування програм.

І, нарешті, об'єктно-орієнтована парадигма доводить до логічної завершеності принцип моделювання реального світу, а точніше тієї його частини, абстракцією якої служить програма. При цьому підході програма складається з об'єктів, що відповідають реальним поняттям або предметам. Виконання програми зводиться до взаємодії об'єктів, яке служить абстракцією реальної взаємодії їх прототипів. Все це разом забезпечило об'єктно-орієнтованому підходу беззаперечне лідерство в галузі розробки програм.

Сьогодні в сімействі мов об'єктно-орієнтованого програмування три найбільш відомих представника: C++, Java і C # (читається Сі шарп). C++ і сьогодні залишається визнаним лідерів в розробці великих і складних програмних систем. Java і C # виросли з C++. Вони мають свою сферу застосування в розподіленому програмуванні і будуть вивчатися нами пізніше.

ІНТЕГРОВАНІ СИСТЕМИ ПРОГРАМУВАННЯ

Для того, щоб комп’ютер міг виконувати програму, її потрібно перекласти на машинну мову. Для цього використовують спеціальні програми-транслятори.

Транслятор – це програма, призначена для перекладу тексту програми однієї мови програмування на іншу. Процес перекладання називається трансляцією.

Розрізняють два типи трансляторів – компілятори та інтерпретатори. Компілятор – це програма, що призначена для перекладу та наступного запам’ятовування повністю всієї програми, яка написана деякою мовою програмування, у програму в машинних кодах.

Процес такого перекладання називається *компіляцією*.

Компілятор створює програму в машинних кодах, яка потім виконується. Для повторного виконання програми компілятор вже не потрібен. Досить завантажити з диска в пам’ять комп’ютера скомпільований раніше варі-ант і виконати його. Існує інший спосіб поєднання процесів трансляції та ви-конання програми. Він називається інтерпретацією.

Інтерпретатор – це програма, яка призначена для повказівкових трансляцій та виконання вихідної програми. Такий процес називається інтерпретацією.

У процесі трансляції відбувається перевірка тексту програми на відповідність до правил мови, яка використовувалася для її опису. Якщо в про-грамі знайдено помилки, транслятор виводить повідомлення про них на при-стрій виведення. Інтерпретатор повідомляє про знайдені помилки після трансляції кожної вказівки програми. Це значною мірою полегшує процес пошуку та виправлення помилок, але суттєво збільшує час трансляції. Компілятор транслює програму набагато швидше, ніж інтерпретатор, але повідомляє про знайдені помилки лише після завершення компіляції всієї програми. Знайти та виправити помилки в цьому випадку важче. Через це інтерпретатори роз-раховані, в основному, на мови, що призначені для навчання програмування. Більшість сучасних мов програмування призначені для розробки складних пакетів програм і розраховані на компіляцію.

Як правило, програми компілятори і інтерпретатори називаються так само, як і мови, для перекладу з яких вони призначені.

Інтегроване середовище програмування – це система програмування, що поєднує редактор для зручного введення і редагування програми, транслятор і налагоджувач помилок.

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.

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*.

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, *C++*, 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 систе-мах для кожного застосування (процесу) надається окрема захи-щена 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 функціо-нування 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 використан-ня типів, що зумовлює зменшення числа помилок під час написан-ня програми.

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

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 вичер-пати 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).

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 дороблятиме мову *Java* так, щоб написані на ній програми можна було запускати з *Web*-бровзерів. Відтоді самостійні *Java*-програми називають *застосуваннями* (applications), а програми, які вико-нуються 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:

0 - простір імен пакета;

1 - простір імен одиниці компіляції (файл класу);

2 - простір імен типу (клас у класі);

3 - простір імен методу;

4 - простір імен локального блоку;

5 - 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 існую-чих пакетів необхідно до тексту програми (першими) додати опера-тори

```
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* нази-вають абстрактний клас. Його введено для реалізації наслідування від декількох класів.

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 програ-мування (API) з наборами пакетів й основні інструменти. Більшість версій *JDK* містять сім інструментів розробки на *Java*:

- компілятор (javac);
- генератор документації (javadoc);
- генератор файлів заголовків і заглушок мови *C++* для
    - *Java* (javah);
- інтерпретатор (java);
- програму перегляду аплетів (appletviewer);
- реасемблер файлів класів;
- 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:

- отримання SDK (переважно з *Internet*);
- встановлення SDK;
- 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).

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.

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.

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()*.

3. Для найменування властивостей використовують ті ж домовленості, що і для методів. Наприклад, властивість можна назвати *thePoint*, а не *ThePoint* чи *thepoint*.

4. Імена констант переважно пишуть великими буквами. Наприклад, *PI*, а не *pi*.
5. Імена методів доступу до властивостей розпочинають з *set* і *get*.

6. Якщо властивість має тип *boolean*, 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 особли-востей, визначальних для всіх програм мовою *Java*. Вона розпочи-нається з коментарію: текст, обрамлений /*...*/, у першому рядку. Цей тип коментарю називають багаторядковим. У мові *Java* є одно-рядкові 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* розріз-няється 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.lang*, 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.

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:

```
javac Example.java
```

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

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".

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.