МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Кафедра обчислювальної техніки

КОНСПЕКТ ЛЕКЦІЙ

з програмного модулю
"Об'єктно-орієнтоване програмування"

Розробник: асистент Алещенко Олексій Вадимович
(посада, П.І.Б.)

Затверджено на засіданні кафедри
Протокол № 11 від 24 травня 2017 р.

Завідувач кафедри ОТ
_____ Стіренко С.Г.
(підпис) (прізвище, ініціали)

Київ – 2017

«Вступ в ООП на мові програмування Java»

Парадигма(ідеологія) об’єктно-орієнтованого програмування (ООП) в даний час стала домінувати в програмному світі [1]. Вона прийшла на зміну структурній техніці програмування, що була розроблена в 1970. Java є повністю об’єктно-орієнтованою мовою, тому потрібно засвоїти принципи ООП якомога краще.

Об'єктно-орієнтоване програмування сягає своїм корінням до створення мови програмування Симула в 1960-тих роках, одночасно з посиленням дискусій про кризу програмного забезпечення. Разом із тим, як ускладнювалось апаратне та програмне забезпечення, було дуже важко зберегти якість програм. Об'єктно-орієнтоване програмування частково розв'язує цю проблему шляхом наголошення на модульності програми [2].

В структурному програмуванні передбачалася розробка окремих алгоритмів та процедур для вирішення конкретної задачі. Такий підхід виправдує себе для невеликих задач, проте для великих проектів ООП більш виправдане. В літературі можна знайти приклад, що для реалізації простого веб-браузера необхідно близько 2000 процедур при структурному програмування. При використанні ж ООП можна створити 100 класів з приблизно 20-ма процедурами(далі методами) в кожному з них. Таким чином набагато простіше шукати помилку серед 20-ти методів одного класу ніж шукати її серед 2000 методів.

Як уже говорилося раніше: клас – це певний шаблон, який слугує для створення об’єктів. Ви можете розробити власний клас, а можете отримати його від інших розробників. В інтернеті зараз є чимало бібліотек класів різноманітного призначення як безкоштовних так і платних. Наприклад, бібліотеки для промальовування різноманітних діаграм у своїх програмах і т.п.

ООП в java базується на ряді понять (або ж концепцій):

- **Клас** - певна абстрактна сутність, наприклад, "Пес", "Кіт", "Автомобіль", "Ціна товару", що на програмному рівні представлена змінними (полями даних) та методами, що оперують над цими полями даних.

- **Об'єкт** - конкретний екземпляр класу. Наприклад, зелена "Тойота" вашого сусіда є екземпляром класу "Автомобіль". По суті, це клас, поля якого ініціалізовані і він завантажений у пам'ять комп'ютера. На основі одного класу, можна створити безліч об'єктів.

- **Успадкування** (або ж "спадкоємство") - утворення нових класів на основі інших.

- **Інтерфейс** - посилальний тип даних. Інтерфейси схожі на класи, проте їхні поля даних є константами, а методи не реалізовані. Об'єкти на основі інтерфейсів не створюються, проте класи можуть реалізовувати певний інтерфейс. І через об'єктну змінну інтерфейсного типу можна викликати реалізації даних методів.

- **Пакети** - каталоги, у яких розміщуються класи. Таким чином ми можемо використовувати однойменні класи, оскільки їхнє розрізнення іде не тільки за іменами, але й за розміщенням їх у каталогах (пакетах).

Насправді доволі часто різні автори по різному виділяють головні концепції ООП. Причиною є те, що в ООП справді доволі багато понять. І з плином часу та розвитком програмування їх лише більшає. Різноманітні поняття доволі тісно взаємопов'язані між собою. В літературі часто виділяють наступні три концепції і навіть вказують, що вони основні для ООП ("три кити"):

- **інкапсуляція** (incapsulation) - концепція побудови класів через закриття(капсулювання) їхньої реалізації.

- **успадкування** (inheritance) - створення одних класів на основі інших

- **поліморфізм** (polymorphism) - можливість використання батьківських класів замість класів нащадків. По суті є частиною реалізованої в мові концепції успадкування.

Деякі теоретики додають до цих трьох ще "**абстрагування**". Власне коли програміст створює клас, він створює певну абстракцію, модель чогось із реального світу. Ряд теоретичних книг побудовані на тому, як потрібно створювати класи, їхні ієрархії, зв'язки між ними. Проте переважно вони настільки теоретичні, що практичні програмісти часто питання абстрагування вирішують виходячи із конкретної задачі, яку потрібно вирішити. Щоправда є і винятки. В даний час виділяють так звані *Патерни проектування*, ряд шаблонів чи то зразків того, як найбільш ефективно вирішити деякі задачі в ООП. Проте для того, щоб їх можна було освоїти, необхідне ґрунтовне вивчення об'єктно-орієнтованого програмування і зокрема наведених вище понять. Поняття з обох вищенаведених списків є важливими для ООП і потрібно розуміти, що є що і як працює. Саме всім цим поняттям та їхній реалізації в мові програмування Java і присвячений даний розділ.

Дані в об’єкті (тобто глобальні змінні даного об’єкту) називаються полями екземпляру класу. Процедури, які оперують даними називаються методами. Специфічний об’єкт, який є екземпляром певного класу матиме специфічні значення екземплярних полів. Сукупність цих значень називається станом об’єкта. Коли викликається певний метод стан об’єкту може змінюватися. Згідно принципу інкапсуляції намагаються, щоб доступ до полів екземпляру здійснювався лише за допомогою певних методів.

Об’єкт – це конкретна реалізація певного класу. На основі одного класу може бути створено безліч об’єктів. При цьому в об’єктах виділяють:

- Поведінку об’єкту – що можна з робити з даним об’єктом, або які методи можна застосовувати до нього
- Стан об’єкту – те як об’єкт змінюється, коли Ви застосовуєте його методи
- Ідентичність об’єкту – відмінність об’єкту від інших об’єктів. Об’єкти можуть мати однаковий стан, проте все рівно вони ідентифікуються як різні об’єкти.

Розробці класів присвячена чимала кількість літератури. Переважно назва класу складається з іменника, оскільки вони представляють собою певну сутність, а методи класу дієсловами, оскільки вони визначають його поведінку. Коли система, яка розробляється доволі велика усі її компоненти уже важко утримувати розробнику в голові. Тому в таких випадках часто використовують UML діаграми, на яких можуть вказувати як взаємозалежності класів у вигляді різноманітних стрілок так і стан та поведінку класів. Ви можете звернутися до відповідної літератури, якщо вас зацікавила дана тема.

Щоб створити об’єкт певного класу програміст викликає один з конструкторів даного класу. Спеціальні методи, які покликані задати об’єкту початковий стан і носять його ім’я. Наприклад стандартна бібліотека джави містить клас Date, який описує момент часу. Як то “December 31, 1999, 23:59:59 GMT”

```
new Date(); // створюємо об’єкт, який містить поточний час
```

Ми можемо зразу ж вивести поточний час системи на екран:

```
System.out.println(new Date());
```

Проте такий підхід застосовується якщо об’єкт потрібен нам в програмі лише раз. Якщо ж його потрібно буде використати повторно, то для цього використовуються об’єктні змінні. Змінні, які посилаються на певний об’єкт.

Так для нашого випадку, наступний рядок створить об’єктну змінну типу Date, яка посилатиметься на створюваний нами об'єкт.

```
Date curDate = new Date();
```

Значення об’єктної змінної може бути змінене, так що буде посилатися на інший об’єкт, або ж взагалі встановлене в null, тобто не вказувати на жодний об'єкт. Через об'єктні змінні можна здійснювати доступ до полів об'єкта та його методів.

Методи в java – це аналог підпрограм, функцій, процедур в інших мовах програмування. За допомогою методів ми виносимо текст повторюваного коду програми окремо в тіло методу, після чого можна викликати даний метод з будь-якого місця програми, безліч разів.

Спрощене оголошення та визначення методу, який ми зараз будемо використовувати, має вигляд:

```
тип_повернення  назва_методу(параметри){


//тіло методу;
інструкція1;
інструкція2;
…..
інструкціяN;
}
```

Тип_повернення – результат виконання методу, наприклад він може повертати об’єм сейфу, тоді тип_повернення буде double. Якщо метод нічого не повертає, то вказується слово ключове слово void.

Виклик методу здійснює наступна інструкція: назва_методу(параметри). Взагалі в термінології мов програмування для виклику методу використовуються аргументи, а в самому методі – це уже параметри, оскільки передаються лише значення змінних, а не самі змінні. Аргументи та параметри повинні бути одного і того ж типу. Якщо ми передаємо цілочисельне значення, то і параметр повинен бути цілочисельним і т.п. Тобто в методі створюються нові змінні. В деяких мовах, наприклад в C++, як аргумент можна передати посилання на певну змінну, таким чином її можна буде модифікувати в функції через вказівник на дану змінну. В Java в метод передаються лише значення змінних, тому розрізнення аргументів і параметрів менш суттєве і ми там і там використовуватимемо термін «параметр». Об'єкти ж в методи передаються по посиланню. Тобто, при передачі в якості аргументу об'єкта, буде передане посилання на об'єкт, а не створений новий об'єкт.

Повернемось до нашої програми. Повторюваними є присвоювання значень змінним, обчислення та виведення об’ємів. Тому корисно буде створити два методи safeValue (double width, double height, double depth) та safeVolume (). Перший для присвоєння змінним значень, а другий для обчислення об'єму.

Модифікована програма матиме вигляд:

```
class Safe {

    double width = 10; // поля класу можуть бути ініціалізовані з самого початку

    double height = 10;
```

```
    double depth = 10;
    double safeVolume = 0;

    // метод присвоєння значень змінним
    void safeValue(double pWidth, double pHeight, double pDepth) {
        width = pWidth;
        height = pHeight;
        depth = pDepth;
    }

    // метод для обчислення об'єму сейфа
    double safeVolume() {
        return width * height * depth;
    }
}

public class CoinVolume {
    public static void main(String[] args) {
        double width1 = 10, height1 = 20, depth1 = 40;
        Safe mySafe1 = new Safe(); //створюємо перший сейф
        Safe mySafe2 = new Safe(); //створюємо другий сейф
```

```
        //задаємо розміри сейфу
        //викликаємо метод safeValue() класу Safe, що ініціалізує поля об'єкту
        mySafe1.safeValue(width1, height1, depth1);
        mySafe2.safeValue(10.0, 15.0, 15.5); //можна і так

        //виводимо на екран об'єми сейфів
        //для чого викликаємо метод safeVolume(), який повертає обчислений об'єм кожного сейфу
        System.out.println("Об'єм 1-го сейфу=" + mySafe1.safeVolume());
        System.out.println("Об'єм 2-го сейфу=" + mySafe2.safeVolume());
    }
}
```

Як бачимо код програми спростився. Тепер набагато легше модифікувати програму, додаючи нові сейфи. Крім того можна, наприклад, вивести повідомлення з інформацією про об’єм, розмістивши його в метод safeVolume() або зовсім в окремий метод, наприклад, printVolume().

Слід зауважити, щодо назв змінних-параметрів. Так для висоти, ширини та глибини в методі safeValue() вибрані назви pWidth, pHeight, pDepth. Вони могли б мати назви і просто width, height, depth, але тоді б вони перекрили доступ до однойменних змінних класу. В такому випадку, щоб звернутися до змінних класу з методу необхідно вживати ключове слово this. Наприклад: this.height=height – тут ми присвоюємо змінній класу this.height одержаний методом параметр height.

Джава дозволяє групувати класи в своєрідні колекції, які називаються пакетами. Це дозволяє, зокрема розмежувати класи з одним і тим же ім'ям. Так якщо ви розробили клас Car, то цілком логічно, що такий клас вже не раз розробляли.

Стандартна бібліотека Java поширюється з набором пакетів. Наприклад, java.lang, java.util, java.net і т.п. Все що необхідно для створення пакету, це створити звичайну папку і в ній розмістити ваші класи. Якщо папка вкладена в іншу утворюється ієрархія класів. Так всі стандартні класи Джава розміщені в каталогах java та javax.

Розробники Джава рекомендують розміщувати пакети по принципу доменних імен, лише в зворотньому порядку, наприклад: com.horstmann.corejava. Якщо ви або ваша фірма має доменне ім'я – це доволі зручний спосіб розміщення, оскільки забезпечує унікальність найменування пакету.

Імпортування пакетів

Ваш клас може використовувати класи з власного пакету або всі відкриті класи з іншого пакету. Проте, якщо класи знаходяться в іншому пакеті, їх потрібно імпортувати. Це можна зробити двома способами:

- вказувати повну назву пакету перед використовуваним класом:

java.util.Date today = **new** java.util.Date();

Такий спосіб доволі незручний.

- Другий спосіб – це використати інструкцію import, яка ставиться на початку сирцевого коду програми.

import java.util.Date; *// імпортуємо класс*

можна також імпортувати цілий пакет, таким чином не лише класс Date буде доступний, але й усі класи, які знаходять в одному каталозі з ним:

import java.util.*;

Таким чином в коді програми можна писати:

```
Date today = new Date();
```

Щоб визначити, який пакет вам потрібно імпортувати можна скористатися документацією класів. Деякі середовища розробки дозволяють зробити це автоматизовано при натисненні певного пункту меню. В NetBeans наприклад можна клацнути правою кнопкою миші будь-де у вікні редагування коду і вибрати пункт "*Fix Import*". Щоправда дана команда також підчищає рядки з імпортом і забирає імпорт тих пакетів, які не використовуються.

Інколи все ж коли ви імпортуєте пакет не вказуючи ім'я класу, то можуть виникати конфлікти імен. В таких випадках необхідно, або точно вказати, який клас ви імпортуєте, або якщо вам необхідно використовувати однойменні класи з двох різних пакетів, то прийдеться використовувати повне ім'я пакету при роботі з класом та його методами [1].

«Робота з масивами в Java»

Масив - це впорядкований набір однотипних елементів, на які посилаються по спільному імені. Це доволі зручний засіб групування інформації. Масиви можна створювати з елементів будь-якого типу. До конкретного елементу в масиві звертаються по індексу (номеру). Вони можуть бути як одновимірні так і багатовимірні.

Одновимірні масиви

Одновимірні масиви - це список однотипних елементів. Загальний формат оголошення такого масиву:

тип-елементів назва-масиву[];

Наприклад:

int month_days[]; *// масив цілих чисел*

Існує також інша форма оголошення масиву:

int[] month_days;

Проте для того, щоб масив почав існувати необхідно виділити під нього пам'ять, за допомогою операції *new*.

назва-масиву = new тип-елементів [розмір];

де *розмір* - планована кількість елементів у масиві.

month_days = **new** int[12];

або зразу ж:

int month_days[] = **new** int[12];

Таким чином відбувається виділення пам'яті під масив і ініціалізації елементів масиву нулями. В подальшому можна напряму звертатися до елементів масиву вказуючи індекс у квадратних дужках. Нумерація елементів в масиві в java відбувається з нуля. Тобто в наведеному прикладі звернення до першого(нульового) елемента - month_days[0], а до останнього - month_days[11]. Java не дозволить програмі звернутися поза межі масиву, щоправда помилка буде вказана лише на етапі виконання програми через викидання винятку(виключення).

month_days[5] = 30;

System.out.println(month_days[5]);

Масиви також можна ініціалізувати зразу ж при їхньому оголошенні, не використовуючи операції new, аналогічно як це відбувається при роботі з простими типами даних.

Наступний приклад зразу ж при оголошенні ініціалізує масив month_days[] кількістю днів в місяцях.

```
public class DaysOfMonth {

    public static void main(String[] args) {

    int month_days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}; //оголошуємо та ініціалізуємо масив

    System.out.println("Травень має " + month_days[4] + " день"); // вивід на консоль

    }

}
```

Результат виконання на екрані:

Травень має 31 день

Наступний приклад демонструє знаходження максимального числа в одновимірному масиві.

```
public class ArrayMax {

    public static void main(String[] args) {

        double array[] = {1.1, 2.2, 1.1, 3.2, 1.2, 2.1};

        double max = array[0];

        for (int i = 0; i < 6; i++) {
```

```
            if (max < array[i])
                max = array[i];
        }
        System.out.println("Максимальне число в масиві: " + max);
    }
}
```

Як бачимо спочатку змінній max присвоюється значення нульового елементу масиву, після чого в циклі іде послідовне порівняння з кожним наступним числом до останнього. Якщо при порівнянні чергове значення в масиві більше за максимальне в змінній max, то змінній max присвоюється дане значення. Як Ви уже зрозуміли, по закінченню циклу у змінній max міститиметься максимальне значення, яке і буде виведене на консоль:

Максимальне число в масиві: 3.2

В Java масиви є спеціальними об’єктами (про об’єкти детальніше в наступних розділах присв’ячних об’єтно-орієнтованому програмуванню), що забезпечує деяку додаткову функціональність масивам. Зокрема, можна дізнатися довжину масиву таким чином array.length. Для вищенаведеного прикладу можна замінити рядок з циклом таким чином:

```
        for (int i =0; i < array.length; i++){
```

Багатовимірні масиви

Багатовимірні масиви по суті – це масив масивів. Робота з багатовимірними масивами подібна до роботи з одновимірними. Відмінність лише в тому, що використовуються додаткові квадратні дужки. Переважно використовуються двовимірні масиви, які служать для роботи з табличними

даними та трьохвимірні масиви. Двовимірний масив та трьохвимірний, можна оголосити наступним чином:

int twoD[][] = **new** int [4][5]; *//створення масиву 4x5*

int threeD[][][] = **new** int[5][5][5]; *//створення масиву 5x5x5*

Для двовимірного лівий індекс означає номер рядка, а правий номер стовпця. Це можна уявити наступним чином:

[0,0][0,1][0,2][0,3][0,4]

[1,0][1,1][1,2][1,3][1,4]

[2,0][2,1][2,2][2,3][2,4]

[3,0][3,1][3,2][3,3][3,4]

Трьохвимірний масив можна уявити у вигляді куба. Крім номера рядка і номера стовпця, додається ще індекс елемента вглибину.

Наступна програма створює масив 5 на 4, заповнює його випадковими числами і виводить на екран.

import java.util.Random; *// імпортуємо клас Random*

public class RandomArray {

public static void main(String[] args) {

int m = 5, n = 4; *//оголошуємо і ініцілізуємо змінні з розмірами масиву*

int Array[][] = **new** int[m][n]; *//оголошуємо і ініціалізуємо масив*

```
Random generator = new Random();        // створюємо генератор випадкович чисел
int gn;                                 //змінна в яку буде записуватися згенероване генератором число


/* заповнюємо масив випадковими числами */
for (int i = 0; i < m; i++)             //проходимось по стовпцях
   for (int j = 0; j < n; j++) {        //проходимось по рядках
      gn = generator.nextInt(100);  //генерація випадкового числа від 0 до 100;
      Array[i][j] = gn;             //записуємо згенероване випадкове число
   }


/* Виводимо результат */
for (int i = 0; i < m; i++) {
   for (int j = 0; j < n; j++)          // зверніть увагу на відсутність фігурної дужки
      System.out.print(Array[i][j] + "     ");//даний рядок відноситься до масиву по j
      System.out.println();             //виводимо символи переводу каретки і нового рядка
   //після кожного проходження стовпцевих елементів рядка
```

```
        }
    }
}
```

В результаті на екрані одержимо:

```
94  47  65  0
99  20  60  69
80  33  63  73
35  50  48  81
39  19  4  85
```

В наведеному прикладі в кожному рядку однакова кількість елементів(стовбців). В Java можна створити двовимірні масиви з різною кількістю елементів в рядках.

```
int twoD[][] = new int[5][]; //створюємо двовимірний масив з 5-ма рядками
twoD[0] = new int[5]; // виділяємо пам'ять для 5-ти елементів нульового рядка
twoD[1] = new int[4]; // перший рядок матиме 4-ри елементи
twoD[2] = new int[3]; // другий - 3
twoD[3] = new int[2]; // третій - 2
twoD[4] = new int[1]; // четвертий – 1
```

Використання таких нерівних (нерегулярних) масивів не рекомендується, оскільки з ними важче працювати і можна припуститися ряд помилок, але в деяких ситуаціях можуть бути доволі корисними.

Як і з одновимірними масивами. Ми можемо зразу ж ініціалізувати масив необхідними значеннями при його оголошенні.

Приклад - Array2.java

```
public class Array2 {
   public static void main(String[] args) {


      int[][] Array= {
         {5, 6, 1, 3},
         {3, 4, 2, 1},
         {1, 2, 2, 2}
       };
       for (int i = 0; i < 3; i++){
         for (int j = 0; j < 4; j++)
             System.out.print (Array[i][j]+"    ");
         System.out.println();
       }
   }
}
```

Результат виконання:

```
5    6    1    3
3    4    2    1
```

1 2 2 2

«Робота зі строками в Java»

Рядок (англ. *string*) в java — це послідовність символів Юнікоду. Наприклад: "Абвгґд". В багатьох мовах програмування рядки зберігаються у масивах. Проте в java рядки представляють собою окремий об'єктний тип. Для рядків може використовуватись оператор "+", що здійснює конкатенацію(з'єднання) двох окремих рядків. Так в розглянутих навчальних прикладах конкатенація використовується при виводі на консоль System.out.println("Один" + "два" + "=" + "Три"). Для здійснення інших дій з рядками, як то пошук символів та буквосполучень, використовуються методи класу String.

Спецсимволи

Оголошення змінної і присвоєння їй рядка відбувається наступним чином:

String str = "Це рядок";

Для того щоб створити рядок, необхідно взяти літери в подвійні лапки: "це рядок". Також можна створити пустий рядок "". Якщо необхідно використати спецсимволи, то застосовують зворотню косу (backslash) "\". Наприклад, помістити в рядок внутрішні подвійні лапки можна таким чином "\" - це лапка". Аналогічно для одинарних лапок, недрукованих символів, тощо. Найчастіше коса застосовується для символу нового рядка:

System.out.println("Це перший рядок тексту, \na це другий текстовий рядок.");

Таким чином рядок (string) може містити текст з кількома текстовими рядками (в англ. застосовують термін multiline string — *багатолінійний*

рядок, *багаторядковий рядок*або ж *багаторядкова стрічка*). Проте власне клас String не дуже підходить для зберігання великих об'ємів текстових даних та маніпуляцій з ними. Для цього краще застосовувати класи StringBuilder або ж StringBuffer.

Конкатенація

Як вже було сказано конкатенація - це поєднання двох рядків, що здійснюється за допомогою оператора "+". Таким чином наступні рядки здійснюють одне й те саме:

String str = "Це рядок";

String str = "Це"+" рядок";

При використанні конкатенації рядків з іншими типами даних відбувається автоматичне приведення до типу String.

System.out.println(2 + "+" + 2 + "=" + "4 (чотири)");

String str3 = "цифра " + 5; *//String + int дає String "цифра 5"*

Також з'єднання двох рядків можна здійснити за допомогою методу concat():

String strEnd = "рулить";

String str = "Java ".concat(strEnd); *//в результаті str="Java рулить"*

Робота з рядками

В java об'єкти класу String не можна змінювати. На перший погляд — це додає проблем при роботі, але насправді це не так. Не можна змінювати сам рядок в пам'яті комп'ютера, але змінній, яка посилається на певний рядок, можна призначити інший рядок.

String str = "Це";

```
String str2 = "рядок";

String str3 = "555";

str = str3; //так можна

str = str + " " + str2;   //і так можна
```

Дію обмеження ви відчуєте, лише коли захочете, наприклад, замінити букву "е" на якусь іншу, або змінити її регістр. В інших мовах - це можна зробити без проблем. В java потрібно утворити новий рядок. Скопіювавши, наприклад, букву "Ц" і додавши до неї "Е". Рядок, на який вже не посилається жодна змінна, буде видалений з пам'яті комп'ютера автоматичним прибиральником сміття java.

В разі, якщо ж все ж таки необхідна маніпуляція з рядком напряму, то для таких цілей існують споріднені із String класи. Зокрема, StringBuffer — корисний, при роботі з великими об'ємами текстових даних, читання з файлу і т.п.

Щоб здійснити такі дії як пошук, заміна і т.п. в класі String існує чималий набір методів. Так, щоб дізнатися довжину рядка можна скористатися методом length():

```
String str = "Це рядок";

int strLength = str.length();

int str2Length = "Це рядок".length(); //можна і так
```

Як бачимо виклик методів відбувається як при роботі із класами, з використанням оператору «.» (точка): «об'єкт.метод()» або «об'єктна_змінна.метод()».

Підрядки

Для того, щоб одержати частину рядка (підрядок) з іншого більшого рядка можна скористатися методом substring(pos1, pos2) класу String.

Наприклад:

String greeting = "Hello";

String s = greeting.substring(0,3); *//скопіювати з greeting символи з 0 до 3, тобто 0, 1 та 2 – "Hel"*

В результаті було скопійовано перших три символи. Символи в рядку нумеруються починаючи з нуля. Можна розрахувати довжину отриманого рядка: pos2-pos1. В даному випадку 3-0=3;

Як вже згадувалось ми, не можемо змінювати рядки. Тому, якщо ми, наприклад, захочемо змінити рядок Hello на Help без застосування додаткової змінної, можна скористатися методом substring() та оператором конкатенації «+»:

greeting = greeting.substring(0, 3) + "p!";

В результаті в greeting="Help!". Таким чином ми не змінюємо сам рядок, а присвоюємо змінній інший рядок.

Порівняння рядків

Для того, щоб порівняти два рядки на рівність можна скористатися методом equals():

str1.equals(str2);

Метод повертає true, якщо str1 та str2 рівні, інакше false. Можна також, замість змінних str1, str2 застосовувати рядкові константи. Наприклад:

«Hello».equals(greeting);

Згаданий метод порівнює рядки з врахування регістру символів. Для того, щоб не враховувався регістр при порівнянні існує метод equalsIgnoreCase().

"Hello".equalsIgnoreCase("hello").

Не використовуйте оператор ==, щоб перевірити рядки на рівність. Таким чином лише перевіряється, чи рядки розташовуються за одним місцем в пам’яті, крім того, результат може бути неоднозначним. Так, якщо ми

```
String str1 = "Hello";

String str2 = "Hello";

if (str1 == str2)

    System.out.println("рівні");

else

   System.out.println("не рівні"); //скоріше всього на екрані отримаємо напис «рівні».
```

Неоднозначність результату пов’язана з тим, що константні рядки в java можуть специфічним чином розподілятися (share) в пам'яті для спільного використання змінними. Хоча може бути, що існуватиме багато копій однакових рядків.

Зауважте, що це не відноситься до рядків, які створені в результаті операцій «+» чи substring:

```
String str1 = "Hello";

String str2 = str1.substring(0,3) + "lo";

if (str1 == str2)

   System.out.println("рівні");
```

else

System.out.println("не рівні"); *//скоріше всього на екрані отримаємо напис «не рівні».*

«Методи класу. Перевантаження, статичність та доступ»

В мові Java в межах одного класу можна визначити два або й більше методів під одним іменем, що мають параметри, які відрізняються або кількістю, або типом. Такі методи називаються перевантаженими, а сам процес як **перевантаження** (англ. *overloading*) методів. Це один із способів реалізації поліморфізму.

Наступний приклад демонструє перевантаження методу sum:

```
public class ProgramSum{
    public static void main(String[] args) {
        int a=2, b=3, k=4;
        double a1=2.10, b1=4.20, k1=5.30;
        int sum=sum(a, b, k);
        double sum2=sum(a1, b1, k1);
        System.out.println("Сума трьох цілих чисел дорівнює: "+sum);
        System.out.println("Сума трьох дробових чисел дорівнює: "+sum2);
    }
    public static int sum (int a, int b, int c){
        return a+b+c;
```

```
    }
    public static double sum (double a, double b, double c){
        return a+b+c;
    }
}
```

В результаті на екрані отримаємо:

Сума трьох цілих чисел дорівнює: 9

Сума трьох дробових чисел дорівнює: 11.600000000000001

Як бачимо тіло методів практично не відрізняються. Відрізняються лише типом параметрів, що приймаються і типом параметрів, що повертаються, хоча останнє може бути однаковим, або й взагалі метод може нічого не повертати. Цікаво, що якщо б не було першого методу з цілочисловими вхідними параметрами, то можливий виклик другого методу без приведення змінних до типу.

```
public class ProgramSum{
    public static void main(String[] args) {
        // TODO code application logic here
        int a=2, b=3, k=4;
        double a1=2.10, b1=4.20, k1=5.30;
        double sum=sum(a, b, k);
        double sum2=sum(a1, b1, k1);
        System.out.println("Сума трьох цілих чисел дорівнює: "+sum);
```

```
        System.out.println("Сума трьох дробових чисел дорівнює: "+sum2);


    }
    public static double sum (double a, double b, double c){
        return a+b+c;
    }
}
```

Результат:

Сума трьох цілих чисел дорівнює: 9.0

Сума трьох дробових чисел дорівнює: 11.600000000000001

Зверніть увагу, що результат у першій стрічці дробовий. Одержаний результат отримали через те, що Java здійснила автоматичне перетворення типів int у double. Проте зворотнє перетворення не здійснюється автоматично, якщо було б навпаки і існував лише перший метод, то при спробі виклику з double параметрами ми б отримали помилку компіляції. В таких випадках потрібно здійснювати явне приведення типів.

В наступному прикладі відбувається виклик трьох перевантажених методів. Додано метод sum() з одним цілочисловим і одним дробовим параметрами.

```
public class ProgramSum{
    public static void main(String[] args) {
        int a=2, b=3, k=4;
        double a1=2.10, b1=4.20, k1=5.30;
```

```
        System.out.println("Сума трьох цілих чисел дорівнює: "+sum(a, b, k));
        System.out.println("Сума трьох дробових чисел дорівнює: "+sum(a1, b1, k1));
        System.out.println("Сума одного цілого числа та одного дробового дорівнює: "+sum(a, k1));
    }
    public static int sum (int a, int b, int c){
        System.out.print("1-й метод.");
        return a+b+c;
    }
    public static double sum (double a, double b, double c){
        System.out.print("2-й метод.");
        return a+b+c;
    }
    public static double sum (int a, double b){
        System.out.print("3-й метод.");
        return a+b;
    }
}
```

Результат виконання.

1-й метод.Сума трьох цілих чисел дорівнює: 9

2-й метод.Сума трьох дробових чисел дорівнює: 11.600000000000001

3-й метод.Сума одного цілого числа та одного дробового дорівнює: 7.3

Можна перевантажити також методи із змінною кількістю аргументів:

static public double sum(double...nums){...}

static public double sum(int...nums){...}

Проте потрібно зважати, що sum(double...nums) та sum(double k, double...nums) буде сприйнято компілятором як методи з однаковою сигнатурою і видасть помилку, що в коді є неоднозначність. Це пояснюється тим, що такі оголошення компілятор при розборі перетворює у методи, що приймають масив і в даному випадку компілятор створить аналоги методів з одними і тими ж вхідними параметрами. Тож два однакові методи не можуть існувати в коді [1].

Джерела:

1. "Освоюємо Java" - https://uk.wikibooks.org/wiki/%D0%9E%D1%81%D0%B2%D0%BE%D1%8E%D1%94%D0%BC%D0%BE_Java.
2. "Об'єктно-орієнтоване програмування" - https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F.