РОБОТА З РЯДКАМИ

Лекція 2.2

доц. кафедри Інформатики
Сінельнікова Т.Ф.

ХНУРЕ, Кафедра Інформатики

informatika@kture.kharkov.ua

- Обробка рядків
- String-конструктори
- Довжина рядка
- Спеціальні рядкові операції
- Рядкові літерали
- Конкатенація рядків
- Конкатенація інших типів даних
- Перетворення рядків і метод toString ()
- Вилучення символів
- Метод charAt()
- Метод getChars()
- Метод getBytes()
- Метод toCharArray()
- Порівняння рядків
- Методи equals() і equalsIgnoreCase()
- Метод regionMatches()
- Методи startsWith() і endsWith()
- Порівняння equals() та операції ==
- Метод compareTo()
- Пошук рядків

Обробка рядків

- Як у більшості інших мов програмування, рядок в Java - це послідовність символів.
- Але, на відміну від багатьох мов, які реалізують рядки як символьні масиви, в Java рядка реалізуються як об'єкти типу string.
- Реалізація рядків у вигляді вбудованих об'єктів забезпечує повний комплект властивостей, які роблять обробку рядків дуже зручною.
- Після створення string-об'єкта символи, що входять в рядок, не можна змінювати.
- Над цим об'єктом можна виконувати всі типи строкових операцій.
- Для того щоб змінити версію існуючої рядка, потрібно створити новий об'єкт типу string, який містить необхідну модифікацію.

Обробка рядків

- Для тих випадків, коли бажана змінна рядок, існує компаньйон класу string з ім'ям stringBuffer, чиї об'єкти містять рядки, які можуть змінюватися після їх створення.
- Класи string і stringBuffer визначені в пакеті java.iang. Таким чином, вони доступні всім програмам автоматично.
- Обидва оголошені як final, що означає, що жоден з цих класів не може мати підкласів.
- Коли говорять, що рядки в об'єктах типу string є незмінними, це означає, що вміст string-об'єкта не може бути модифіковано після того, як він був створений. Однак змінну, оголошену як string-посилання, можна в будь-який час змінити так, щоб вона вказувала на інший string-об'єкт.

String-конструктори

- Клас string підтримує кілька конструкторів. Щоб створити порожній об'єкт типу string, потрібно викликати замовчуваний конструктор. Наприклад, наступний оператор

String s = new String ();

- створює екземпляр класу string, не містить символів (тобто порожній рядок).
- Щоб створити string-об'єкт, ініціалізований масивом символів, використовуйте наступний конструктор:

String (char chars [])
наприклад:
char chars [] = {'a', 'b', 'c'};
String s = new String (chars);
Цей конструктор ініціалізує (об'єктну) змінну s рядком "abc".

String-конструктори

- Як ініціалізатор можна вказати піддіапазон символьного масиву, для чого використовується наступний конструктор:

String (char chars [], int startlndex, int numChars)

- де startindex визначає індекс (індекс - це просто порядковий номер символу в рядку, причому нумерація виконується як в масиві - з нуля), з якого починається піддіапазон; numChars визначає число символів в діапазоні. наприклад:

char chars [] = {'a', 'b', 'з', 'd' 'e', 'f'}; String s = new String (chars, 2, 3);

- що ініціалізує рядковий об'єкт s символами cde.
- За допомогою конструктора

String (String strObj)

- можна створити string-об'єкт, який містить таку ж символьну послідовність, як інший string-об'єкт. Тут strObj - об'єкт типу string.

String-конструктори

- Тип char в Java використовує 16-розрядне представлення символів (з набору Unicode), тоді як в Internet для представлення рядкових символів використовується 8-розрядний набір символів ASCII. Оскільки 8-розрядні рядки ASCII використовуються досить широко, в класі string є конструктори, які ініціалізували рядок 8-розрядними byte-масивами.

Форми цих конструкторів такі:

String (byte asciiChars [])

String (byte asciiChars [], int startIndex, int numChars)

де asciiChars вказує байтовий масив. Друга форма дозволяє вказати піддіапазон. У кожному з цих конструкторів перетворення байтів в символи використовує кодовий набір символів платформи, заданий за замовчуванням.

Довжина рядка

- Довжина рядка визначається кількістю містяться в ній символів. Для отримання цього значення викличте метод length () у формі:

int length ()

- Наступний фрагмент виведе число 3, т. к. в рядку s є три символи:

```
char chars [] = {'a', 'b', 'з'};
    String s = new String (chars);
    System.out.printIn (s.length ());
```

Спеціальні рядкові операції

- Оскільки робота з рядками - звичайна і дуже важлива частина програмування, в синтаксис мови Java додана підтримка для деяких спеціальних строкових операцій.
- До цих операцій належать автоматичне створення нових string-об'єктів із строкових літералів, конкатенація множинних string-об'єктів за допомогою операції + і перетворення інших типів даних в строкове подання.
- Існують явні методи для реалізації всіх цих функцій, але Java виконує їх автоматично як для зручності програміста, так і для того, щоб зробити запис програми більш ясною.

Рядкові літерали

- Для кожного рядкового літерала в програмі Java автоматично створює string-об'єкт. Наприклад, наступний кодовий фрагмент створює дві еквівалентні рядки:

```
char chars [] = {'a', 'b', 'з'};
    String s1 = new String (chars);
    String s2 = "abc"; / / використання рядкового літерала
```

- Оскільки об'єкт типу string створюється для кожного строкового літерала, то цей літерал можна застосовувати в будь-якому місці, де вказується string-об'єкт.

```
System.out.println ("abc". Length ());
```

- Тут рядковий літерал ("abc") вказано на місці, де повинна б була стояти об'єктна посилання s2 з попереднього фрагмента. Аргумент "abc". Length про викликає метод length () прямо для рядка "abc" (замість того, щоб викликати його для об'єкта s2).

Конкатенація рядків

- Взагалі, Java не дозволяє застосовувати операції до string-об'єктів. Проте в цьому правилі є один виняток. Це операція +, яка пов'язує два рядки, будуючи в результаті string-об'єкт з об'єднаною послідовністю символів. Можна також організувати ланцюжок з декількох + операцій. Наприклад, наступний фрагмент пов'язує три рядки:

```
String age = "9";
    String s = "Йому" + age + "років.";
    System.out.println (s);
```

- Тут відбувається конкатенація (зчеплення) трьох рядків, в результаті якої на екран виводиться рядок "Йому 9 років.".
- Ще одне практичне використання конкатенації - це створення дуже довгих рядків. Замість введення довгих послідовностей символів у вихідний код можна розбити їх на менші частини і використовувати ланцюжок + операцій для їх зчеплення. наприклад:

```
/ / Використання конкатенації для створення довгих рядків.
    class ConCat {
    public static void main (String args []) {
    String longStr = "Це була б дуже довгий рядок," + "не зручний ні для введення, ні
    для виводу." + "Але конкатенація рядків" + "усуває цей недолік.";
    System.out.println (longStr);)}
```

Конкатенація інших типів даних

- Операцію конкатенації рядків можна використовувати з іншими типами даних.
  Наприклад, розглянемо наступну, трохи змінену, версію попереднього прикладу:

```
int age = 9;
String s = "Йому" + age + "років.";
System.out.println (s);
```

- У цьому випадку змінна age - має int-тип, а не string, як в попередньому фрагменті, але висновок - такий же, як раніше. Це тому, що int-значення змінної age автоматично перетворюється в її рядкове подання всередині string-об'єкта і потім зчіплюється аналогічним способом.
- Однак будьте уважні, коли змішуєте інші типи операцій у виразах конкатенації рядків. Ви можете отримати несподівані результати. Розглянемо наступний фрагмент:

```
String s = "чотири:" + 2 + 2; System.out.println (s);
```

- Цей фрагмент виводить на екран:

```
чотири: 22
```

- а не чотири: 4. З урахуванням старшинства операцій спочатку виконується конкатенація першого операнда ("чотири:") з рядковим еквівалентом другого ("2"). Цей результат потім зчіплюється з рядковим еквівалентом третій операнд (теж "2"). Щоб виконати спочатку цілочисельне складання, потрібно використовувати круглі дужки:
  String s = "чотири:" + (2 +2); Тепер s містить рядок "чотири: 4".

Перетворення рядків і метод toString ()

- Кожен клас реалізує toString (), оскільки даний метод визначено в класі Object.
- Проте реалізації toString (), заданої за замовчуванням, рідко достатньо. Метод toString () має таку загальну форму:

String toString ()

- Реалізація toString () просто повертає об'єкт типу string, який містить легку для читання рядок, що описує об'єкт вашого класу.
- Перевизначаючи toString () для створюваних вами класів, ви отримуєте рядкові представлення об'єктів, повністю інтегровані в середовище програмування Java.
- Наприклад, вони можуть використовуватися в операторах print () і printin () у виразах конкатенації.

Перетворення рядків і метод toString ()

```
//Перевизначення toString () для Box-класів.
    class Box {
    double width;
    double height;
    double depth;
    Box (double w, double h, double d) {
    width = w;
    height = h;
    depth = d;}

    public String toString ()
    {
    return "Розміри Box-об'єкту:" + width + "x" +
    depth + "x" + height + ".";}}
    class toStringDemo {
    public static void main (String args []) (Box b =
    new Box (10, 12, 14);
    String s = "Box b:" + b;
    / / Конкатенація Box-об'єкта
    System.out.println (b);
    / / Перетворення Box-об'єкта в рядок
    / / System.out.println (s);
    }
    }
```

- Висновок цієї програми:

Розміри Box-об'єкту: 10 x 14 x 12.
Box b: Розміри Box-об'єкту: 10 x 14 x 12.

Зверніть увагу, що метод toString () викликається автоматично, коли Box-об'єкт використовується у виразі конкатенації або у зверненні до println ().

Вилучення символів

- Клас String надає кілька способів вилучення символів з об'єкта типу String.
- Хоча символи, які складають рядок String-об'єкта, не можуть бути індексовані, як в символьному масиві, багато з String-методів використовують індекс (порядковий номер або позицію) символу в рядку для виконання своїх операцій.
- Подібно масивів, індекс рядка починається з нуля.

Метод *charAt()*

- Для витягання одиночного символу з String-об'єкта ви можете прямо посилатися на індивідуальний символ через метод charAt ().

char charAt (int where)

- де параметр where - індекс (номер) символу, який ви хочете отримати. Значення where має визначати позицію шуканого символу в рядку і не може бути негативним. charAt () повертає символ, що знаходиться у зазначеній позиції рядка. Наприклад, фрагмент:

char ch;
ch = "abc". charAt (1);
призначає символьне значення b змінної ch.

Метод *getChars()*

- Якщо потрібно витягти більше одного символу, то можна використовувати метод getChars ().

void getChars (int sourceStart, int sourceEnd, char target [], int targetStart)

- Тут sourceStart вказує індекс початку підрядка;
- sourceEnd вказує індекс, який на 1 більше індексу кінця бажаною підрядка. Таким чином, підрядок містить символи в позиціях від SourceStart до sourceEnd-1.
- Масив, який прийматиме символи, вказується параметром target []. Позиція в target, починаючи з якої буде скопійована підрядок, передається через параметр targetstart.

```
class getCharsDemo {
    public static void main (String args []) {
    String s = "This is a demo of the getChars method.";
    int start = 10;
    int end = 14;
    char buf [] = new char [end - start];
    s.getChars (start, end, buf, 0); System.out.println (buf);}}
```

- Висновок цієї програми:
  demo

Метод *getBytes()*

- Є альтернатива getChars (), яка зберігає символи в масиві байтів.
- Цей метод називається getBytes (). Він виконує перетворення символів в байти задане за замовчуванням на використовуваної платформі.

Byte [] getBytes ()

- Є й інші форми getBytes ().
- getBytes () найбільш корисний, коли ви експортуєте string-значення в середовище, яке не підтримує 16-розрядні символи Unicode.
- Наприклад, більшість протоколів Internet і форматів текстових файлів використовує 8-розрядну кодування ASCII для всього текстового обміну.

Метод *toCharArray()*

- Якщо ви хочете перетворити всі символи в об'єкті типу String в символьний масив, найпростіший спосіб полягає у виклику методу toCharArray ().
- Він повертає масив символів всього рядка і має таку загальну форму:

char [] toCharArray ()

- Ця функція забезпечує певні зручності, так як досягти того ж результату можна і за допомогою методу getChars ().

Порівняння рядків

- Клас String включає декілька методів, які порівнюють рядки або підрядка всередині рядків.

Методи *equals()* і *equalsIgnoreCase()*

- Щоб порівнювати два рядки на рівність, потрібно використовувати метод equals (). Він має таку загальну форму:

boolean equals (Object str)

де str - string-об'єкт, який порівнюється із зухвалим string-об'єктом. Метод повертає значення true, якщо рядки містять одні й ті ж символи в однаковому порядку, інакше повертається false. Порівняння чутливе до регістру.

- Щоб виконати порівняння, яке ігнорує відмінності у регістрі, викликається метод equalsIgnoreCase (). При порівнянні двох рядків він припускає, що символи A-Z і a-z не розрізняються. Загальний формат цього методу:

boolean equalsIgnoreCase (String str)

де str - string-об'єкт, який порівнюється із зухвалим string-об'єктом. Він теж повертає true, якщо рядки містять одні й ті ж символи в одному і тому ж порядку, інакше повертає false.

Методи *equals()* і *equalsIgnoreCase()*

```
// Демонструє equals() і equalsIgnoreCase().
class equalsDemo {
public static void main(String args[]) (
String s1 = "Hello";
String s2 = "Hello";
String s3 = "Good-bye";
String s4 = "HELLO";
System.out.println(s1 + " равно " + s2 + " -> " + si.equals(s2));
System.out.println(s1+ " равно " + s3 + " -> " + si.equals(s3));
System.out.println(s1 + " равно " + s4 + " -> " + si.equals(s4));
System.out.println(s1 + " equalsIgnoreCase " + s4 + " -> " + s1.equalsIgnoreCase
    (s4)) ; } }
```

- Вывод этой программы:

```
Hello      равно Hello -> true
Hello      равно equals Good-bye -> false
Hello      равно equals HELLO -> false
Hello      equalsIgnoreCase HELLO -> true
```

Метод *regionMatches()*

- Метод regionMatches () порівнює деяку область всередині рядкового об'єкта з іншого деякою областю в іншому рядковому об'єкті.

boolean regionMatches (int startIndex, String str2, int str2StartIndex, int numChars)

boolean regionMatches (boolean ignoreCase, int startIndex, String str2, int str2StartIndex, int numChars)

- Для обох версій startIndex визначає індекс, з якого область починається в зухвалій string-об'єкті.
  Порівнюваний string-об'єкт вказується параметром str2.
  Індекс, в якому порівняння почнеться всередині str2, визначається параметром str2startIndex.
- Довжина порівнюєш підрядка пересилається через numChars. У другій версії, якщо gnoreCase - true, реєстр символів ігнорується. Інакше, реєстр враховується.

Методи startsWith() і endsWith()

- У класі string визначено дві підпрограми, які є спеціалізованими формами методу regionMatches ().
- Метод startsWith () визначає, чи починається даний String-об'єкт із зазначеною рядка. Навпаки, метод endsWith () визначає, закінчується чи string-об'єкт зазначеної рядком. Вони мають такі загальні форми:

boolean startsWith (String str)

boolean endsWith (String str)

де str - перевіряється string-об'єкт. Якщо рядки узгоджені, повертається true, інакше - false.

- наприклад,
  "Foobar". EndsWith ("bar")
- і
  "Foobar". StartsWith ("Foo")
- обидва повертають true.

Методи *startsWith()* і *endsWith()*

- Друга форма startsWith () за допомогою свого другого параметра (startindex) дозволяє визначити початкову точку області порівняння (в зухвалій об'єкті):

  boolean startsWith (String str, int startindex)

  де startindex визначає індекс символу в зухвалому рядку, з якого починається пошук символів для операції порівняння.
- наприклад:

"Foobar". StartsWith ("bar", 3)

- повертає true (тому що рядок першого аргументу виклику точно збігається з підрядком "Foobar", що починається з четвертой1 позиції у вихідній рядку).

Порівняння *equals()* та операції ==

- Важливо зрозуміти, що метод equals () і оператор == виконують дві різних операції.
- Метод equals про порівнює символи усередині string-об'єкта, а оператор == - дві об'єктні посилання, щоб бачити, звертаються вони до одного й того ж екземпляру (об'єкту).

```
/ / Equals () в порівнянні з ==
    class EqualsNotEqualTo {
    public static void main (String args []) {
    String s1 = "Hello";
    String s2 = new String (sl);
    System.out.println (s1 + "дорівнює" + s2 + "->" + s1.equals (s2));
    System.out.println (si1 + "==" + s2 + "->" + (s1 == s2));}}
```

- Змінна s1 посилається на string-екземпляр, створений рядком "Hello". Об'єкт, на який вказує s2, створюється з об'єктом s1 як ініаналізатора. Таким чином, вміст двох string-об'єктів ідентично, але це - різні об'єкти. Це означає, що s1 і s2 не посилаються на один і той же об'єкт і. Тому, при порівнянні з допомогою операції == виявляються не рівними, як показує висновок попереднього прикладу:

```
Hello дорівнює Hello -> true
    Hello == Hello -> false
```

Метод *compareTo()*

- Часто, не досить просто знати, ідентичні чи два рядки. Для додатків сортування потрібно знати, яка з них менше, дорівнює, або більше ніж інша.
- Один рядок вважається менше ніж інша, якщо вона розташована перед іншою в словниковому (упорядкованому за алфавітом) списку.
- Рядок вважається більше ніж інша, якщо вона розташована після іншої в словниковому списку.

int compareTo (String str)

- Тут str - string-об'єкт, порівнюваний із зухвалим string-об'єктом. Результат порівняння повертається (в зухвалу програму) і інтерпретується так:

Менше нуля: рядок виклику - менше, ніж str.
Більше нуля: рядок виклику - більше, ніж str.
Нуль: два рядки рівні.

Метод *compareTo()*

```
//Бульбашкове сортування рядків,
class SortString {
static String arr[] = {
"Now", "is", "the", "time", "for", "all",
    "good", "men", "to", "come", "to", "the",
    "aid", "of", "their", "country" I ;
public static void main(String args[])
{ for(int j =0; j < arr.length; j++)
{
for(int i = j +1; i < arr.length; i++)
 { if(arr[i].compareTo(arr[j]) < 0)
{ String t = arr[j];
arrfj] = arr[i];
arr[i] = t; } }
System.out.println(arr[j]); } } }
```

- Вивод цієї програми :

```
Now
aid
all
come
country
for
good
is
men
of
the
the
their
time
to
to
```

- Як ви бачите, compareTo () приймає до уваги символи нижнього і верхнього регістру.
- Якщо ви хочете ігнорувати відмінності у регістрі при порівнянні двох рядків, використовуйте метод *int compareToignoreCase (String str)*
- Цей метод був доданий в Java 2.

Пошук рядків

- Клас string надає два методи, які дозволяють виконувати пошук зазначеного символу або підрядка всередині рядки:
  - Indexof (). Пошук першого входження символу або підрядка.
  - lastIndexOf (). Пошук останнього входження символу або підрядка.
  - При невдалому пошуку повертається - 1. Для пошуку першого входження символу використовуйте

int indexOf (int ch)

  - Для пошуку останнього входження символу використовуйте

int lastIndexOf (int ch)

Тут ch - розшукуваний символ.

  - Для пошуку першого або останнього входження підрядка використовуйте

int indexOf (String str) int lastIndexOf (String str}
Тут str визначає підрядок.

- Можна визначити початкову точку пошуку, застосовуючи такі форми:

int indexOf (int ch, int startIndex)
int lastIndexOf (int ch, int startIndex)
int indexOf (String str, int startIndex)
int lastIndexOf (String str, int startIndex)

Тут startIndex вказує індекс (номер) символу, з якого починається пошук для indexOf () пошук виконується від символу з індексом startIndex до кінця рядка. Для lastIndexOf про пошук виконується від символу з індексом startIndexOf () до нуля.

Пошук рядків

```
// Демонстрирує indexOf{) к lastIndexOf().
class indexOfDemo {
public static void main(String args[]) {
String s = "Now is the time for all good men " +
"to come to the aid of their country.";
System.out.println(s);
System.out.println("indexOf(t) = " +
      s.indexOf('t'));
System.out.println("lastIndexOf(t) =        "        +
      s.lastIndexOf('t'));
System.out.println("indexOf(the) = "        +
      s.indexOf("the"));
System.out.println("lastIndexOf(the)        =        " +
      s.lastIndexOf("the"));
System.out.println("indexOf(t, 10) =        "        +
      s.indexOf('t', 10));
System.out.println("lastIndexOf(t, 60)      = " +
      s.lastIndexOf('t', 60));
System.out.println("indexOf(the, 10)        =        " +
      s.indexOf("the", 10));
System.out.println("lastIndexOf(the, 60) = " +
s.lastIndexOf("the", 60)); } }
```

- Вивод цієї програми:

```
Now is the time for all good men to come to the aid
      of their country.
indexOf(t) = 7
lastIndexOf(t) = 65
indexOf(the) = 7
lastIndexOf(the) = 55
indexOf(t, 10) =11
lastIndexOf(t, 60) =55
indexOf(the, 10) =44
lastIndexOf (the, 60) =55
```

Зміна рядка

- Оскільки String-об'єкти незмінні, всякий раз, коли ви хочете змінити String-об'єкт, потрібно або копіювати його в stringBuffer, або використовувати один з наступних String-методів, які створять нову копію рядки з вашими модифікаціями.

Метод *substring()*

- Ви можете витягти підрядок за допомогою методу substring (). Він має дві форми. Перша:

String substring (int startlndex)

- Тут startlndex специфікує індекс символу, з якого почнеться підрядок. Ця форма повертає копію підрядка, що починається з номера startlndex і тягнеться до кінця рядка виклику.
- Друга форма substring!) Дозволяє вказувати як початковий, так і кінцевий індекси підрядка:

String substring (int startlndex, int endlndex)

- Тут startlndex вказує початковий індекс; endindex визначає індекс останнього символу підрядка.
- Возвращает рядок містить всі символи від початкового до кінцевого індексу (але не включаючи символ з кінцевим індексом).

Метод *substring()*

```
// Заміна підрядка,
class StringReplace {
public static void main(String args[]) {
String org = "This is a test. This is, too.";
String search = "is";
String sub = "was";
String result = "";
int i ;
do {
// замінити всі підрядки, що співпали
System.out.println(org) ;
i = org.indexOf(search) ;
if(i != -1) {
result = org.substring(0, i);
result = result + sub;
result = result + org.substring(i + search.length());
org = result; } }
while(i != -1);
} }
```

- Вивод цієї програми:

```
This is a test. This is, too.
Thwas is a test. This is, too.
Thwas was a test. This is, too.
Thwas was a test. Thwas is, too.
Thwas was a test. Thwas was, too.
```

Метод *concat()*

- Можна зчіплювати два рядки, використовуючи метод concat о, з такою сигнатурою:

String concat (String str)

- Даний метод створює новий об'єкт, що включає рядок виклику з вмістом об'єкта str, доданим в кінець цього рядка, concat () виконує ту ж функцію, що й операція конкатенації +.
- Наприклад, фрагмент

```
String s1 = "one";
    String s2 = s1.concat ("two");
```

- Поміщає рядок "onetwo" в s2. Він генерує той же результат, що наступна послідовність:

```
String s1 = "one";
    String s2 = s1 + "two";
```

Метод *replace()*

- Метод replace () замінює все входження одного символу в рядку виклику іншим символом.

String replace (char original, char replacement)

- Тут original визначає символ, який буде замінений символом, зазначеним в replacement. Рядок, отримана в результаті заміни, повертається в зухвалу програму. наприклад,

String s = "Hello". Replace ('l', 'w');

перешкодить в s рядок "Hewwo".

Метод *trim()*

- Метод trim () повертає копію рядка виклику, з якої вилучені будь-які ведучі і завершальні прогалини.

String trim ()

- Приклад:

String s = "Hello World". Trim ();

Цей оператор поміщає в рядок s "Hello World".

- Метод trim про дуже корисний, коли ви обробляєте команди користувача.
- Наприклад, наступна програма запитує у користувача назва штату і потім відображає столицю цього штату. Вона використовує trim про для видалення будь-яких провідних і завершальних прогалин, які, можливо, по необережності були введені користувачем.

Метод *trim()*

```
// Використання trim() для обробки команд,
import java.io.*;
class UseTrim (
public static void main(String args [ ])
throws lOException {
// створити BufferedReader, що
    використовує System.in
BufferedReader br = new
BufferedReader(new
    InputStreamReader(System.in));
String str;
System.out.println("Enter 'stop1 to quit.");
 System.out.println("Enter State: ");
do {
str = br.readLine();
str = str.trimO;       // видалити пробіли
if(str.equals("Illinois"))
System.out.println("Capital is Springfield.");
else if(str.equals("Missouri"))
System.out.println("Capital is Jefferson
    City.");
else if(str.equals("California"))
System.out.println("Capital is Sacramento.");
else if(str.equals("Washington"))
System.out.println("Capital is Olympia.");
// ... }
while{!str.equals("stop"));
} }
```

Перетворення даних, що використовує метод valueOf()

- Метод vaiueOf () перетворює дані з їх внутрішнього формату в зручну для читання форму.
- Це статичний метод, який переобтяжений в класі string для всіх вбудованих типів Java так, що кожен тип можна перетворити в рядок,

static String valueOf (double niт)

static String valueOf (long num)

static String valueOf (Object ob)

static String valueOf (char chars [])

- vaiueOf про викликається, коли необхідно рядкове подання деякого іншого типу даних - наприклад, під час операцій конкатенації.
- Ви можете викликати цей метод прямо, з будь-яким типом даних в аргументі, і отримувати розумне рядкове подання.
- Всі прості типи перетворяться до їх звичайному string-поданням.
- Будь-який об'єкт, який ви передаєте в метод valueOf (), повертає результат звернення до методу toString ().
- Для більшості масивів valueOf про повертає рядок, який вказує, що це - масив деякого типу. Для масивів типу char, однак, створюється string-об'єкт, який містить символи цього масиву. Існує спеціальна версія vaiueOf (), яка дозволяє вказувати підмножина char-масиву. Вона має таку загальну форму:

static String valueOf (char chars [], int staxrtlndex, int numChars)

- Тут chars - масив, який містить символи; startindex - індекс у масиві символів, в якому починається бажана підрядок; numChars вказує довжину підрядка.

Зміна регістру символів в рядку

- Метод toLowerCase () перетворює вер символи в рядку з верхнього регістру на нижній. Метод toUpperCase () перетворить всі символи в рядку з нижнього регістру на верхній. Неалфавітних символи, типу цифр, залишаються непорушними.

  String toLowerCase ()

  String toUpperCase ()

- Обидва методи повертають об'єкт типу string, який містить верхньо-або ніжнерегістровий еквівалент викликає String-об'єкта.
- / / Демонструє toUpperCaseO і toLowerCase ().
  class ChangeCase {
  public static void main (String args []) {
  String s = "Це тест.";
  System.out.println ("Оригінал:" + s);
  String upper = s.toUpperCase (); String lower = s.toLowerCase ();
  System.out.println ("Uppercase:" + upper);
  System.out.println ("Lowercase:" + lower);}
- Висновок, виконаний цією програмою:
  Оригінал: Це тест.
  Uppercase: ЦЕ ТЕСТ.
  Lowercase: це тест.

Клас *StringBuffer*

- StringBuffer - це клас, рівний за становищем класу String.
- Він забезпечує багато функціональних можливостей для рядків.
- String представляє незмінні символьні послідовності фіксованої довжини.
  StringBuffer представляє зростаючі і перезаписувані символьні послідовності.
- StringBuffer може вставляти символи і підрядка в середину рядка або додавати їх в кінець рядка.
- StringBuffer зростає автоматично, щоб створити місце для таких додавань, і часто має більше попередньо виділеної пам'яті, ніж фактично необхідно для зростання.

Конструктори *StringBuffer*

StringBuffer()
StringBuffer(int *size)*
StringBuffer(String *str)*

- Заданий за замовчуванням конструктор (без параметрів) резервує ділянку пам'яті для шістнадцяти додаткових символів, що не беруть участь в розподілі.
- Друга версія приймає цілочисельний аргумент, який явно встановлює розмір буфера.
- Третя версія приймає string-аргумент, який встановлює початкове вміст об'єкта типу StringBuffer і резервує ділянку пам'яті для ще шістнадцяти додаткових символів.

Методи *length()* і *capacity()*

- Поточну довжину об'єкта типу StringBuffer можна знайти за допомогою методу length (), а загальний розподілений обсяг - за допомогою методу capacity (). Вони мають такі загальні форми:

int length ()

int capacity ()

- приклад:

```
/ / StringBuffer length () в порівнянні з capacity ()
.class StringBufferDemo {
    public static void main (String args []) {
    StringBuffer sb = new StringBuffer ("Hello");
    System.out-println {"buffer =" + sb);
    System.out.printing "length =" + sb.length O);
    System.out.println ("capacity =" + sb.capacity 0);}}
```

- Висновок цієї програми, який показує, як StringBuffer резервує додатковий простір для можливих маніпуляцій:

```
buffer = Hello
    length = 5
    capacity = 21
    Об'єкт sb ініціалізується
```

Метод *ensureCapacity()*

- Якщо ви хочете попередньо виділити ділянку пам'яті для деякого числа символів після того, як StringBuffer був створений, то для установки розміру буфера можна використовувати метод ensureCapacity (). Це корисно, якщо ви знаєте заздалегідь, що будете додавати (в кінець буфера StringBuffer) велика кількість маленьких рядків, ensurecapacity про має таку загальну форму:

void ensureCapacity (int capacity)

- де capacity визначає загальний розмір (ємність) буфера.

Метод *setLength()*

- Щоб встановлювати довжину буфера в межах об'єкту типу StringBuffer, використовуйте метод setLength (). Його загальна форма:

void setLength (int len)

де **len** визначає довжину буфера. Це значення має бути невід'ємним.

- Коли ви збільшуєте розмір буфера, до кінця існуючого буфера додаються нульові символи. Якщо ви викликаєте setLength () із значенням менше ніж поточне значення, повернене методом length (), то символи, що зберігаються поза нової довжини, будуть втрачені.

Методи *charAt()* і *setCharAt()*

- Значення одиночного символу можна отримати з StringBuffer за допомогою методу charAt (). Встановлювати значення символу в StringBuffer може метод setcharAt (). Їх загальні формати:

char charAt (int where)

void setcharAt (int where, char ch)

- Параметр where в charAto вказує індекс одержуваного символу. В setcharAt про where вказує індекс встановлюваного символу, а ch - нове значення цього символу. Для обох методів параметр where повинен бути невід'ємним і не повинен визначати позицію поза кінця буфера.

```
/ / Демонструє charAt () і setCharAt ().
    class setCharAtDemo {
    public static void main (String args []) {
    StringBuffer sb = new StringBuffer ("Hello");
    System.out.println ("buffer before =" + sb;
    System.out.println ("charAt (1) before =" + sb.charAt (i)); sb.setCharAt (1, 'i');
    sb.setLength (2);
    System.out.println ("buffer after =" + sb);
    System.out.println ("charAt (1) after =" + sb.charAt (1));}}
```

- Вивод, згенерований цією програмою:

```
buffer before = Hello
    charAt (1) before = e
    buffer after = Hi charAt (1) after = i
```

Метод *getChars()*

- Для копіювання підрядка StringBuffer в масив можна використовувати метод getChars (). Його загальна форма:

void getchars (int sourceStart, int sourceEnd, char target [], int targetStart)

де sovircestart вказує індекс початку підрядка; sourceEnd визначає індекс, на 1 більший, ніж індекс кінця підрядка. Це означає, що підрядок містить символи від sourceStart до sourceEnd-1. масив символів вказується параметром target. Індекс (номер) позиції в target, з якою буде скопійована підрядок, передається в targetStart. Необхідно подбати про те, щоб масив target був досить великим для розміщення всіх символів зазначеної підрядка.

Метод *append()*

- Метод append () додає рядкові подання будь-якого іншого типу даних в кінець викликає об'єкта типу stringBuffer. Він має перевантажені версії для всіх вбудованих типів і для типу object. Є кілька його форм:

  StringBuffer append (String str) StringBuffer append (int num)

  StringBuffer append (Object obj)

- Щоб отримати рядкове подання кожного параметра, викликається метод string, vaiueOf (). Результат додається в кінець поточного stringBuffer-об'єкта. Сам буфер повертається кожною версією append o. Це дозволяє поєднувати в ланцюжок всі послідовні виклики append o, як показано в наступному прикладі:

  / / Демонструє append (). class appendDemo {
  public static void main (String args []) (
  String s;
  int a = 42;
  StringBuffer sb = new StringBuffer (40);
  s = sb.append ("a ="). append (a). append ("!"). toString (); System.out.println (s); j}

- Висновок цього прикладу:
  a = 42!

Метод *insert()*

- Метод insert про вставляє один рядок в іншу.

StringBuffer insert (int index. String str)

StringBuffer insert (int index, char ch)

StringBuffer insert (int index, Object obj)

- Параметр index вказує індекс (номер позиції) (в зухвалій stringBuffer-об'єкті), з якого буде вставлена рядок, символ або об'єкт, зазначені у другому параметрі.
- Наступна програма вставляє "подобається" між "Мені" і "Java":

```
/ / Демонструє insert ().
class insertDemo {
public static void main (String args []) {
StringBuffer sb = new StringBuffer ("Мені Java!");
sb.insert (2, "подобається");
System.out.println (sb);
}
```

- Вивод цього прикладу:
  Мені подобається Java!

Метод *reverse()*

- Можна змінити порядок символів в об'єкті типу stringBuffer, використовуючи метод reverse () з форматом:

  StringBuffer reverse ()

- Даний метод повертає реверсувати (із зворотним розташуванням символів) об'єкт виклику. Наступна програма демонструє reverse ():
- // Використання reverse () для реверсу
  StringBuffer-об'єкта. class ReverseDemo {
  public static void main (String args []) {
  StringBuffer s = new StringBuffer ("abcdef");
  System.out.println (s);
  s.reverse ();
  System.out.println (s);}
  }
- Вивод цієї програми:
  abcdef
  fedcba

Методи *delete()* і *deleteCharAt()*

StringBuffer delete(int *startIndex*, int *endlndex)*

StringBuffer deleteCharAt(int *loc)*

- Метод delete () видаляє послідовність символів з об'єкта виклику. Параметр startindex вказує індекс першого видаляється символу; endlndex визначає індекс, на 1 більший, ніж в останнього видаляється. Таким чином, видаляється підрядок простягається від startindex до endlndex-1. Повертається stringBuffer-об'єкт, отриманий в результаті видалення.
- Метод deieteCharAt () видаляє символ з індексом, зазначеним в loc, і повертає результуючий StringBuffer-об'єкт.
- Програма, яка демонструє методи delete o і deieteCharAt про:
  / / Демонструє delete O і deieteCharAt ()
  class deleteDemo {
  public static void main (String args []) (
  StringBuffer sb = new StringBuffer ("Це перевірка.");
  System.out.println ("До delete:" + sb);
  sb.delete (4, 7);
  System.out.println ("Після delete:" + sb);
  sb.deieteCharAt (0);
  System. out.println ("Після deieteCharAt:" + sb);}}
- Вивод цієї програми:
  До delete: Це перевірка.
  Після delete: Це евірка.
  Після deleteCharAt: то евірка.

Метод *replace()*

- StringBuffer replace (int startIndex, int endIndex, String str)
  Замінна підрядок вказується індексами startIndex і endIndex. Таким чином, замінна підрядок займає позиції від startIndex до endIndex -1. Замінює рядок передається через параметр str. Модифікований таким чином stringBuffer-об'єкт повертається в зухвалу програму.
- / / Демонструє replace ()
  class replaceDemo (
  public static void main (String args []) {
  StringBuffer sb = new StringBuffer ("Це є тест.");
  sb.replace (3, 4, "був");
  System.out.println ("Після replace:" + sb);}}
- Вивод цієї програми:
  Після replace: Це був тест.

Метод *substring()*

String substring(int *startIndex)*

String substring(int *startIndex*, int *endIndex)*

- Перша форма повертає підрядок, яка починається в позиції startIndex і тягнеться до кінця викликає stringBuffer-об'єкта.
- Друга форма повертає підрядок, яка починається в позиції startIndex і тягнеться до позиції endIndex-1.
- Перераховані методи працюють точно так само, як аналогічні методи, визначені в класі string, які були описані раніше.