РЯДКИ ТА МАСИВИ ЯК ПОСИЛАЛЬНІ ТИПИ ДАНИХ

Питання 2.3.

Клас String

- String представляє рядок як послідовність символів.
  - На відміну від C, рядок не закінчується символом ‘\0’.
  - Просто довжина рядка зберігається окремо.
- Типові приклади отримання String
  - String favLanguage = "Java";.
  - String favLanguage = new String("Java");.
- Після отримання рядка можна викликати багато доступних методів.
  - Наприклад, довжина рядка знаходиться за допомогою методу length();
  - favLanguage.length().

Конструктори та методи

Method	Description
`String(char[] data)`	Initializes this `String` object to the characters in the `data` array. Modifying `data` after initializing this `String` object has no effect on the object.
`String(String s)`	Initializes this `String` object to `s`'s string.
`char charAt(int index)`	Returns the character located at the zero-based `index` in this `String` object's string. Throws `java.lang.StringIndexOutOfBoundsException` when `index` is less than 0 or greater than or equal to the length of the string.
`String concat(String s)`	Returns a new `String` object containing this `String` object's string followed by the `s` argument's string.
`boolean endsWith(String suffix)`	Returns true when this `String` object's string ends with the characters in the `suffix` argument, when `suffix` is empty (contains no characters), or when `suffix` contains the same character sequence as this `String` object's string. This method performs a case-sensitive comparison (a is not equal to A, for example) and throws `NullPointerException` when `suffix` is `null`.
`boolean equals(Object object)`	Returns true when `object` is of type `String` and this argument's string contains the same characters (and in the same order) as this `String` object's string.
`boolean equalsIgnoreCase(String s)`	Returns true when `s` and this `String` object contain the same characters (ignoring case). This method returns false when the character sequences differ or when `null` is passed to `s`.

`int indexOf(int c)`	Returns the zero-based index of the first occurrence (from the start of the string to the end of the string) of the character represented by `c` in this `String` object's string. Returns -1 when this character is not present.
`int indexOf(String s)`	Returns the zero-based index of the first occurrence (from the start of the string to the end of the string) of `s`'s character sequence in this `String` object's string. Returns -1 when `s` is not present. Throws `NullPointerException` when `s` is `null`.
`String intern()`	Searches an internal table of `String` objects for an object whose string is equal to this `String` object's string. This `String` object's string is added to the table when not present. Returns the object contained in the table whose string is equal to this `String` object's string. The same `String` object is always returned for strings that are equal.
`int lastIndexOf(int c)`	Returns the zero-based index of the last occurrence (from the start of the string to the end of the string) of the character represented by `c` in this `String` object's string. Returns -1 when this character is not present.
`int lastIndexOf(String s)`	Returns the zero-based index of the last occurrence (from the start of the string to the end of the string) of `s`'s character sequence in this `String` object's string. Returns -1 when `s` is not present. Throws `NullPointerException` when `s` is `null`.
`int length()`	Returns the number of characters in this `String` object's string.

Method	Description
`String replace(char oldChar, char newChar)`	Returns a new `String` object whose string matches this `String` object's string except that all occurrences of `oldChar` have been replaced by `newChar`.
`String[] split(String expr)`	Splits this `String` object's string into an array of `String` objects using the *regular expression* (a string whose *pattern* [template] is used to search a string for substrings that match the pattern) specified by `expr` as the basis for the split. Throws `NullPointerException` when `expr` is `null` and `java.util.regex.PatternSyntaxException` when `expr`'s syntax is invalid.
`boolean startsWith(String prefix)`	Returns true when this `String` object's string starts with the characters in the `prefix` argument, when `prefix` is empty (contains no characters), or when `prefix` contains the same character sequence as this `String` object's string. This method performs a case-sensitive comparison (such as a is not equal to A), and throws `NullPointerException` when `prefix` is `null`.
`String substring(int start)`	Returns a new `String` object whose string contains this `String` object's characters beginning with the character located at `start`. Throws `StringIndexOutOfBoundsException` when `start` is negative or greater than the length of this `String` object's string.
`char[] toCharArray()`	Returns a character array that contains the characters in this `String` object's string.

`String toLowerCase()`	Returns a new `String` object whose string contains this `String` object's characters where uppercase letters have been converted to lowercase. This `String` object is returned when it contains no uppercase letters to convert.
`String toUpperCase()`	Returns a new `String` object whose string contains this `String` object's characters where lowercase letters have been converted to uppercase. This `String` object is returned when it contains no lowercase letters to convert.
`String trim()`	Returns a new `String` object that contains this `String` object's string with *whitespace characters* (characters whose Unicode values are 32 or less) removed from the start and end of the string, or this `String` object when there is no leading/trailing whitespace.

- Конструктор String(String s) не ініціалізує String-об'єкт рядковим літералом,як при new String("Java").
  - Замість цього він поводиться аналогічно до конструктора копіювання з C++, ініціалізуючи String-об'єкт вмістом іншого String-об'єкту.
  - Насправді, рядковий літерал є String-об'єктом.
  - Це можна довести для себе, виконавши System.out.println("abc".length()); та System.out.println("abc" instanceof String);.
  - Виклик першого методу поверне 3, а другого - true ("abc" is a String object).

Метод intern()

- Зберігає унікальну копію об’єкту String у внутрішній таблиці (internal table) String-об’єктів.
  - intern() дає можливість порівнювати рядки за їх посиланнями та == або !=.
  - Ці оператори – найшвидший спосіб порівняти рядки, що особливо цінне при сортуванні великої кількості рядків.

```
public class StringDemo
{
   public static void main(String[] args)
   {
      System.out.println("abc".length());
      System.out.println("abc" instanceof String);
      System.out.println("abc" == "a" + "bc");
      System.out.println("abc" == new String("abc"));
      System.out.println("abc" == new String("abc").intern());
   }
}
```

За умовчанням String-об’єкти, записані літералами "abc" та константними рядковими виразами ("a" + "bc"), є interned.

Проте String-об’єкти, створені за допомогою конструкторів String, не є interned

```
3
true
true
false
true
```

Обережно!

- Будьте обережні з порівнянням рядків (порівнюються посилання).
  - Можна легко внести баг, коли один з рядків, що порівнюються, не був interned.
  - При сумнівах використовуйте методи equals() або equalsIgnoreCase().
  - Наприклад, "abc".equals(new String("abc")) та "abc".equalsIgnoreCase(new String("ABC")) повертають true.

Методи CharAt() та split()

- CharAt() корисний для виокремлення символів з рядка.

```
public class StringDemo
{
   public static void main(String[] args)
   {
      String s = "abc";
      for (int i = 0; i < s.length(); i++)
         System.out.println(s.charAt(i));
   }
}
```

- split() розбиває список значень, розділених комами в рядку, на масив String-об'єктів.
  - Використовує регулярні вирази, що ідентифікують послідовності символів, між якими відбувається розбиття (розглянемо пізніше).

> **Note** StringIndexOutOfBoundsException and ArrayIndexOutOfBoundsException are sibling classes that share a common java.lang.IndexOutOfBoundsException superclass.

Класи StringBuffer та StringBuilder

- Об'єкти String ***незмінювані (immutable)***. Ви не можете змінити рядок об'єкта String.
  - Методи String насправді повертають новий об'єкт String з модифікованим контентом.
  - Оскільки часто це марнотратство, Java постачає класи StringBuffer та StringBuilder.
  - StringBuffer та StringBuilder ідентичні, за винятком того, що StringBuilder пропонує кращу продуктивність, але не може використовуватись у багатопоточних задачах без явної синхронізації.
- StringBuffer та StringBuilder забезпечують внутрішній символьний масив для ефективної побудови рядків.
  - Після створення об'єкту StringBuffer/StringBuilder Ви можете викликати різні методи для словарних операцій (додавання, видалення, вставка) з масивом.
  - Потім викликають метод toString() для конвертації вмісту масиву в об'єкт типу String та повернення цього об'єкта.

Method	Description
`StringBuffer()`	Initializes this `StringBuffer` object to an empty array with an initial capacity of 16 characters.
`StringBuffer(int capacity)`	Initializes this `StringBuffer` object to an empty array with an initial capacity of `capacity` characters. This constructor throws `java.lang.NegativeArraySizeException` when `capacity` is negative.
`StringBuffer(String s)`	Initializes this `StringBuffer` object to an array containing `s`'s characters. This object's initial capacity is 16 plus the length of `s`. This constructor throws `NullPointerException` when `s` is `null`.
`StringBuffer append(boolean b)`	Appends "`true`" to this `StringBuffer` object's array when `b` is `true` and "`false`" to the array when `b` is `false`, and returns this `StringBuffer` object.
`StringBuffer append(char ch)`	Appends `ch`'s character to this `StringBuffer` object's array, and returns this `StringBuffer` object.
`StringBuffer append(char[] chars)`	Appends the characters in the `chars` array to this `StringBuffer` object's array, and returns this `StringBuffer` object. This method throws `NullPointerException` when `chars` is `null`.
`StringBuffer append(double d)`	Appends the string representation of `d`'s double precision floating-point value to this `StringBuffer` object's array, and returns this `StringBuffer` object.
`StringBuffer append(float f)`	Appends the string representation of `f`'s floating-point value to this `StringBuffer` object's array, and returns this `StringBuffer` object.
`StringBuffer append(int i)`	Appends the string representation of `i`'s integer value to this `StringBuffer` object's array, and returns this `StringBuffer` object.
`StringBuffer append(long l)`	Appends the string representation of `l`'s long integer value to this `StringBuffer` object's array, and returns this `StringBuffer` object.
`StringBuffer append(Object obj)`	Calls `obj`'s `toString()` method and appends the returned string's characters to this `StringBuffer` object's array. Appends "`null`" to the array when `null` is passed to `obj`. Returns this `StringBuffer` object.

`StringBuffer append(String s)`	Appends s's string to this `StringBuffer` object's array. Appends "`null`" to the array when `null` is passed to `s`. Returns this `StringBuffer` object.
`int capacity()`	Returns the current capacity of this `StringBuffer` object's array.
`char charAt(int index)`	Returns the character located at `index` in this `StringBuffer` object's array. This method throws `StringIndexOutOfBoundsException` when `index` is negative or greater than or equal to this `StringBuffer` object's length.
`void ensureCapacity(int min)`	Ensures that this `StringBuffer` object's capacity is at least that specified by `min`. If the current capacity is less than `min`, a new internal array is created with greater capacity. The new capacity is set to the larger of `min` and the current capacity multiplied by 2, with 2 added to the result. No action is taken when `min` is negative or zero.
`int length()`	Returns the number of characters stored in this `StringBuffer` object's array.
`StringBuffer reverse()`	Returns this `StringBuffer` object with its array contents reversed.
`void setCharAt(int index, char ch)`	Replaces the character at `index` with `ch`. This method throws `StringIndexOutOfBoundsException` when `index` is negative or greater than or equal to the length of this `StringBuffer` object's array.
`void setLength(int length)`	Sets the length of this `StringBuffer` object's array to `length`. If the `length` argument is less than the current length, the array's contents are truncated. If the `length` argument is greater than or equal to the current length, sufficient null characters (`'\u0000'`) are appended to the array. This method throws `StringIndexOutOfBoundsException` when `length` is negative.
`String substring(int start)`	Returns a new `String` object that contains all characters in this `StringBuffer` object's array starting with the character located at `start`. This method throws `StringIndexOutOfBoundsException` when `start` is less than 0 or greater than or equal to the length of this `StringBuffer` object's array.
`String toString()`	Returns a new `String` object whose string equals the contents of this `StringBuffer` object's array.

Демонстрація StringBuffer

```
public class StringBufferDemo
{
   public static void main(String[] args)
   {
      StringBuffer sb = new StringBuffer("Hello,");
      sb.append(' ');
      sb.append("world. ");
      sb.append(args.length);
      sb.append(" argument(s) have been passed to this method.");
      String s = sb.toString();
      System.out.println(s);
   }
}
```

- Після створення об'єкту StringBuffer, він ініціалізується внутрішнім (interned) контентом "Hello,";
  - Далі в буфер додаються символ, інший об'єкт типу String, ціле число та ще один об'єкт типу String.
  - Потім вміст StringBuffer конвертується в об'єкт String та виводиться на екран.
  - Можна запустити з командного рядка з деякими параметрами. Тоді число зміниться.

```
Hello, world. 0 argument(s) have been passed to this method.
```

Розглянемо сценарій

- Пишемо код, який перетворить ціле число в рядок.
  - В якості частини форматувальника (formatter) потрібно дописувати попереду конкретну кількість незначущих пробілів.
  - Вирішили використовувати наступний код, щоб побудувати рядок з трьома пробілами:

```
int numLeadingSpaces = 3; // default value
String spacesPrefix = "";
for (int j = 0; j < numLeadingSpaces; j++)
   spacesPrefix += "0";
```

  - Цикл неефективний, оскільки кожна ітерація створює свої об'єкти StringBuilder та String. Компілятор перепише код так:

```
int numLeadingSpaces = 3; // default value
String spacesPrefix = "";
for (int j = 0; j < numLeadingSpaces; j++)
   spacesPrefix = new StringBuilder().append(spacesPrefix).append("0").toString();
```

Більш продуктивний підхід для попереднього циклу

- Створення об’єкту типу StringBuilder/StringBuffer перед входом у цикл, виклик у циклі відповідного методу append(), а після циклу – методу toString().

```
int numLeadingSpaces = 3; // default value
StringBuilder sb = new StringBuilder();
for (int j = 0; j < numLeadingSpaces; j++)
   sb.append('0');
String spacesPrefix = sb.toString();
```

- Уникайте оператора конкатенації рядків у довгих циклах, оскільки він створює багато зайвих об’єктів String та StringBuilder!

Клас System

- The java.lang.System utility class оголошує методи класу, які забезпечують доступ
  - до поточного часу (у мс),
  - system property values,
  - environment variable values, and other kinds of system information.
- Також у ньому містяться методи класу, які підтримують системні задачі з копіювання одного масиву в інший, надсилання запиту на збір сміття та ін.

Метод	Опис
void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)	Копіює задану значенням length кількість елементів з масиву src, починаючи зі зсуву srcPos відносно початку, в масив dest зі зсувом destPos відносно його початку. Викидає *NPE*, коли src або dest рівні null, *IndexOutOfBoundsException* – якщо копіювання спричиняє вихід за межі масиву, *java.lang.ArrayStoreException* – коли елемент в src неможливо зберегти в dest через несумісність типів.

Методи класу System

Метод	Опис
long currentTimeMillis()	Повертає поточний системний час у мілісекундах від 01.01.1970, 00:00:00 UTC
void gc()	Інформує віртуальну машину, що зараз добре запустити збирання сміття. Це лише підказка для неї, гарантії запуску немає.
String getEnv(String name)	Повертає значення змінної середовища за її назвою name.
String getProperty(String name)	Повертає значення системної властивості (платформозалежного атрибуту, наприклад, версія) за її назвою name. Повертає null, коли такої властивості не існує. На Android корисні системні властивості: file.separator, java.class.path, java.home, java.io.tmpdir, java.library.path, line.separator, os.arch, os.name, path.separator, user.dir.
void runFinalization()	Інформує віртуальну машину, що добре було б виконати фіналізацію всіх (outstanding) об'єктів, які затримались у пам'яті. Це лише підказка, гарантії фіналізації немає.

Демонстрація

```
public class SystemDemo
{
   public static void main(String[] args)
   {
      int[] grades = { 86, 92, 78, 65, 52, 43, 72, 98, 81 };
      int[] gradesBackup = new int[grades.length];
      System.arraycopy(grades, 0, gradesBackup, 0, grades.length);
      for (int i = 0; i < gradesBackup.length; i++)
         System.out.println(gradesBackup[i]);
      System.out.println("Current time: " + System.currentTimeMillis());
      String[] propNames =
      {
         "file.separator",
         "java.class.path",
         "java.home",
         "java.io.tmpdir",
         "java.library.path",
         "line.separator",
         "os.arch",
         "os.name",
         "path.separator",
         "user.dir"
      };
      for (int i = 0; i < propNames.length; i++)
         System.out.println(propNames[i] + ": " +
                            System.getProperty(propNames[i]));
   }
}
```

- Метод arraycopy() – найшвидший спосіб для портативного копіювання одного масиву в інший.
  - Також, коли Ви пишете клас, чиї методи повертають посилання на внутрішній масив, краще використовувати arraycopy() для створення копії масиву, а потім повертати посилання на копію.
- Таким чином Ви не даєте клієнтам напряму маніпулювати (напартачити) внутрішнім масивом.

currentTimeMillis() повертає час у мс. Це значення нечитабельне, частіше використовують клас java.util.Date

Вивід програми

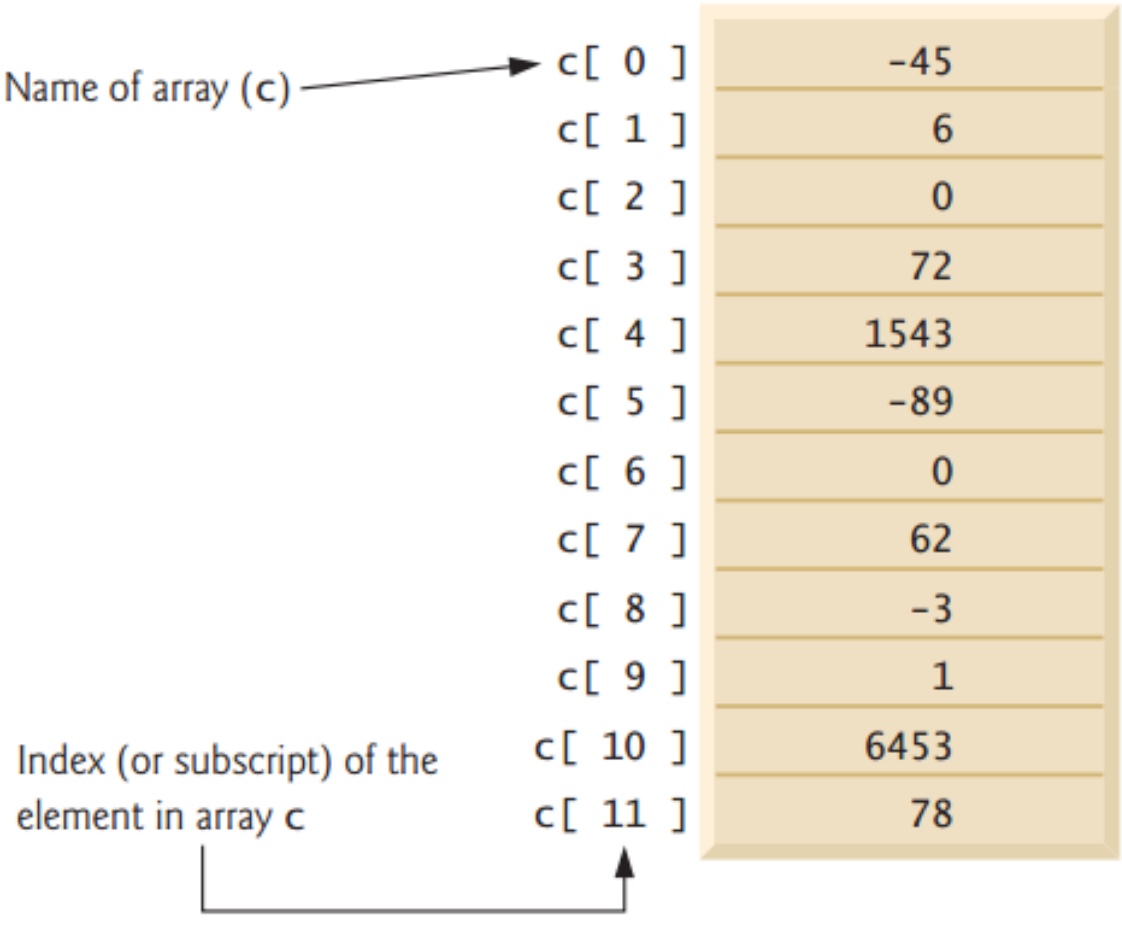
```
86
92
78
65
52
43
72
98
81
Current time: 1353115138889
file.separator: \
java.class.path: .;C:\Program Files (x86)\QuickTime\QTSystem\QTJava.zip
java.home: C:\Program Files\Java\jre7
java.io.tmpdir: C:\Users\Owner\AppData\Local\Temp\
java.library.path: C:\Windows\system32;C:\Windows\Sun\Java\bin;C:\Windows\system32;C:\Windows;c:\
Program Files (x86)\AMD APP\bin\x86_64;c:\Program Files (x86)\AMD APP\bin\x86;C:\Program Files\
Common Files\Microsoft Shared\Windows Live;C:\Program Files (x86)\Common Files\Microsoft Shared\
Windows Live;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\
WindowsPowerShell\v1.0\;C:\Program Files (x86)\ATI Technologies\ATI.ACE\Core-Static;C:\Program Files
(x86)\Windows Live\Shared;C:\Program Files\java\jdk1.7.0_06\bin;C:\Program Files (x86)\Borland\
BCC55\bin;C:\android;C:\android\tools;C:\android\platform-tools;C:\Program Files (x86)\apache-
ant-1.8.2\bin;C:\Program Files (x86)\QuickTime\QTSystem\;.
line.separator:

os.arch: amd64
os.name: Windows 7
path.separator: ;
user.dir: C:\prj\dev\ljfad2\ch08\code\SystemDemo
```

Масиви

- Найпростіші структури даних, які складаються з пов’язаних даних одного типу.
  - Зручні для обробки груп значень
- Незважаючи на широке використання, мають обмежені можливості.
  - Розмір масиву потрібно задавати наперед.
  - У процесі виконання потрібно буде створювати додаткові масиви.
- Java API пропонує також використовувати колекції – набір базових структур даних (стеки, множини, списки, карти та ін.), що дають більші можливості.
  - Java 8 дає можливість використання альтернативної реалізації за допомогою лямбда-виразів та потоків даних (streams), які краще паралеляться.

Представлення масиву

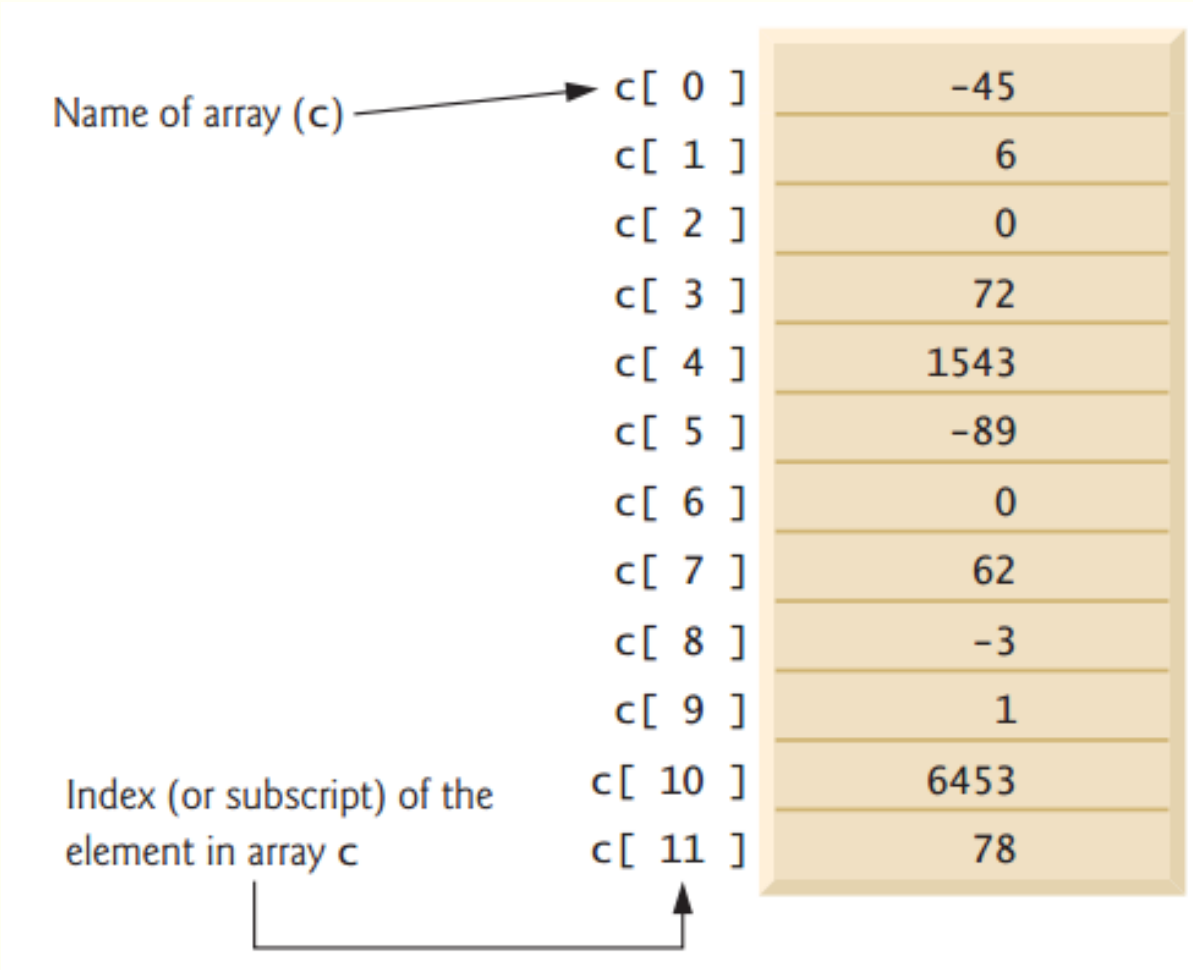
- ***Масив*** – група змінних (елементів), що містять значення одного типу.
  - Елементами масиву можуть бути як примітивні типи, так і посилкові типи, включаючи і самі масиви.
  - Для звернення до елементу масиву використовується його індекс – номер позиції в масиві.
  - Індекс знаходиться в [...].
  - Індекс – невід'ємне ціле число
    - Можливі типи: int, byte, short, char
    - Тип long використовувати не можна



Докладніший огляд

- Назва масиву – c
- Кожен масив знає свою довжину, яка міститься у змінній об'єкта (instance variable) c.length.
  - public final length – немає можливості змінити
- В масиві 12 елементів, доступ до яких отримується за допомогою запису c[0],...,c[11].
- Приклади дій з масивом

```
sum = c[0] + c[1] + c[2];
 x = c[6] / 2;
```



Оголошення та створення масивів

- Створюються, як інші об'єкти, за допомогою ключового слова new

```
int[] c = new int[12];
```

  - Відповідний вираз повертає посилання (*reference*), яке зберігається в змінній-масиві c.
  - При створенні масиву всім елементам автоматично присвоюється значення за замовчуванням.
    - Для числових примітивних типів – 0;
    - Для boolean – false;
    - Для посилкових типів – null.

```
int[] c; // declare the array variable
c = new int[12]; // create the array; assign to array variable
```

Загальний синтаксис створення масивів

***int[12] c;* - синтаксична помилка!**

```
String[] b = new String[100]; // create array b
String[] x = new String[27]; // create array x
```

  - Рекомендується створювати кожен масив у окремому рядку.

Приклади використання масивів. Ініціалізація

```
1  // Fig. 7.2: InitArray.java
2  // Initializing the elements of an array to default values of zero.
3
4  public class InitArray
5  {
6     public static void main(String[] args)
7     {
8        // declare variable array and initialize it with an array object
9        int[] array = new int[10]; // create the array object
10
11       System.out.printf("%s%8s%n", "Index", "Value"); // column headings
12
13       // output each array element's value
14       for (int counter = 0; counter < array.length; counter++)
15          System.out.printf("%5d%8d%n", counter, array[counter]);
16    }
17 } // end class InitArray
```

```
Index   Value
    0       0
    1       0
    2       0
    3       0
    4       0
    5       0
    6       0
    7       0
    8       0
    9       0
```

Приклади використання масивів. Ініціалізатори

- ***Ініціалізатор масиву*** *(array initializer)* – список виразів, розділених комою, у фігурних дужках.

```
1  // Fig. 7.3: InitArray.java
2  // Initializing the elements of an array with an array initializer.
3
4  public class InitArray
5  {
6     public static void main(String[] args)
7     {
8        // initializer list specifies the initial value for each element
9        int[] array = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
10
11       System.out.printf("%s%8s%n", "Index", "Value"); // column headings
12
13       // output each array element's value
14       for (int counter = 0; counter < array.length; counter++)
15          System.out.printf("%5d%8d%n", counter, array[counter]);
16
17 } // end class InitArray
```

```
Index   Value
    0      32
    1      27
    2      64
    3      18
    4      95
    5      14
    6      90
    7      70
    8      60
    9      37
```

Приклади використання масивів. Обчислення значень

```
1  // Fig. 7.4: InitArray.java
2  // Calculating the values to be placed into the elements of an array.
3
4  public class InitArray
5  {
6     public static void main(String[] args)
7     {
8        final int ARRAY_LENGTH = 10; // declare constant
9        int[] array = new int[ARRAY_LENGTH]; // create array
10
11       // calculate value for each array element
12       for (int counter = 0; counter < array.length; counter++)
13          array[counter] = 2 + 2 * counter;
14
15       System.out.printf("%s%8s%n", "Index", "Value"); // column headings
16
17       // output each array element's value
18       for (int counter = 0; counter < array.length; counter++)
19          System.out.printf("%5d%8d%n", counter, array[counter]);
20    }
21 } // end class InitArray
```

```
Index   Value
    0       2
    1       4
    2       6
    3       8
    4      10
    5      12
    6      14
    7      16
    8      18
    9      20
```

Приклади використання масивів. Додавання елементів

```
1  // Fig. 7.5: SumArray.java
2  // Computing the sum of the elements of an array.
3
4  public class SumArray
5  {
6     public static void main(String[] args)
7     {
8        int[] array = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
9        int total = 0;
10
11       // add each element's value to total
12       for (int counter = 0; counter < array.length; counter++)
13          total += array[counter];
14
15       System.out.printf("Total of array elements: %d%n", total);
16    }
17 } // end class SumArray
```

```
Total of array elements: 849
```

Enhanced оператор циклу for

- Проходить по елементах масиву БЕЗ використання лічильника, уникаючи проблеми виходу за межі масиву.

```
1  // Fig. 7.12: EnhancedForTest.java
2  // Using the enhanced for statement to total integers in an array.
3
4  public class EnhancedForTest
5  {
6     public static void main(String[] args)
7     {
8        int[] array = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
9        int total = 0;
10
11       // add each element's value to total
12       for (int number : array)
13          total += number;
14
15       System.out.printf("Total of array elements: %d%n", total);
16    }
17 } // end class EnhancedForTest
```

```
Total of array elements: 849
```

Може ЛИШЕ ОТРИМУВАТИ значення, проте не дає їх змінювати.

```
for (parameter : arrayName)
   statement
```

Передача масивів у методи

- Нехай маємо масив

```
double[] hourlyTemperatures = new double[24];
```

- Для передачі масиву просто записується його назва.
  - Довжину передавати не потрібно, оскільки масиву вона відома.

```
modifyArray(hourlyTemperatures);
```

- У методі список його параметрів має задавати параметр масиву (*array parameter*). Наприклад,

```
void modifyArray(double[] b)
```

- Виклик методу передає посилання на масив hourlyTemperature, тому при використанні змінної b викликаний метод звертається до того ж об'єкта-масиву, що й hourlyTemperatures при передачі.

Передача масивів у методи

- Загалом існує 2 підходи до передачі аргументів методу
  - ***Передача за значенням*** *(pass-by-value, call-by-value)*
    - Якщо передається елемент масиву примітивного типу, при виклику методу передається КОПІЯ ЗНАЧЕННЯ елементу (скалярна величина).
    - Метод працює лише з копією значення.
    - Зміни в переданій копії не впливають на оригінальну змінну.
  - ***Передача за посиланням*** *(pass-by-reference, call-by-reference)*
    - Викликаний метод може напряму отримати доступ та виконувати зміни в даних аргумента
    - Покращується продуктивність, оскільки не потрібно виконувати копіювання великих об'ємів даних.
    - Коли аргументом методу є цілий масив або окремий його елемент посилкового типу, викликаний метод отримує КОПІЮ ПОСИЛАННЯ.

Передача масивів у методи

- Java не дозволяє вибирати тип передачі – всі аргументи передаються за значенням!
  - У виклик методу можуть передаватись або копії примітивних значень, або копії посилань на об'єкти.
    - Власне об'єкти передаватись не можуть.
  - Хоча посилання на об'єкт передається за значенням, метод може взаємодіяти з об'єктом, на який посилаються, викликаючи його публічні методи за допомогою копії посилання на об'єкт.
    - Копія посилання (аргумент методу) посилається на той же об'єкт у пам'яті, тому викликаний метод може напряму працювати з об'єктом.
    - Для великих масивів передача посилання – суттєве підвищення продуктивності коду

Передача масивів та окремих значень

```
1  // Fig. 7.13: PassArray.java
2  // Passing arrays and individual array elements to methods.
3
4  public class PassArray
5  {
6     // main creates array and calls modifyArray and modifyElement
7     public static void main(String[] args)
8     {
9        int[] array = { 1, 2, 3, 4, 5 };
10
11       System.out.printf(
12          "Effects of passing reference to entire array:%n" +
13          "The values of the original array are:%n");
14
15       // output original array elements
16       for (int value : array)
17          System.out.printf("   %d", value);
18
19       modifyArray(array); // pass array reference
20       System.out.printf("%n%nThe values of the modified array are:%n");
21
22       // output modified array elements
23       for (int value : array)
24          System.out.printf("   %d", value);
25
26       System.out.printf(
27          "%n%nEffects of passing array element value:%n" +
28          "array[3] before modifyElement: %d%n", array[3]);
29
30       modifyElement(array[3]); // attempt to modify array[3]
31       System.out.printf(
32          "array[3] after modifyElement: %d%n", array[3]);
33    }
```

Рядок 30 передає array[3] в метод modifyElement()

Передача масивів та окремих значень

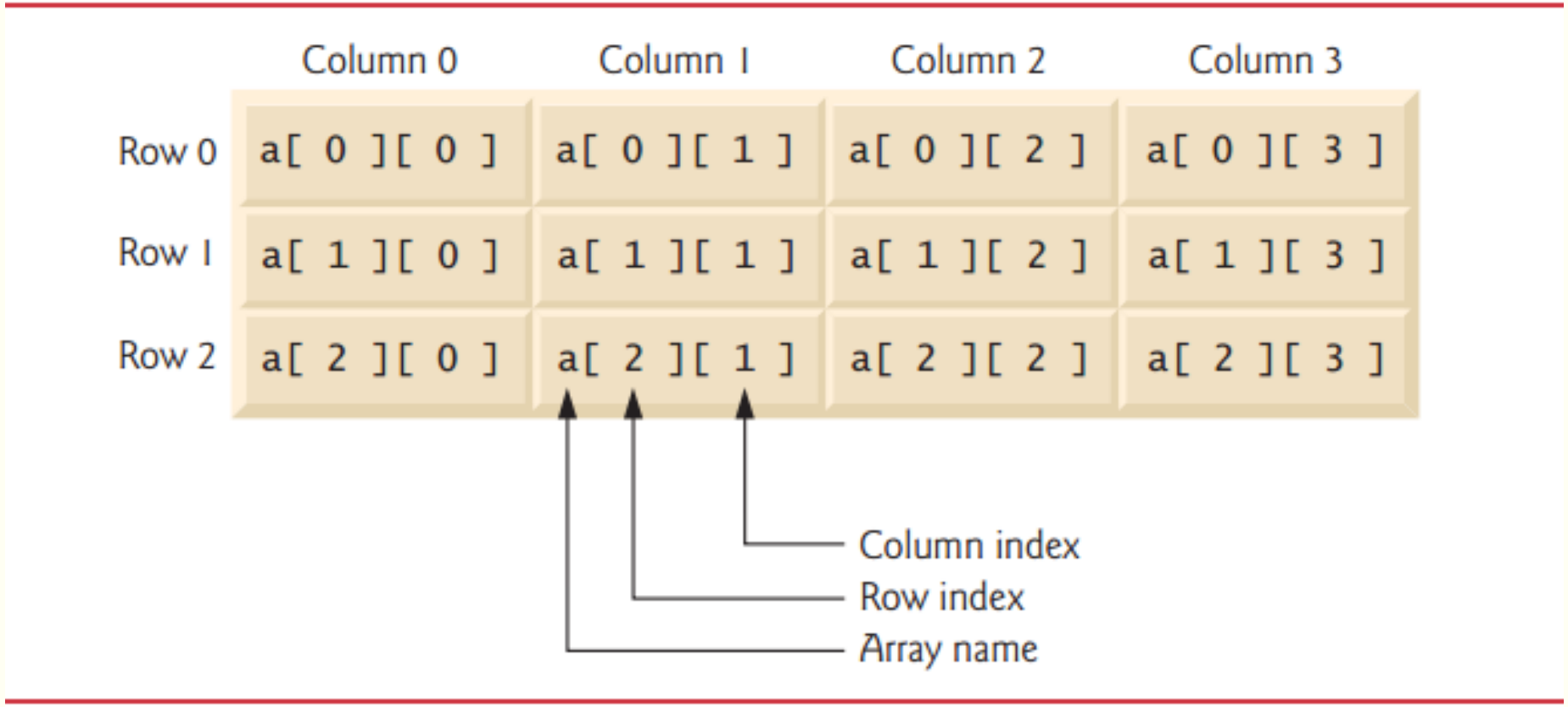
```
34
35      // multiply each element of an array by 2
36      public static void modifyArray(int[] array2)
37      {
38         for (int counter = 0; counter < array2.length; counter++)
39            array2[counter] *= 2;
40      }
41
42      // multiply argument by 2
43      public static void modifyElement(int element)
44      {
45         element *= 2;
46         System.out.printf(
47            "Value of element in modifyElement: %d%n", element);
48      }
49   } // end class PassArray
```

```
Effects of passing reference to entire array:
The values of the original array are:
   1   2   3   4   5

The values of the modified array are:
   2   4   6   8   10

Effects of passing array element value:
array[3] before modifyElement: 8
Value of element in modifyElement: 16
array[3] after modifyElement: 8
```

Оператор у рядку 45 не має ефекту на значення array[3] у main, проте оператор в рядку 39 спричинить зміни

Багатовимірні масиви

- Двовимірні масиви часто використовуються для представлення таблиць
  - Необхідно задавати 2 індекси – для рядка і для стовпця у формі a[*row*][*column*];



Масиви одновимірних масивів

- Ініціалізація відбувається саме по такому принципу з групуванням за рядком

```
int[][] b = {{1, 2}, {3, 4}};
```

  - Кількість рядків та стовпців може відрізнятись.

- Багатовимірні масиви підтримуються як одновимірні масиви.
  - Тому масив b насправді складається з 2 елементів (окремих одновимірних масивів).

- Вимоги, щоб довжини рядків у масиві b були однакові немає.
  - Наприклад,

```
int[][] b = {{1, 2}, {3, 4, 5}};
```

- Створення двовимірного масиву 3x4: `int[][] b = new int[3][4];`

  - Для задавання розмірів можна використовувати змінні, оскільки new створює масиви під час виконання, а не компілювання.
  - Елементи багатовимірного масиву ініціалізуються, коли створюється об'єкт-масив.

```
int[][] b = new int[2][];   // create 2 rows
b[0] = new int[5]; // create 5 columns for row 0
b[1] = new int[3]; // create 3 columns for row 1
```

Приклад двовимірного масиву: виведення даних

```
1  // Fig. 7.17: InitArray.java
2  // Initializing two-dimensional arrays.
3
4  public class InitArray
5  {
6     // create and output two-dimensional arrays
7     public static void main(String[] args)
8     {
9        int[][] array1 = {{1, 2, 3}, {4, 5, 6}};
10       int[][] array2 = {{1, 2}, {3}, {4, 5, 6}};
11
12       System.out.println("Values in array1 by row are");
13       outputArray(array1); // displays array1 by row
14
15       System.out.printf("%nValues in array2 by row are%n");
16       outputArray(array2); // displays array2 by row
17    }
18
19    // output rows and columns of a two-dimensional array
20    public static void outputArray(int[][] array)
21    {
22       // loop through array's rows
23       for (int row = 0; row < array.length; row++)
24       {
25          // loop through columns of current row
26          for (int column = 0; column < array[row].length; column++)
27             System.out.printf("%d  ", array[row][column]);
28
29          System.out.println();
30       }
31    }
32 } // end class InitArray
```

```
Values in array1 by row are
1  2  3
4  5  6

Values in array2 by row are
1  2
3
4  5  6
```

Поширені маніпуляції з масивами за допомогою for

Присвоєння елементам стовпця нульових значень

```
for (int column = 0; column < a[2].length; column++)
   a[2][column] = 0;
```

Додавання всіх елементів масиву

```
int total = 0;

for (int row = 0; row < a.length; row++)
{
   for (int column = 0; column < a[row].length; column++)
      total += a[row][column];
}
```

Клас Arrays

```
1  // Fig. 7.22: ArrayManipulations.java
2  // Arrays class methods and System.arraycopy.
3  import java.util.Arrays;
4
5  public class ArrayManipulations
6  {
7     public static void main(String[] args)
8     {
9        // sort doubleArray into ascending order
10       double[] doubleArray = { 8.4, 9.3, 0.2, 7.9, 3.4 };
11       Arrays.sort(doubleArray);
12       System.out.printf("%ndoubleArray: ");
13
14       for (double value : doubleArray)
15          System.out.printf("%.1f ", value);
16
17       // fill 10-element array with 7s
18       int[] filledIntArray = new int[10];
19       Arrays.fill(filledIntArray, 7);
20       displayArray(filledIntArray, "filledIntArray");
21
```

- Постачає готові статичні методи для поширених дій з масивами
  - sort() для сортування масиву,
  - binarySearch() для пошуку у відсортованому масиві,
  - equals() для порівняння масивів
  - fill() для вставки значень у масиви.
- Метод System.arraycopy дозволяє копіювати масиви

Клас Arrays

```
22          // copy array intArray into array intArrayCopy
23          int[] intArray = { 1, 2, 3, 4, 5, 6 };
24          int[] intArrayCopy = new int[intArray.length];
25          System.arraycopy(intArray, 0, intArrayCopy, 0, intArray.length);
26          displayArray(intArray, "intArray");
27          displayArray(intArrayCopy, "intArrayCopy");
28
29          // compare intArray and intArrayCopy for equality
30          boolean b = Arrays.equals(intArray, intArrayCopy);
31          System.out.printf("%n%nintArray %s intArrayCopy%n",
32             (b ? "==" : "!="));
33
34          // compare intArray and filledIntArray for equality
35          b = Arrays.equals(intArray, filledIntArray);
36          System.out.printf("intArray %s filledIntArray%n",
37             (b ? "==" : "!="));
```

Рядок 25 копіює елементи масиву intArray в intArrayCopy.
Рядки 30 та 35 викликають статичний метод Arrays.equals(), щоб визначити, чи еквівалентні всі елементи масиву. Якщо елементи однакові та в одному порядку, повертає true; інакше – false.

Клас Arrays

```
39          // search intArray for the value 5
40          int location = Arrays.binarySearch(intArray, 5);
41
42          if (location >= 0)
43             System.out.printf(
44                "Found 5 at element %d in intArray%n", location);
45          else
46             System.out.println("5 not found in intArray");
47
48          // search intArray for the value 8763
49          location = Arrays.binarySearch(intArray, 8763);
50
51          if (location >= 0)
52             System.out.printf(
53                "Found 8763 at element %d in intArray%n", location);
54          else
55             System.out.println("8763 not found in intArray");
56       }
```

- Рядки 40 та 49 викликають статичний метод Arrays.binarySearch(), щоб виконати бінарний пошук в масиві intArray, використовуючи другий аргумент (відповідно 5 і 8763) в якості ключа.

Клас Arrays

```
58      // output values in each array
59      public static void displayArray(int[] array, String description)
60      {
61         System.out.printf("%n%s: ", description);
62
63         for (int value : array)
64            System.out.printf("%d ", value);
65      }
66   } // end class ArrayManipulations
```

```
doubleArray: 0.2 3.4 7.9 8.4 9.3
filledIntArray: 7 7 7 7 7 7 7 7 7 7
intArray: 1 2 3 4 5 6
intArrayCopy: 1 2 3 4 5 6

intArray == intArrayCopy
intArray != filledIntArray
Found 5 at element 4 in intArray
8763 not found in intArray
```

Проблема автоматичної зміни розміру масиву

- Java API постачає кілька наперед заданих структур даних (колекцій), що зберігають групи пов'язаних об'єктів у пам'яті.
- Клас-колекція ArrayList<T> (пакет java.util) може динамічно змінювати свій розмір, щоб пристосувати більше елементів.
  - За домовленістю, T – це заповнювач (placeholder), який заміняється конкретним типом елементів при створенні нового ArrayList.
  - Класи з таким типом підстановками називаються узагальненими (дженериками, generic classes – лекція 3).
  - Лише непримітивні типи можна використовувати, щоб оголосити змінні та створити об'єкти з узагальнених класів.

```
ArrayList<String> list;
```

```
ArrayList<Integer> integers;
```

ДЯКУЮ ЗА УВАГУ!

Наступне запитання: інкапсуляція та приховування інформації в Java

Поширені методи з класу ArrayList<T>

Method	Description
`add`	Adds an element to the *end* of the `ArrayList`.
`clear`	Removes all the elements from the `ArrayList`.
`contains`	Returns `true` if the `ArrayList` contains the specified element; otherwise, returns `false`.
`get`	Returns the element at the specified index.
`indexOf`	Returns the index of the first occurrence of the specified element in the `ArrayList`.
`remove`	Overloaded. Removes the first occurrence of the specified value or the element at the specified index.
`size`	Returns the number of elements stored in the `ArrayList`.
`trimToSize`	Trims the capacity of the `ArrayList` to the current number of elements.

Демонстрація ArrayList<String>

```
1  // Fig. 7.24: ArrayListCollection.java
2  // Generic ArrayList<T> collection demonstration.
3  import java.util.ArrayList;
4
5  public class ArrayListCollection
6  {
7     public static void main(String[] args)
8     {
9        // create a new ArrayList of Strings with an initial capacity of 10
10       ArrayList<String> items = new ArrayList<String>();
11
12       items.add("red"); // append an item to the list
13       items.add(0, "yellow"); // insert "yellow" at index 0
14
15       // header
16       System.out.print(
17          "Display list contents with counter-controlled loop:");
18
19       // display the colors in the list
20       for (int i = 0; i < items.size(); i++)
21          System.out.printf(" %s", items.get(i));
22
```

- Метод add() додає елементи до ArrayList (рядки 12–13).
  - З одним аргуметом – додає в кінець ArrayList.
  - З двома аргументами – в задану позицію (перший аргумент, індексація з 0). Індекси всі наступних елементів збільшуються на 1. Зазвичай операція повільніша, ніж додавання в кінець.
- Рядки 20–21 показують елементи ArrayList.
  - size() повертає поточну кількість елементів ArrayList.
  - get () отримує елемент по заданому індексу

```
23          // display colors using enhanced for in the display method
24          display(items,
25             "%nDisplay list contents with enhanced for statement:");
26
27          items.add("green"); // add "green" to the end of the list
28          items.add("yellow"); // add "yellow" to the end of the list
29          display(items, "List with two new elements:");
30
31          items.remove("yellow"); // remove the first "yellow"
32          display(items, "Remove first instance of yellow:");
33
34          items.remove(1); // remove item at index 1
35          display(items, "Remove second list element (green):");
36
37          // check if a value is in the List
38          System.out.printf("\"red\" is %sin the list%n",
39             items.contains("red") ? "": "not ");
40
41          // display number of elements in the List
42          System.out.printf("Size: %s%n", items.size());
43       }
44
```

Рядки 24–25 відображають елементи, а 27-28 – додають ще 2 елементи в ArrayList.

Метод remove() використовується для видалення елементів з конкретним значенням (рядок 31).

- Видаляє лише перший такий елемент. Якщо такого немає в ArrayList, видалення не відбувається.
- Перевантажена версія методу видаляє елемент за індексом. (рядок 34), решта елементів після нього зсуваються назад на 1.

Рядок 39 використовує метод contains(), щоб перевірити, чи є елемент в ArrayList.

- Повертає true, якщо елемент знайдено і false в іншому разі.
- Порівнює аргумент з кожним елементом ArrayList по порядку, тому може бути повільним.

Демонстрація ArrayList<String>

```
45      // display the ArrayList's elements on the console
46      public static void display(ArrayList<String> items, String header)
47      {
48         System.out.printf(header); // display header
49
50         // display each element in items
51         for (String item : items)
52            System.out.printf(" %s", item);
53
54         System.out.println();
55      }
56   } // end class ArrayListCollection
```

```
Display list contents with counter-controlled loop: yellow red
Display list contents with enhanced for statement: yellow red
List with two new elements: yellow red green yellow
Remove first instance of yellow: red green yellow
Remove second list element (green): red yellow
"red" is in the list
Size: 2
```

Алмазна (<>) нотація для створення об'єкту узагальненого класу

```
ArrayList<String> items = new ArrayList<String>();
```

- Вираз ArrayList<String> з'являється в оголошенні змінної та при створенні екземпляру класу.
  - Java SE 7 представила алмазну (<>) нотацію для спрощення таких інструкцій
  - Використовуючи <> у виразі для створення екземпляру узагальненого класу вказує компілятору визначити, what belongs in the angle brackets.
  - У Java SE 7+ можна переписати інструкцію так:

```
ArrayList<String> items = new ArrayList<>();
```

  - Коли компілятор зустрічає <> у виразі створення екземпляру класу, він використовує оголошення змінної для визначення типу елементів ArrayList's (String)—це називають винесенням типу (inferring the element type).