***Лекція***

**План.**

1. Внесок Java у розвиток Інтернету
   1. Jаvа – аплети.
   2. Безпека.
   3. Переносимість.
2. Байт-код Java
3. Об'єктно-орієнтоване програмування
   1. Інкапсуляція.
   2. Поліморфізм.
   3. Наслідування.
4. **Внесок Java у розвиток Інтернету**

Розвиток Інтернету послужив головною спонукальною причиною для виходу Java на передові рубежі програмування. У свою чергу, Java зробив благотворний вплив на розвиток Інтернету. Ця мова не лише спростила   
веб-проrраммирование, але і поклав початок нового різновиду мережевих програм, званих аплетами, які повністю змінили уявлення про те, яким може бути веб-вміст. Java також дозволив вирішити найбільш складні завдання, що виникають при створенні мережевих програм: забезпечення переносимості і безпеки. Розглянемо детальніше аплети і їх основні властивості.

* 1. **Jаvа–аплети.**

Аплет - це особливий різновид програм на Java, призначена для передачі по Інтернету і автоматичного виконання в середовищі, сформованим Jаvа – сумісним браузером. Аплет завантажується на вимогу клієнтської програми, а для його передачі по мережі втручання користувача не потрібно. Якщо користувач клацає на посиланні, що вказує на документ, який містить аплет, останній буде автоматично скопійований і запущений браузером. Більшість аплетів невеликі по розмірах. Зазвичай вони служать для відображення інформації, що надається серверами або для підтримки введення даних користувачем. Іноді з їх допомогою реалізуються нескладні функції. Наприклад, калькулятор зручніше розмістити у вигляді аплета на стороні клієнта чим виконувати обчислення на стороні сервера. Таким чином, аплет дозволяє переносити деякі функції з сервера на комп'ютер клієнта.

З появою аплетів розширилося коло об'єктів, придатних для вільної передачі у мережевому середовищі. Існують дві категорії об'єктів, якими сервер може обмінюватися з клієнтом: пасивні (статичні) дані і активні виконувані програми (динамічні дані). Наприклад, переглядаючи електронну пошту, ви маєте справу із статичними даними. Навіть якщо в поштовому відправленні пересилається програма, то її код не активізується до тих пір, поки не отримає управління. Аплет, навпаки, є динамічною, самостійно виконуваною програмою для запуску якої не доводиться приймати ніяких заходів. Такі програми виконують роль активних агентів на клієнтських машинах, але ініціалізувалися сервером.

Незважаючи на те що за допомогою динамічних програм можна реалізувати корисні функції, що мають відношення до безпеки і переносимості. Очевидно, що програма, яка завантажується на комп'ютер клієнта і автоматично запускається, не повинна завдавати шкоди клієнтській системі. Крім того, вона повинна допускати виконання в різнотипних середовищах: на різних платформах під управлінням різних операційних систем.

* 1. **Безпека.**

Як відомо, запуск звичайної програми, завантаженої через інтернет, зв'язаний з ризиком, оскільки вона може бути заражена вірусом або служити свого роду "троянським конем" для зловмисного проникнення в систему. А зловмисні дії такої програми можливі через те, що вона отримує несанкціонований доступ до системних ресурсів. Так, вірус, аналізуючи вміст файлової системи локального комп'ютера, може збирати конфіденційні дані, наприклад номери платіжних карт, відомості про банківські рахунки і паролі. Для безпечного завантаження і запуску аплетів на клієнтській машині необхідно усунути саму можливість атаки систему з боку аплета.

Захист від атак реалізується шляхом створення спеціального середовища для виконання аплета, що не дозволяє йому звертатися до ресурсів комп'ютера. За відсутності такого середовища аплет виконуватися не може (далі буде показано, як вирішується подібне завдання). Можливість завантажувати аплет з упевненістю в тому, що він не завдасть шкоди системі належить до найпривабливіших особливостей Java.

* 1. **Переносимість.**

Переносимість є важливою властивістю мережевих програм. Значущість цієї характеристики обумовлена тим, що в мережі можуть бути присутній різнотипні комп'ютери що працюють під управлінням різних операційних систем. Якщо програма на Java призначена для виконання на довільному комп'ютері, підключеному до Інтернету, то повинні існувати способи забезпечення роботи цієї програми в різних системах. Наприклад, один і той же аплет повинен працювати на машинах з різнотипними процесорами, в різних операційних системах і з різними браузерами. Зберігати різні версії аплета для різнотипних комп'ютерів занадто складно, якщо взагалі можливо. Один і той же код повинен працювати на всіх машинах. Таким чином потрібна підтримка процесу генерації переносного коду, що виконується.

1. **Байт-код Java.**

Добитися безпеки і переносимості програм по мережі дозволяє генерований компілятором Java код, що не є виконуваним. Такий код називається байт-кодом. Його оптимізований набір команд, призначених для виконання у виконуючій системі Java, що називають віртуальною машиною Java (Java Virtual Machine - JVМ). Віртуальна машина Java, по суті, є інтерпретатор ***байт-кода. Такий підхід може здатися не зовсім звичайним, оскільки для підвищення*** продуктивності компілятори більшості сучасних мов генерують виконуваний код. Але виконання програми під управлінням віртуальної машини дозволяє дозволити багато складностей, що виникають в роботі веб-приложений.

Трансляція початкового коду Java в байт-код істотно спрощує перенесення програм з одного середовища в іншу, оскільки для забезпечення працездатності коду вистачає реалізувати на кожній платформі віртуальну машину. Якщо на комп'ютері є присутній пакет виконуючої системи, то на нім може працювати будь-яка програма, написана на Java. Незважаючи на те що віртуальні машини на різних платформах можуть бути реалізовані по-різному, вони повинні однаково інтерпретувати байткод. Якби початковий текст програми на Java компілювався у власний код для кожного типу процесора, що взаємодіє з Інтернетом, необхідно було би передбачити окрему версію цієї програми. Таке рішення не можна назвати прийнятним. Отже, виконання байт-кода під управлінням віртуальної машини - - найпростіший шлях до забезпечення переносимості програм.

Виконання програми під управлінням віртуальної машини допомагає також забезпечити безпеку. Віртуальна машина може заборонити програмі виконувати операції, побічні ефекти яких здатні вплинути на ресурси за межами виконуючої системи. Крім того, безпека досягається за допомогою деяких обмежень, передбачених в мові Java.

Як правило, програма, що інтерпретується, виконується повільніше, ніж скомпільована у машинний код. Але для коду Java відмінності в швидкодії не дуже істотні. Адже байт-код оптимізований, і тому програма виконується під управлінням віртуальної машини значно швидше, ніж слід було б чекати.

Незважаючи на те що Java був задуманий як мова, що інтерпретувалася, ніщо не заважає використовувати оперативну (тобто виконувану на льоту) компіляцію байт-кода у власний код процесора для підвищення продуктивності. З цією метою відразу ж після першої реалізації NМ компанія Sun Microsystems почала роботу над технологією HotSpot, у рамках якої був розроблений динамічний компілятор байт-кода. Якщо до складу віртуальної машини входить динамічний компілятор, байт-код по частинах перетвориться у власний виконуваний код. Перетворювати відразу всю програму на Java у виконуваний код недоцільно із-за різноманітних перевірок, які можуть проводитися тільки на етапі виконання програми. Тому динамічний компілятор виконує перетворення коду частями в міру необхідності (звідси і його інша назва - J/Т-компілятор, тобто компілятор, що приходить в дію лише у потрібний момент часу). Більше того, компіляції піддаються не усі фрагменти байт-кода, а лише ті, швидкість виконання яких можна підвищити завдяки компіляції, а інший код інтерпретується. Незважаючи на усі обмеження, властиві динамічній компіляції, вона проте дозволяє істотно підвищити продуктивність програм. І незважаючи на динамічне перетворення байт-кода в виконуваний код, переносимість і захист зберігаються, оскільки JVМ як і раніше бере участь в процесі виконання програм.

1. **Об'єктно-орієнтоване програмування.**

Однією з головних властивостей Java є підтримка об'єктно-орієнтованого програмування (JОП). Об'єктна методологія невід'ємна від Java, а усі програми на Java в тій або іншій мірі є об'єктно-орієнтованими. Тому має сенс коротко розглянути принципи JОП, перш ніж переходити до написання навіть найпростішої програми на Java. Далі ви побачите, як ці принципи реалізуються на практиці.

Об'єктно-орієнтований підхід до програмування дозволяє розробляти досить складні програми. З моменту появи першого комп'ютера методологія програмування зазнала ряд істотних змін, пов'язаних із зростанням складнощі програм. На зорі обчислювальної техніки процес програмування було введення машинних команд в двійковій формі з пульта управління ЕОМ. У той час розміри програм не перевищували декількох сотень команд, і тому такий підхід вважався цілком прийнятним. Потім з'явилася мова асемблера. Символьне представлення машинних команд і процедура компіляції дозволили перейти до створенню складніших програм. У зв'язку з подальшим збільшенням об'єму програмного кода з'явилися мови високого рівня. Вони стали тими інструментами які дозволили програмістам впоратися з поступовим ускладненням програм.

Першим з широко поширених мов високого рівня став FORTRAN. Розробка FORTRAN стала важливим етапом в розвитку мов програмування, але ця мова не цілком підходила для створення легких для читання програм.

У 1960-і роки почало зароджуватися структурне програмування. Згодом для підтримка цього підходу був створені такі мови, як З і Pascal. Завдяки структурованим мовам програмування з'явилася можливість створювати програми середній складності. Головними властивостями структурованих мов стали підтримка незалежних підпрограм, локальних змінних, наявність розширеного набору конструкцій, що управляють, і відсутність оператора GOTO. Але, незважаючи на те що структуровані мови стали потужними інструментами програмування, із збільшенням об'єму і складності проектів їх можливості швидко вичерпалися.

Об'єктно-орієнтоване програмування має на увазі інший підхід. Програми організовуються на основі даних за наступним головним принципом: дані управляють доступом до коду. У об'єктно-орієнтованих мовах програмування визначаються дані і процедури, яким дозволяється обробляти ці дані. Таким чином, тип даних визначає ті операції, які застосовані до цих даних. У усіх об'єктно-орієнтованих мовах програмування, у тому числі і в Java підтримуються три основні принципи ООП : інкапсуляція, поліморфізм і спадкоємство.

Розглянемо кожного з цих принципів окремо.

* 1. **Інкапсуляція.**

*Інкапсуляція є механізмом програмування, об'єднуючим кодом* і дані, якими він маніпулює. Він запобігає несанкціонованому доступу до даних ззовні і їх некоректне використання. У об'єктно-орієнтованих мовах програмування код і дані організовуються в деяку подібність "чорного ящика". В результаті такого об'єднання коду і даних створюється об'єкт. Іншими словами *об'єкт - це компонент, що підтримує інкапсуляцію.*

Дані і код всередині об'єкту можуть бути закритими (private) або відкритими ((pablic). Закритий код або дані доступні тільки елементам, що містяться в тому же самому об'єкті. Тому звернутися до такого коду або даних поза об'єктом неможливо. Якщо код або дані є відкритими, то до них можна звертатися з будь-кого частини програми (не дивлячись на те, що вони знаходяться усередині об'єкту). Як правило відкриті елементи об'єкту використовуються для створення керованого інтерфейсу до його закритим елементам.

Основною мовною конструкцією, що підтримує інкапсуляцію в Java, являється *клас. Класи детальніше розглядатимуться далі, але про них треба сказати декілька* слів вже тепер. Клас визначає тип об'єкту. У нім визначаються як дані, так і код, що виконує певні дії над цими даними. У Java визначення чи так звана специфікація класу, служить для побудови об'єктів. Об'єкти є екземпляри класів. Отже, клас - це ряд "креслень" по яких будується об'єкт. Код і дані у складі класу називаються членами класу. Дані, визначені у складі класу, прийнято називати змінними-членами, або змінними екземпляра а код, що виконує дії над цими даними, - методами-членами, або ***методами.***

***Метод - це термін, яким в Java прийнято означати підпрограму.*** Якщо ви знайомі з мовами С/С++, то, ймовірно, знаєте, що в цих мовах тій самій меті служить термін функція.

* 1. **Поліморфізм.**

***Поліморфізм (від грецького слова, що означає "багато форм") - ця властивість, що дозволяє*** за допомогою одного інтерфейсу звертатися до загального класу дій. Конкретна дія визначається ситуацією. Як приклад, що дозволяє краще зрозуміти принцип поліморфізму, можна привести кермо автомобіля. Кермо (тобто інтерфейс взаємодії) залишається одним і тим же, незалежно від того, який саме механізм рульового управління застосовується в автомобілі, будь то зубчаста, рейкова передача або гідро підсилювач. Таким чином, знаючи, як поводитися з кермом, ви зможете управляти автомобілем будь-якого типу.

Той же самий принцип застосовується і в програмуванні. Розглянемо в якості прикладу стек (структуру даних, організованих за принципом "останнім поступив, - першим обслужений"). Допустимий, в програмі потрібно три різнотипні стеки. Перший стек служить для зберігання цілочислових значень, другий - для зберігання значень з плаваючою точкою і третій - для зберігання символьних значень. Кожен стек реалізується за допомогою одного і того ж алгоритму, незважаючи на те що в стеках зберігаються різнотипні дані. У разі мови, що не підтримує ООП, довелося б створювати три різних набору процедур управління стеками, присвоївши їм різні імена. Але в Java завдяки поліморфізму можна створити один загальний набір процедур управління стеками який діятиме по-різному залежно від конкретного типу стека. Таким чином, знаючи, як працювати з одним стеком, можна поводитися з усіма трьома стеками. Принцип поліморфізму добре ілюструється наступним вираженням: "один інтерфейс - безліч методів". Це означає можливість створення універсального інтерфейсу для групи взаємозв'язаних дій. Поліморфізм спрощує програму завдяки можливості визначити загальний клас дій за допомогою одного і того ж інтерфейсу. Вибрати певну дію (тобто метод) - завдання компілятора, і він вирішує її залежно від конкретних умов. Як програмістові вам не доводиться вибирати метод вручну. Треба лише пам'ятати принципи використання загального інтерфейсу.

* 1. **Наслідування.**

Наслідування - це процес, в ході якого один об'єкт набуває властивість іншого об'єкту. Наслідування має дуже велике значення, оскільки з його допомогою підтримується ієрархічна класифікація. Якщо вдуматися, то знання найчастіше мають ієрархічну структуру.

Без такої ієрархічної структури для кожного об'єкту довелося б наново визначати увесь набір властивостей. Завдяки наслідуванню для об'єкту досить вказати ті властивості, які відрізняють його від інших класів, а інші загальні атрибути він наслідує від своїх батьківських класів. Таким чином, завдяки наслідуванню можливо створити об'єкт, що є екземпляром загальнішого класу.