Лабораторна робота № 1
Основні елементи мови Java

Мета роботи: вивчити основні елементи мови Java, які будуть необхідні для побудови мережевих клієнт/серверних програм на Java.

Теоретична частина

Коротка історія розвитку мови Java

Історія мови Java починається з 1991 року, коли група інженерів з компанії Sun під керівництвом Патріка Нотона (Patrick Naughton) та члена Ради директорів (і різнобічного фахівця) Джеймса Гослинга (James Gosling) зайнялася розробкою мови, яку можна було використовувати для програмування побутових пристроїв. Так як подібні пристрої не потребують багато енергії та мають невеликій об'єм пам'яті, то мова повинна була бути маленькою та генерувати дуже компактні програми. Окрім того, оскільки різні виробники могли вибирати різні процесори, то було важливо не прив'язуватися до конкретної архітектури. Цей проект отримав кодову назву “Green”.

Після ряду подій у 1996 році компанія Sun випустила першу версію Java. Через декілька місяців після неї з'явилася версія Java 1.02. Користувачі швидко зрозуміли, що версія Java 1.02 не підходить для розробки серйозних програм. У версії Java 1.1 були усунені найбільші недоліки, набагато поліпшені засоби відображення та реалізована нова модель подій для програмування графічного інтерфейсу користувача.

У 1998 році побачила світ версія Java 2 Standard Edition Software Development Kit Version 1.2 (стандартна редакція пакету інструментальних засобів для розробки програмного забезпечення на мові Java 2, версії 1.2).

Окрім Standard Edition були запропоновані ще два варіанти: Micro Edition (“мікроредакція”) для портативних пристроїв, наприклад для мобільних телефонів, та Enterprise Edition (редакція для корпоративних програм).

Далі були випущені версії 1.3 та 1.4. У версії 5.0 мова Java зазнала найбільш серйозної модифікації з моменту реалізації версії 1.1. (З початку версія 5.0 мала номер 1.5, але на конференції JavaOne у 2004 році була прийнята нова нумерація версій).

Створення консольних програм на Java

Для створення простіших консольних програм на мові Java необхідно, щоб на комп'ютер з будь-якою операційною системою був встановлений пакет Java SDK (Software Development Kit) версії 1.4 або більш нової. Для запуску програм на Java потрібна встановлена віртуальна машина Java, яка входить до середовища Java Runtime Environment. Подивитись, яка з версій Java встановлена можна за допомогою наступної команди:

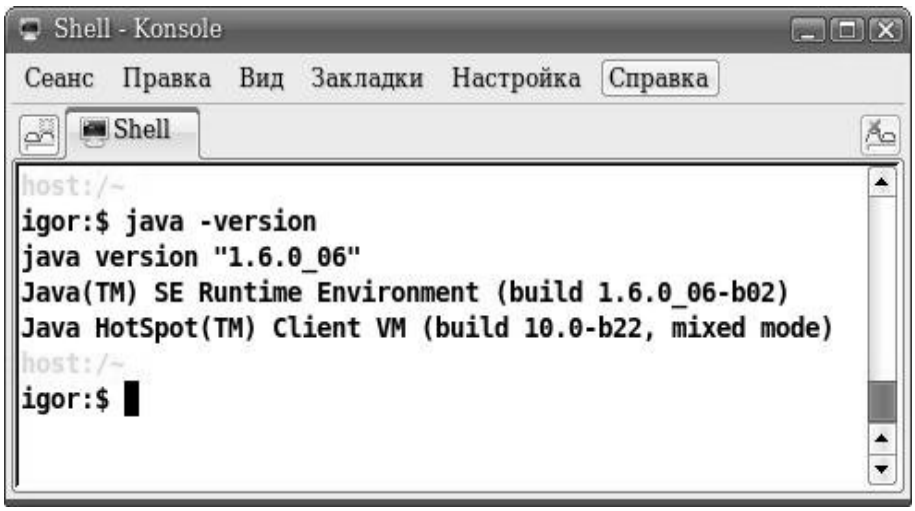

Рис. 1.1. Команда java дозволяє запустити на виконання програму на мові Java

Для набору коду простіших програм на Java може бути задіяний звичайний текстовий редактор, наприклад, *Notepad* (в MS Windows) або *Kate* (у GNU/Linux з KDE). Побудуємо простішу консольну програму на Java, програмний код якої наведено на рис. 1.2 та виконаємо компіляцію (команда *javac*) і запуск (команда *java*). Зверніть увагу на наступні моменти:

1. Регістр літер має значення.
2. У випадку виконання команди *java* регістр символів у назві програми не повинен відрізнятися від назви класу (у даному випадку це Welcome).
3. Компілятор викликається з файлом вихідного коду на Java і в імені файлу необхідно вказати розширення. Розширення не потрібне при виклику команди *java*.

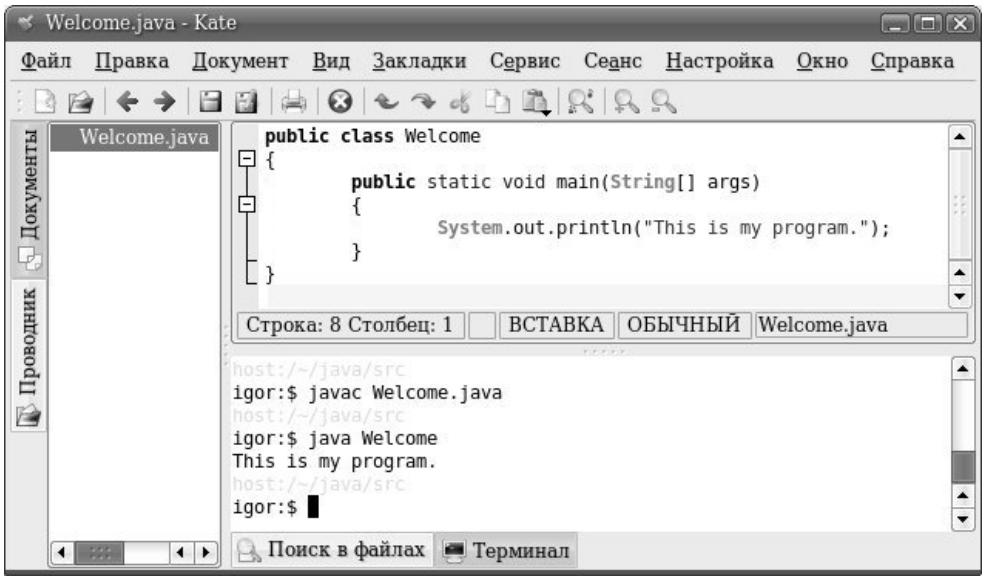

Рис. 1.2. Код консольної програми на Java у редакторі Kate
та термінальні команди компіляції і запуску

Команда *javac* – це компілятор, який створює файл з розширенням *.class,* який потім можуть бути виконані за допомогою віртуальної машини Java.

Далі розглянемо програму для складання двох чисел цілого типу. Вона складається з двох частин. У першої йде опис змінних класу – два цілих числа *a* та *b*. Друга частина реалізує операцію складання. Різні варіанти реалізації кода цієї простої програми наведені нижче:

```
public class MySumm
{
 public static int a;
 public static int b;
 public static void main(String[] args)
 {
   a = 5;
   b = 10;
   int c = a + b;
  System.out.println("Summa A+B = " + Integer.toString(c));
 }
}
```

```
public class MySumm
{
 public static void main(String[] args)
 {
   int a = 5;
   int b = 10;
   int c = a + b;
  System.out.println("Summa A+B = " + Integer.toString(c));
 }
}
```

```
public class MySumm
{
 public static void main(String[] args)
 {
 System.out.println("Summa A+B = " + Integer.toString(5+10));
 }
}
```

```
public class MySumm
{
 public static void main(String[] args)
 {
   System.out.print("Summa A+B = ");
   System.out.println(5+10);
 }
}
```

Зверніть увагу, як можна перевести значення з цілого типу у рядковий та зробити простішу конкатенацію.

Перепишемо цю програму так, щоб вона проводила складання двох чисел, які були передані їй через командний рядок. Отримати аргументи програми з командного рядка можливо через змінну *args* (параметр функції *main*).

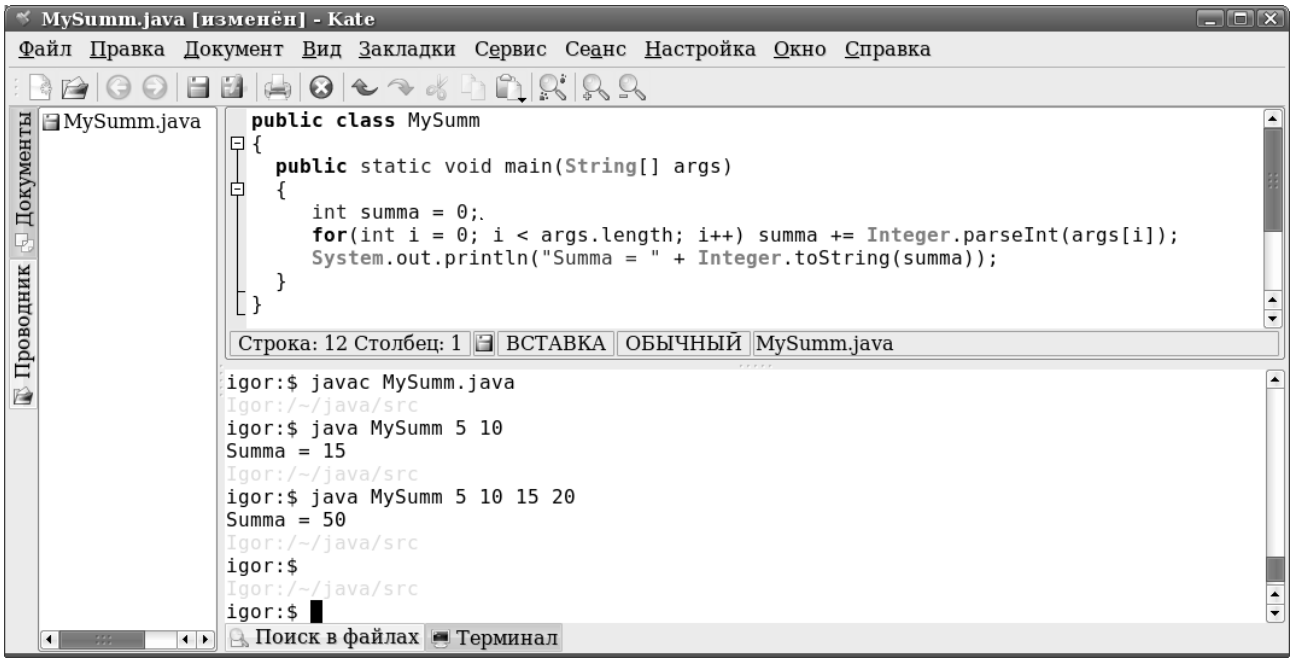

Рис. 1.3. Програма складання чисел, які передані через командний рядок

Таким чином, будь-яка програма на Java являє собою один, або декілька класів. Клас починається зі службового слова *class*, за яким слідує ім'я класу, за яким в свою чергу йде тіло класу (class body).

Складне ім'я *System.out.println* означає, що у класі *System*, який входить до Java API, є змінна з ім'ям *out*. Вона є екземпляром одного з класів Java API (*PrintStream*), в якому є метод *println()* (тобто *System.out* – це об'єкт). При використані методу *println()*, після виводу тексту у вікно терміналу, курсор буду переведений на новий рядок. Якщо потрібно, щоб курсор залишався на поточному рядку, то необхідно застосовувати метод *print()*.

Будь-яка програма на Java, яка оформлена як програма (application), повинна містити метод *main*. Цей метод може бути як один, так і бути присутнім у декількох класах. Він записується як звичайний метод і може містити будь-який код, але він обов'язково повинен бути відкритим (*public*), статичним (*static*) та таким, що не повертає значення (*void*). Його аргументом обов'язково є масив рядків (*String[]*) *args*. Така особливість в оголошені метода *main* виникає тому, що він буде викликаний автоматично системою Java на самому початку виконання програми. Якщо класів з методом *main* декілька, то обов'язково потрібно вказувати при виклику інтерпретатору *java* ім'я цього класу.

Щоб дізнатися скільки аргументів отримала програма через командний рядок, потрібно скористатися властивістю *length* (рис. 1.3). Зверніть увагу на те, що перший аргумент командного рядка – це *args[0]*, а останній *args[args.length-1]*.

Якщо потрібно в код на Java вставити коментарі, то це здійснюється так, як і на мові C++:

```
// Це коментар в один рядок
/* Це коментар
   у декілька
   рядків */
```

Третій різновид коментарів для декількох рядків: /** */ – використовується для автоматичної генерації документації.

Зауваження: якщо при компіляції ви допустили багато помилок, або ви забажали зберегти помилки окремо для більш детального їх перегляду у редакторі, то для цього можна використати наступну команду:

javac MySumm.java 2> errors.txt

Типи даних

Мова Java є строго типізованою. Це означає, що тип кожної змінної повинен бути оголошений. На мові Java є вісім основних, або простих типів (primitive types) даних. Чотири з них являють собою цілі числа, два – дійсні числа з плаваючою крапкою, один – символи у форматі Unicode та останній – логічні значення.

Цілі типи даних використовуються для представлення як позитивних, та і негативних чисел, які не мають дробової частини. В таблиці 1.1 наведені цілі типи даних.

Таблиця 1.1

Цілі типи

Тип	*Необхідний об'єм пам'яті (байти)*	*Діапазон (включно)*
int	4	-2147483648 ÷ 2147483647
short	2	-32768 ÷ 32767
long	8	-9223372036854775808 ÷ 9223372036854775807
byte	1	-128 ÷ 127

На мові Java діапазони цілих типів на залежать від комп'ютера, на якому виконується програма. Це суттєво спрощує перенос програмного забезпечення з однієї платформи до іншої.

Цілі числа типу *long* мають суфікс L (наприклад, 3000000L). Шістнадцятирічні числа мають префікс 0x (наприклад, 0x1BFA). Восьмирічні числа мають префікс 0 (наприклад, 010 – це десятирічне 8).

Зауваження:

1. Не рекомендують використовувати восьмирічні числа, щоб уникнути непорозумінь.
2. На мові Java відсутні беззнакові типи даних.

Типи з плаваючою крапкою (табл. 1.2) представляють значення, які мають дробову частину.

Таблиця 1.2

Типи даних з плаваючою крапкою

Тип	*Необхідний об'єм пам'яті (байти)*	*Діапазон*
float	4	≈ ± 3,40282347E+38
double	8	≈ ± 1,7976931348623157E+308

Числа типу *float* мають суфікс F (наприклад, 3.402F). Числа з плаваючою крапкою, які не мають суфікса F (наприклад, 3.402), завжди розглядаються як числа типу *double*. Для їх представлення можна (але не обов'язково) використовувати суфікс D (наприклад, 3.402D).

Зауваження: числа з плаваючою крапкою не можна використовувати у фінансових обчисленнях, де помилки округлення неприпустимі. Наприклад, у результаті виконання команди *System.out.println(2.0 - 1.1)* буде виведено не 0.9, а 0.8999999999999999. Уникнути таких ситуацій можливо за допомогою класу *BigDecimal*. Приклад реалізації функції вирахування *subtract* наведений нижче. Для її реалізації залучений спеціальний клас *BigDecimal*, який знаходиться у пакеті *java.math* (підключається за допомогою директиви *import*).

```
import java.math.*;

public class Test
{
 public static void Println( double Value )
 {
        System.out.println( Value );
 }

 public static double subtract( double value1, double value2, int scale )
 {
        BigDecimal dValue1 = new BigDecimal( value1 );
        dValue1 = dValue1.subtract( new BigDecimal( value2 ) );
        dValue1 = dValue1.setScale( scale, BigDecimal.ROUND_HALF_UP );
        return dValue1.doubleValue( );
 }

 public static void main( String[] args )
 {
        Println( 2.0 - 1.1 );                  // Результат: 0.8999999999999999
        Println( subtract( 2.0, 1.1, 1 ) );  // Результат: 0.9
 }
}
```

Всі обчислення, які проводяться над числами з плаваючою крапкою, виконуються згідно стандарту IEEE 754. Зокрема, у мові Java існує три спеціальних значення з плаваючою крапкою:

1. Позитивна нескінченність.
2. Негативна нескінченність.
3. NaN (не число).

Вони використовуються для позначення переповнення та помилок. Наприклад, у *класі Double* існує метод *isNaN*, який призначений для перевірки на ситуацію NaN: if(Double.isNaN(x)) ...

Тип char використовується для роботи з символами. Для того, щоб правильно використовувати тип char, потрібно мати представлення про принципи кодування Unicode. У 1991 році була випущена специфікація Unicode 1.0, яка використовувала менше половини з можливих 65536 кодів. У Java споконвічно були прийняті 16-бітні символи Unicode. На даний час 16-бітового типу char недостатньо для опису всіх символів Unicode.

Щоб зрозуміти, як ця проблема вирішається на Java, починаючи з JDK 5.0, треба ввести декілька термінів. Будемо називати *кодовою точкою* (code point) значення, яке пов'язане з символом у схемі кодування. Згідно стандарту Unicode, кодові точки записуються у шістнадцятирічному форматі та

випереджаються символами U+. Наприклад, для латинської літери A кодова точка дорівнює U+0041.

В Unicode кодові точки об'єднуються в 17 *кодових площинах* (code plane). Перша кодова площина, яка має назву основної багатомовної площини (basic multilingual plane), складається з "класичних" символів Unicode з кодовими точками від U+0000 до U+FFFF. Шістнадцять додаткових площин з кодовими точками від U+10000 до U+10FFFF містять *додаткові символи* (supplementary character).

Кодування UTF-16 – це спосіб представлення всіх кодів Unicode послідовністю змінної довжини. Символи з основної багатомовної площини представлені 16-бітовими значеннями, які мають назву *кодових одиниць* (code unit). Додаткові символи позначаються послідовними парами кодових одиниць. Кожне із значень пари попадає на 2048-байтову область основної багатомовної площини, яка має назву область підстановки (surrogates area); від U+D800 до U+DBFF для першої кодової одиниці та від U+DC00 до U+DFFF для другої кодової одиниці. Такий підхід дозволяє відразу визначити відповідає значення коду конкретного символу або є частиною коду додаткового символу. Наприклад, математичному коду символів, які позначають множину цілих чисел, відповідає кодова точка U+1D56B та дві кодових одиниці, U+D835 та U+DD6B.

У Java тип char описує кодову одиницю UTF-16.

Кодові одиниці Unicode можна виражати у вигляді шістнадцятирічних чисел в діапазоні від \u0000 до \uFFFF. Наприклад, значення \u2122 відповідає символу торгової марки (™), а \u03C0 – грецькій букві π.

Крім префіксу \u, який випереджає кодову одиницю Unicode, існує також декілька спеціальних символьних послідовностей (табл. 1.3).

Таблиця 1.3

Спеціальні символи

Спеціальний символ	*Опис*	*Значення Unicode*
\b	Повернення на одну позицію	\u0008
\t	Табуляція	\u0009
\n	Перехід на новий рядок	\u000a
\r	Повернення каретки	\u000d
\"	Подвійні лапки	\u0022
\'	Одинарні лапки	\u0027
\\	Зворотна коса риса	\u005c

Для логічних значень типу *boolean* передбачені два значення: *false* та *true*. Вони відповідають результатам обчислення логічних виразів.

Зауваження: перетворення значень логічних змінних у цілі та навпаки неможливо.

На мові Java оголошення змінної можна розміщувати в будь-якому місті коду у межах блоку. Але рекомендується оголошувати змінну як можна ближче до точки коду, де вона буде використана.

Слід зазначити, що на Java оголошення та визначення змінних не розрізняються.

Константи

Для позначення констант використовується ключове слово *final*, наприклад:

```
public class MyPage
{
  public static void main( String[] args )
  {
     final double CM_PER_INCH = 2.54;
     double PaperWidth = 8.267;
     double PaperHeight = 11.693;
     System.out.println( "Розмір сторінки в сантиметрах: " + PaperWidth * CM_PER_INCH +
                         " на " + PaperHeight * CM_PER_INCH );

     // Для версії JDK 5.0 і вище!
     System.out.printf( "Розмір сторінки в сантиметрах: %.1f на %.1f\n",
                        PaperWidth * CM_PER_INCH, PaperHeight * CM_PER_INCH );
  }
}
```

Результат:

```
Розмір сторінки в сантиметрах: 20.998179999999998 на 29.700219999999998
Розмір сторінки в сантиметрах: 21,0 на 29,7
```

На прикладі продемонстрований інший підхід виводу результату, який знайомий з мови C/C++. Для оформлення формату виводу використовується специфікатор %.1f.

Часто виникає необхідність у константах, які доступні декільком методам в середині одного класу. Зазвичай їх називають *константами класу* (class constant). Такі константи оголошуються за допомогою ключових слів *static final*. Якщо така константа оголошується з модифікатором *public*, то вона є доступною з інших класів. Наприклад (для версії JDK 5.0 і вище):

```
public class MyPage
{
  public static void PrintPageSize( double PaperWidth, double PaperHeight )
  {
          System.out.printf( "Розмір сторінки в сантиметрах: %.1f на %.1f\n",
                             PaperWidth * CM_PER_INCH, PaperHeight * CM_PER_INCH);
  }
  public static void main( String[] args )
  {
     PrintPageSize( 8.267, 11.693 );
  }

  public static final double CM_PER_INCH = 2.54;
}
```

Зауваження: ключове слово *const* на даний час у Java не використовується, але воно є зарезервованим.

Математичні функції та константи

Клас *Math* (не плутати з пакетом *java.math*) містить набір математичних статичних методів (статичні методи належать не об'єктам класів, а самім класам), які часто є необхідними при рішенні практичних задач. Це наступні методи: Math.pow( ), Math.sqrt( ), Math.sin( ), Math.cos( ), Math.tan( ), Math.atan( ), Math.atan2( ), Math.exp( ), Math.log( ), Math.round( ), Math.random( ) та інші. У класі є дві константи – Math.PI та Math.E.

Починаючи з JDK 5.0, при виклику методів для математичних обчислень клас *Math* можна не вказувати, якщо включити замість цього на початку файлу з кодом наступний вираз:

import static java.lang.Math.*;

Наприклад:

```
import static java.lang.Math.*;

public class TestMath
{
  public static void main( String[] args )
  {
     System.out.println( sqrt(16) );
  }
}
```

Для підвищення продуктивності функції з класу *Math* використовують програми з апаратного модуля, який призначений для обчислень з плаваючою крапкою. Якщо є несуттєвим швидкість роботи, а більш важливо отримати результати, що є передбачувані, то слід використовувати клас *StrictMath*. Він реалізує алгоритми з бібліотеки *fdlibm* та гарантує ідентичність результатів на всіх платформах.

Рядки

Рядок Java – це послідовність символів Unicode. Наприклад, рядок “Java\u2122” складається з п'яти символів: J, a, v, a, ™. В мові не має вбудованого типу для рядків. Замість цього стандартна бібліотека мови містить клас *String*. Кожний рядок, який укладений у подвійні лапки, являє собою екземпляр класу *String*. Наприклад:

```
String str1 = ""; // Порожній рядок
String str2 = "Hello";
```

Щоб встановити довжину рядка, використовують метод length( ):

```
public class TestString
{
  public static void main( String[] args )
  {
        String str = "Hello";
```

```
        int n = str.length();
        System.out.println( "Length = " + n );
  }
}
```

Зауваження: метод charAt( ) повертає символ (char) з вказаної позиції рядка. З версії JDK 5.0, для отримання більш коректних результатів, замість методу charAt( ) рекомендовано використовувати метод codePointAt( ).

За допомогою методу substring( ) класу *String* можна отримати підрядок поточного рядку (перший параметр – позиція з якої починається підрядок, другий параметр – до якої позиції підрядок). Наприклад:

```
String str1 = "Hello Igor!";
String str2 = str1.substring( 0, 5 );  // Вертає підрядок "Hello"
String str3 = str1.substring( 4, 8 ); //  Вертає підрядок "o Ig"
```

В класі *String* відсутні методи, які дозволяють змінювати символи в існуючих рядках! Якщо, наприклад, треба рядок "Hello" змінити на "H2O!o" то це неможливо. В документації для опису об'єктів *String* використовується термін *незмінні* (immutable). З цього випливає, що можливо тільки створювати нові рядки, якщо вони нам потрібні. Однак незмінні рядки мають велику перевагу – компілятор може робити рядки, що спільно використовувані.

Якщо потрібно створити рядок з окремих символів (наприклад, які поступають з клавіатури або файлу), то треба використовувати клас *StringBuffer*.

За допомогою знаку + можна робити конкатенацію рядків. При конкатенації рядка зі значенням, яке відрізняється від строкового, це значення перетворюється у рядок. Приклади:

```
String str1 = "Hello";
String str2 = "Igor";
String str3 = str1 + " " + str2 + "!";
String str4 = "Будинок ";
int n = 15;
String str5 = str4 + n;
System.out.println( str3 );  // Результат: "Hello Igor!"
System.out.println( str5 ); // Результат: "Будинок 15"
```

Для перевірки збігу рядків використовують методи equals( ) або equalsIgnoreCase( ), які вертають *true* або *false*. Наприклад:

```
String str1 = "Hello";
if( str1.equals( "hello" ) ) System.out.print( "OK 1  " );
else System.out.print( "ERROR 1  " );
if( str1.equalsIgnoreCase( "hello" ) ) System.out.println( "OK 2" );
else System.out.println( "ERROR 2" );
```

Результат: ERROR 1 OK 2

Можлива також наступна запис, яка не викликає помилок:

```
if( "hello".equals( str1 ) ) System.out.println( "OK 1" );
else System.out.println( "ERROR 1" );
```

Є корисними наступні методи класу *String*, назви яких не потребують коментарів: toLowerCase( ), toUpperCase( ), trim( ), compareTo( ) та ін.

Взагалі клас *String* містить більше 50 методів.

Уведення даних

До появи JDK 5.0 були відсутні зручні засоби читання інформації з клавіатури. Тепер ці проблеми вирішені.

Для того, щоб організувати читання інформації з консолі, необхідно створити об'єкт *Scanner* та зв'язати його зі стандартним вхідним потоком *System.in*. Після цього є можливим використання різних методів читання даних з клавіатури. Наприклад, метод nextLine( ) забезпечує читання текстового рядку, який може містити пробіли; метод next( ) забезпечує читання тільки одного слова; метод nextInt( ) дозволяє зчитувати ціле значення, а метод nextDouble( ) – дійсне значення.

Слід зазначити, що клас *Scanner* належить до пакету *java.util*.

Приклад:

```
import java.util.*;

public class TestInput
{
  public static void main( String[] args )
  {
        Scanner in = new Scanner( System.in );

        System.out.print( "Ваше ім'я?  " );
        String name = in.nextLine( );

        System.out.print( "Скількі Вам років?  " );
        int age = in.nextInt( );

        System.out.println( );
        System.out.print( "Добрибень " + name + ". " );
        System.out.println( "Вам тільки " + age + " роки (років) і це прекрасно!" );
  }
}
```

Для створення форматованого рядку без виводу можна скористатися статичним методом String.format( ):

```
String str = String.format( "Вам тільки %d роки (років) і це прекрасно!", age );
```

Масиви

Масив – це структура даних, яка зберігає величини однакового типу. Доступ до окремого елементу масиву здійснюється за допомогою цілого індексу. Приклад оголошення одномірного масиву на Java:

```
int[ ] a;
```

Однак цей оператор лише оголошує змінну *a*, але не ініціалізує її (не виділяє під масив пам'ять). Для створення масиву потрібно задіяти оператор *new*.

```
int[ ] a = new int[100];
```

Якщо кількість елементів в одномірному масиві невідома, то використовують вираз: *ім'я_масиву.length*:

```
for( int i = 0; i < a.length; i++ )  System.out.println( a[i] );
```

Зауваження: після створення масиву його розмір змінити не можливо. Якщо у ході виконання програми необхідно часто змінювати розмір масиву, то потрібно використовувати іншу структуру даних, наприклад, список.

В JDK 5.0 був реалізований новий цикл, який дозволив перебирати всі елементи масиву (а також будь-якого іншого набору даних), не використовуючи лічильник. Новий варіант циклу for наступний:

for(*змінна* : *набір_даних*) *вираз*

При обробці циклу змінній послідовно присвоюється кожний елемент набору даних, після чого виконується вираз (або блок):

```
for( int element : a)  System.out.println( element );
```

Приклад оголошення та ініціалізації масиву: int[ ] mas = { 1, 3, 5, 7, 11 };

За необхідністю одну змінну масиву можна скопіювати в іншу, але при цьому обидві змінні будуть посилатися на один і той же масив:

```
int[ ] b = a;
b[3] = 9;       // Тепер елемент a[3] також дорівнює 9
```

Для копіювання всіх елементів одного масиву в інший слід застосувати метод arraycopy( ) з класу *System*:

System.arraycopy(from, fromIndex, to, toIndex, count);

Зауваження: клас *Arrays* з пакету *java.util* містить ряд статичних методів, які є корисними при виконанні різних операцій з елементами одномірного масиву (наприклад, метод sort( )).

Для доступу до елементів багатовимірного масиву використовують декілька індексів. Приклад роботи з двовимірним масивом:

```
public class TestMatrix
{
  public static void main( String[ ] args )
  {
          int M = 5; // Кількість рядків
          int N = 3; // Кількість стовпців

          double[ ][ ] matrix = new double[M][N];

          for( int i = 0; i < matrix.length; i++ )
            for( int j = 0; j < matrix[i].length; j++ )
                    matrix[i][j] = 10;

          for( int i = 0; i < matrix.length; i++ )
          {
            for( int j = 0; j < matrix[i].length; j++ ) System.out.print( matrix[i][j] + " " );
            System.out.println( );
          }

          System.out.println( );

          for( double[ ] row : matrix )
          {
           for( double Value : row ) System.out.print( Value + " " );
           System.out.println( );
          }
  }
}
```

Приклад оголошення, ініціалізації та виводу двовимірного масиву:

```
int[ ][ ] magicSquare = { {16, 3, 2, 13},
                          {5, 10, 11, 8},
                          {9, 6, 7,12},
                          {4, 15, 14, 1} };

for( int[ ] row : magicSquare )
{
   for( int Value : row ) System.out.print( Value + "\t" );
   System.out.println( );
}
```

Практична частина

Вивчить основні особливості побудови простіших консольних програм на мові Java з застосуванням JDK 5.0 або 6.0, які були розглянуті у теоретичній частині лабораторної роботи.

Закріпіть знання виконавши запропонований викладачем варіант обчислення математичного виразу згідно таблиць 1.4 – 1.6.

Додаткову документацію по методам класів дивитися у каталозі: /opt/SDK/jdk/.

У звіті відобразити код програми з коментарями та поясненнями, а також висновками.

Таблиця 1.4

Варіанти завдань

Варіант завдання	*Варіант виразу*	*Варіант вхідних даних*	*Варіант завдання*	*Варіант виразу*	*Варіант вхідних даних*
1	1	1	11	1	2
2	2	2	12	2	1
3	3	1	13	3	2
4	4	2	14	4	1
5	5	1	15	5	2
6	6	2	16	6	1
7	7	1	17	7	2
8	8	2	18	8	1
9	9	1	19	9	2
10	10	2	20	10	1

Таблиця 1.5

Варіанти виразів

Варіант виразу	*Вирази*
1	*2*
1	$S=\dfrac{\sum_{i=1}^{5}\sin\lvert 1-\ln a_i\rvert}{\sum_{i=1}^{11}\sin^2 18\,a_i^3}$
2	$S=\dfrac{\sum_{i=1}^{11}\sin(1-3*\sin^3 a_i)}{3*\sum_{i=1}^{11}\cos^2 a_i^5}$
3	$S=\dfrac{\sum_{i=1}^{11}\cos\lvert 1-\ln^2 a_i\rvert}{\sum_{i=1}^{11}\sin^2 18\,a_i^{a_i}}$
4	$S=\dfrac{\ln(\sum_{i=1}^{11}(\lvert\sin a_i\rvert^{\sin a_i}+\lvert\cos a_i\rvert^{\cos a_i}))}{3*\sum_{i=1}^{11}\sin^2 a_i^{\sin a_i}}$
5	$S=\dfrac{\sin^2(\sum_{i=1}^{11}(\ln(\lvert 1-\cos^2 a_i\rvert)))}{\sum_{i=1}^{11}\cos(\lvert 1-\sin a_i\rvert)}$
6	$S=\sum_{i=1}^{11}\dfrac{1-2\cos^2 a_i}{\lvert\sin a_i\rvert*\lvert\cos a_i\rvert}$

Продовження таблиці 1.5

1	*2*
7	$S=\sum_{i=1}^{11}(1+\cos a_i+\cos\frac{a_i}{2})$
8	$S=\sum_{i=1}^{11}\frac{\cos 2a_i}{1-\sin 2a_i}$
9	$S=\sqrt{\frac{\sum_{i=1}^{11}(a_i-\tilde{a})^2}{10}}, \tilde{a}=\frac{1}{11}\sum_{i=1}^{11}a_i$
10	$S=\sum_{i=1}^{11}(1+\sin a_i+\sin\frac{a_i}{2})$

Таблиця 1.6

Вхідні дані

Варіант вхідних даних	a_i
1 (тип даних int)	5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55
2 (тип даних float)	0.3, 0.7, 0.9, 1.3, 1.7, 1.9, 2.3, 2.7, 2.9, 3.3, 3.7