Лабораторна робота № 3

Засоби для роботи з текстом та класи-огортки

Мета роботи: Вивчити способи роботи з рядками та символами, а також засоби класів-огорток в Java.

1. Теоретичні відомості

У мові С ++ відсутня вбудована підтримка рядків. Рядок в них визначає адресу послідовності байтів (збереженої в масиві), вміст яких трактується як символи до тих пір, поки не буде зустрінутий нульовий байт, що вказує на кінець рядка. У пакеті java.lang знаходиться клас, що представляє собою структуру даних рядка. Цей клас, званий String, не що інше, як об'єктне уявлення незмінного масиву символів (він містить властивість `private final char[] value` для зберігання масиву символів рядка). Більшість конструкторів даного класу приймають в якості параметра масив елементів типу `char` або масив байтів. В класі java.lang.String є методи, які дозволяють порівнювати рядки, здійснювати в них пошук, отримувати певні символи і частини рядків та ін. Клас java.lang.StringBuffer і більш новий java.lang.StringBuilder (який є його повною копією) використовуються тоді, коли рядок після створення потрібно змінювати (їх властивість `char[] value` не має модифікатора `final`). Відмінність StringBuilder від StringBuffer полягає в тому що у StringBuilder відсутня синхронізація публічних методів між різними потоками виконання (`threads`), тому він може використовуватись тільки в однопоточних додатках і є більш продуктивним порівняно з StringBuffer. Клас String дозволяє створити *незмінні* (*immutable*) об'єкти, а класи StringBuilder і StringBuffer-*змінні* (*mutable*) об'єкти. Оскільки класи String і StringBuffer / StringBuilder оголошені як `final`, від жодного з цих класів не можна успадковувати підкласи.

Створення рядків

Як і в разі будь-якого іншого класу, об'єкти класу String створюються за допомогою оператора `new`. Для створення порожнього рядка використовується конструктор без параметрів:

```
String s = new String();
```

Наведений нижче фрагмент коду створює об'єкт `s` класу String, ініціалізувавши його масивом символів, переданим конструктору як параметр.

```
char[] chars = {'d', 'f', 'д','ж', '\u042f', '\u00a9',
                '\u01fe'};
String s = new String(chars);
System.out.println(s);
```

Цей фрагмент коду виводить рядок `dfджЯ©ǿ`. Зверніть увагу на можливість роботи з символами за допомогою їх кодів з таблиці UTF-8.

Ще один конструктор класу String має додаткові параметри:

```
String(char[] chars, int начальнийІндекс,
       int кількістьСимволів);
```

Використовуємо такий спосіб ініціалізації в наступному прикладі:

```
char[] chars = {'d', 'f', 'д','ж', '\u042f', '\u00a9',
                '\u01fe'};
String s = new String(chars, 1, 3);
System.out.println(s);
```

Цей код виведе `фдж`.

Java містить декілька синтаксичних *доповнень* - *спеціальні рядкові операції*, мета яких - допомогти програмістам у виконанні операцій з рядками. До цих операцій належать: створення об'єктів типу String зі строкових констант (літералів)(вказуються у подвійних лапках), створення нового об'єкту-рядку злиттям декількох рядків за допомогою операції конкатенації (+), перетворення інших типів даних в рядки.

Java містить стандартне скорочення для операції створення рядка - запис у вигляді строкової константи (літерала), у якій *вміст рядка оточується парой подвійних лапок*. Наведений нижче фрагмент коду еквівалентний одному з попередніх, в якому рядок ініціалізувався масивом типу `char`.

```
String s = "abc";
System.out.println(s);
```

Також можна створювати рядок посиланням на інший рядок:

```
String s1 = "привіт";
String s2 = s1;
System.out.println(s2); //привіт
```

Ще один спосіб створення рядків - передача до конструктора `String(byte[] bytes, String charsetName)` байтового масиву із зазначенням назви способу кодування символів, за допомогою якого повинні інтерпретуватися байти:

```
byte[] b={0x61,0x62,(byte)0xd0,-97,-47,-128,-48,-72,-48,-78,
          -48,-75,-47,-126};
String s = new String(b,"UTF-8");
System.out.println(s); //abcde
```

Як рядки-назви кодування символів використовуються назви, зареєстровані в реєстрі IANA Charset Registry. В Java кодуванням за замовчуванням вважається кодування *UTF-8*, при його використанні назва кодування може не зазначатитсь.

Один із загальних методів, що використовується з об'єктами String - метод `length()`, який повертає число символів в рядку. Наступний фрагмент коду виводить число 3, оскільки рядок складається з 3 символів:

```
String s = "abc";
System.out.println(s.length);
```

Оскільки для рядка-литерала створюється об'єкт класу String, може існувати можливість виклику методів цього класу безпосередньо з рядку-літералу. Наступний приклад також виводить число 3.

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

Виконати конкатенацію (об'єднання) декількох рядків можна так:

```
int age = 20;
String s = "Його вік -" + age + "років";
```

Оператор `System.out.println(s)` виводить на екран:

```
Його вік - 20 років
```

Корисність наведеного вище оператора можна зрозуміти, розглянувши альтернативний спосіб об'єднання рядків, записаний з явними викликами тих же методів, які неявно були використані в попередньому прикладі:

```
int age = 20;
String s = new StringBuilder("Його вік - ")
.append(String.valueOf(age)).append(" років").toString();
```

Породжується об'єкт класу StringBuilder, який ініціалізується рядком-літералом `"Його вік - "`, ціле значення змінної `age` передається методу `append()` об'єкта класу StringBuilder, який перетворює його у текстовий вигляд (використовуючи статичний метод `valueOf()` класу String) і додає в кінець рядку, який міститься в об'єкті. Оскільки метод `append()` повертає посилання на об'єкт StringBuilder з початковим текстом і доданим рядком, то можна з нього знову викликати метод `append()`, яким додається ще один рядок-константа `" років"`. Призначення методу `toString()` - перетворення до рядка даних будь-якого об'єкта (в даному випадку об'єкта класу StringBuilder). Метод `toString()` завжди неявно викликається в операторі `System.out.println()`, в результаті чого дані будь-якого типу і об'єкти будь-якого класу перетворюються до типу-рядка, необхідного для параметра методу `println()`.

Спільне використання оператора арифметичного додавання і оператора злиття рядків (конкатенації) може приводити до помилок виконання програми, наприклад, на перший погляд може здатись, що оператор

```
String s = "п'ять = " + 2 + 3;
```

присвоїть рядковій змінній `s` значення `"п'ять = 5"`. Однак при спробі роздрукувати цю змінну, на екран буде виведено - `п'ять = 23`. Чому це відбувається можна зрозуміти, проаналізувавши вищенаведений приклад. Тут знову ж неявно породжується об'єкт класу StringBuilder, ініційований рядком-літералом, а потім за допомогою методу `append()` спочатку приєднується рядкове представлення першого числа, а потім другого, після чого об'єкт класу StringBuilder переводиться в рядок методом `toString()`.

Для того, щоб перед перетворенням в рядок і додаванням до об'єкту StringBuilder було виконане додання цілих чисел, потрібно використовувати дужки:

```
String s = "п'ять = " + (2 + 3);
```

Об'єкти класу String не можуть змінюватися. Не можна ні вставити нові символи до вже існуючого рядока, ні поміняти в ньому одні символи на інші. І

додати один рядок в кінець іншого теж не можна. Тому компілятор Java перетворить операції, що виглядають, як модифікація об'єктів String, в операції з класом StringBuilder (або StringBuffer).

Метод `insert()` вставляє символи, зазначені як другий параметр, в певне місце в рядок у буфері, що задається першим його параметром (якщо параметр = 0, символи будуть вставлені перед першим символом рядка в буфері, значення параметра не повинно бути менше нуля і перевищувати довжину рядка в буфері, в іншому випадку викидається виключення StringIndexOutOfBoundsException). У черговому прикладі рядок `"Програмування"` вставляється між `"Мова "` і `"Java"`:

```
StringBuffer sb = new StringBuffer("Мова Java");
sb.insert(5, "програмування");
System.out.println(sb);
```

При запуску ця програма виводить наступний рядок:

```
Мова програмування Java
```

Перевизначення методу toString класу java.lang.Object

Нагадаємо, зо у Java всі класи неявно успадковані від класу java.lang.Object, який має метод `toString`. Цей матод повертає повне ім'я класу (включає ім'я пакета, до якого входить клас, і ім'я класу), після якого знаходиться символ `@` і безнаковое шістнадцяткове представлення хеш-коду об'єкта, пов'язане з місцезнаходженням його в пам'яті. Наприклад, наступний код:

```
Object o = new Object();
System.out.println(o);
```

виведе

```
java.lang.Object@7919298d
```

У документації з даного методу рекомендується, щоб у всіх дочірніх класах (нагадаємо, що всі без винятку класи успадковують клас java.lang.Object) метод `toString()` перевизначався з метою виведення на екран інформативного уявлення про об'єкт. У наведеному нижче прикладі в класі Point перевизначається успадкований від класу Object метод `toString()` своїм власним, що дозволяє йому виводити значення змінних об'єкта. Нагадаємо, що анотація `@Override` додатково вказує компілятору на перевизначення методу іншого класу.

```
class Point {
   int x, y;
   Point(int x, int y) {
     this.x = x;
     this.y = y;
   }
   @Override
   public String toString() {
     return "координата x = " + x + ", координата y = " + y;
   }
}
```

Тепер при виведенні об'єкта класу Point (з неявним використанням методу `toString()` в операторі `System.out.println()`) ми замість повного імені класу об'єкта і його хеш-коду побачимо координати точки на площині, що є більш інформативним для об'єкта в даному випадку:

```
class ToStringDemo {
   public static void main(String[] args) {
     Point p = new Point(10, 20);
     System.out.println(p);
   }
}
```

Нижче наведено результат, отриманий при запуску цього прикладу:

```
C:\>java ToStringDemo
координата x = 10, координата y = 20
```

Робота з символами рядків

Для того, щоб повернути одиночний символ з рядка, Ви можете послатися на символ по його індексу (послідовному номеру, починаючи з нуля) в рядку за допомогою методу класу String `charAt(int index)`. Метод класу StringBuilder/(StringBuffer) `setCharAt(int index, char ch)` дозволяє замінювати символ рядка за вказаним індексом на `ch`, а метод класу String `replace(char oldChar, char newChar)` - змінювати все входження символу `oldChar` на символ `newChar`. Відзначимо, що тільки метод `charAt(int index)` є і в класі String, і в класах StringBuffer/StringBuilder. Методи ж, що змінюють довжину рядків є тільки в класах StringBuffer/StringBuilder. Крім `setCharAt(int index, char ch)`, до них відносяться `append(char c)`, `deleteCharAt(int index)` і `insert(int offset, char c)`, `replace(int start, int end, String str)` і `reverse()`. Зверніть увагу, що параметри методу `replace(int start, int end, String str)` класівStringBuffer/StringBuilder відрізняються від параметрів методу `replace(char oldChar, char newChar)` класу String: В останньому випадку довжина рядка завжди незмінна.

Використання методів повернення та установки символів ілюструється на прикладі програми шифрування рядка `Привіт, як справи?`:

```
  String input = "Привіт, як справи?";
  System.out.println(input);
/*Створюємо об'єкти класу StringBuilder для вихідного
  та зашифрованого рядків*/
  String ins = input;
  StringBuilder outsb = new StringBuilder(input);
/*Логіка шифрування - символ в поточній позиції вихідного
  тексту стає символом в позиції на a++ більшою з урахуванням
  поділу по модулю, рівному довжині рядка*/
/*Визначає позицію першої літери вихідного тексту
  в шифрограмі*/
  int a = 1;
  int length = ins.length();
```

```
  for (int i = 0; i <length; i++) {
    /*Зчитування символу за індексом i*/
      char c = ins.charAt(i);
      int j = (i + a++) % length;
    /*Запис символу за індексом j*/
      j = (int) (i + a++) % length;
      outsb.setCharAt(j, c);
  }
  System.out.println("Шифр:" + outsb);
```

Результатом шифрування буде рядок `Прпвір, ак вприви?`

Якщо необхідно відразу витягти декілька символів рядка (або всі символи), можна скористатися методом класу String `getChars()`. У наведеному нижче фрагменті коду показано, як слід витягувати масив символів з об'єкта типу String:

```
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);
    }
}
```

Зверніть увагу - метод `getChars()` не включає в вихідний буфер символ з індексом `end`. Це видно з виведення нашого прикладу - рядок складається з 4 символів.

```
З: \> java getCharsDemo
demo
```

Для зручності роботи в String є ще один метод - `toCharArray()`, який повертає в вихідному масиві типу `char` весь рядок. Альтернативна форма того ж самого механізму дозволяє записати вміст рядка в масив типу `byte`, при цьому значення старших байтів в 16-бітних символах відкидаються. Відповідний метод називається `getBytes()`, і його параметри мають таке ж значення, що і параметри `getChars()`, але з єдиною різницею - в якості третього параметра треба використовувати масив типу `byte`.

```
  String ts = "AZaz Яя TeSt StRiNg";
  System.out.println(ts);
/*Отримуємо масив байтів з кодами символів рядка*/
  byte tsBytes[] = ts.getBytes();
  for (int i = 0; i < tsBytes.length; i++) {
      System.out.print(tsBytes[i] + "");
  }
```

В результаті роботи програми на екран буде виведено значення байтів, відповідних символам рядка:

```
65 90 97 122 32 -48 -81 -47 -113 32 84 101 83 116 32 83 116 82
105 78 103
```

Зверніть увагу, що значення байтів відповідають таблиці кодування UTF-8 і для російських букв вони виводяться двома байтами для кожного символу.

Перевірка рядків на рівність

Якщо ви хочете дізнатися, чи однакові два рядки, вам слід скористатися методом `equals()` класуString (перевизначення відповідного методу класу Object). Альтернативна форма цього методу називається `equalsIgnoreCase()`, при її використанні відмінність регістрів букв при порівнянні до уваги не береться. Нижче наведено приклад, який ілюструє використання обох методів:

```
  String s1 = "Hello";
/*Створюємо новий об'єкт з копією рядка s1*/
  String s2 = new String(s1); // те ж саме String s2 = s1
  String s3 = "Рядок";
  String s4 = "Рядок";
  String s5 = "рядок";
/*Метод public boolean equals(Object anObject) повертає true,
  якщо поточний об'єкт і anObject - це одне і те ж (розташовані
  за однією і тією ж адресою - this == anObject) або, якщо усі
  символи поточного об'єкта збігаються з символами і їх порядком
  слідування в anObject*/
  System.out.println(s1 + "equals" + s2 + "->"
                  + S1.equals (s2)); // true
   System.out.println(s1 + "==" + s2 + ", ->"
                  + (S1 == s2)); // false
/*Оптимізуючий компілятор Java аналізує наявні в коді
  програми літерні константи, і для однакових за змістом
  констант використовує одні й ті ж об'єкти-рядки - так звані
  інтерновані рядки*/
  System.out.println(s3 + "==" + s4 + ", ->"
                 + (S3 == s4)); // true
  System.out.println(s3 + "equalsIgnoreCase" + s5 + "->"
                  + S3.equalsIgnoreCase (s5)); // true
  }
}
```

Результат запуску цього прикладу:

```
Hello equals Hello -> true
Hello == Hello, -> false
Рядок == рядок, -> true
Рядок equalsIgnoreCase рядок -> true
```

Програма демонструє, що метод `equals()` і оператор == виконують різні перевірки. Якщо оператор == порівнює дві змінні-посилання на об'єкти і перевіряє, чи вказують вони на різні об'єкти або на один і той же, то метод `equals()` додатково порівнює символи всередині рядків.

В класі String реалізована група сервісних методів, які є спеціалізованими версіями методу `equals()`. Метод `regionMatches()` використовується для

порівняння підрядка в заданій стрічці з підрядком в рядку-параметрі. Метод `startsWith()` перевіряє, чи починається дана подстрока фрагментом, переданим методу в якості параметра. Метод `endsWith()` перевіряє, чи співпадає з параметром кінець рядка.

Найчастіше буває недостатньо просто знати, чи є два рядки ідентичними. Для додатків, в яких потрібна сортування, потрібно знати, який з двох рядків менший. Для відповіді на це питання потрібно скористатися методом `compareTo()` класу String (клас String реалізує інтерфейс Comparable). Якщо ціле значення, повернене методом, негативно, то рядок, з якого був викликаний метод, менше рядка-параметра, якщо позитивно - більше. Якщо ж метод `compareTo()` повернув значення `0`, рядки ідентичні. Нижче наведена програма, в якій виконується бульбашкове сортування масиву рядків, а для порівняння рядків використовується метод `compareTo()`. Ця програма видає відсортований в алфавітному порядку список рядків-слів.

```
String arr [] = {"Мова", "програмування", "Ява", "-",
"сучасний", "об'єктно", "орієнтований", "мова програмування"};
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];
            arr[j] = arr[i];
            arr[i] = t;
        }
    }
    System.out.println(arr[j]);
}
```

Пошук рядків

В клас String включена підтримка пошуку певного символу або підрядка, для цього в ньому є два методи - `indexOf()` і `lastIndexOf()`. Кожен з цих методів повертає індекс символу, який шукається, або індекс початку підрядка, який шукається. У будь-якому випадку, якщо пошук виявився невдалим методи повертають значення `-1`. Метод `indexOf()` обчислює індекс (починаючи з `0`) з початку рядка, а метод `lastIndexOf()` шукає індекс останнього входження символу в рядок. У черговому прикладі показано, як користуватися різними варіантами методів пошуку.

```
/*Демонстрація пошуку символів і подстрок в рядку*/
  String s = "Мова програмування Java - "
           + "сучасна об'єктно-орієнтована "
           + "мова прорграмування";
  System.out.println(s);
  System.out.println("indexOf(M) = " + s.indexOf('M'));
  System.out.println("indexOf(Z) = " + s.indexOf('Z'));
  System.out.println("indexOf(я) = " + s.indexOf('я'));
  System.out.println("lastIndexOf(я)= " + s.lastIndexOf('я'));
  System.out.println("indexOf(уван)= " + s.indexOf("уван"));
  System.out.println("lastIndexOf(уван) = "
```

```
                    + s.lastIndexOf("уван"));
  System.out.println("indexOf(я, 21) = " + s.indexOf('я', 21));
  System.out.println("lastIndexOf(я, 21) = "
                    + s.lastIndexOf('я', 21));
  System.out.println("indexOf(уван, 50) = "
                    + s.indexOf("уван", 50));
  System.out.println("lastIndexOf(уван, 50) = "
                    + s.lastIndexOf("уван", 50));
```

Нижче наведено результат роботи цієї програми. Зверніть увагу на те, що індекси в рядках починаються з нуля.

```
Мова програмування Java - сучасна об'єктно-↲
орієнтована мова програмування
indexOf(M) = 0
indexOf(Z) = -1
indexOf(я) = 17
lastIndexOf(я)= 72
indexOf(уван)= 12
lastIndexOf(уван) = 67
indexOf(я, 21) = 72
lastIndexOf(я, 21) = 17
indexOf(уван, 50) = 67
lastIndexOf(уван, 50) = 12
```

Модифікація рядків

Оскільки об'єкти класу String не змінюваний, щоразу, коли вам потрібно модифікувати рядок, доведеться або копіювати об'єкт класу String в об'єкт StringBuilder/StringBuffer, або використовувати один з описуваних нижче методів класу String, які створюють нову копію рядка, вносячи в неї необїідні зміни.

Ви можете отримати підрядок з об'єкта String, використовуючи метод `substring()`. Цей метод створює нову копію символів з діапазону індексів оригінального рядка, які вказані при виклику. Можна вказати тільки індекс першого символу потрібного підрядка - тоді будуть скопійовані всі символи, починаючи з зазначеного і до кінця рядка. Також можна вказати і початковий, і кінцевий індекси - при цьому в новий рядок будуть скопійовані всі символи, починаючи з першого зазначеного, і до (але не включаючи його) символу, заданого кінцевим індексом.

```
"Hello World".substring (6) -> "World"
"Hello World".substring (3,8) -> "lo Wo"
```

Злиття, або конкатенація рядків виконується за допомогою методу `concat()`. Цей метод створює новий об'єкт String, копіюючи в нього вміст вихідного рядка і додаючи в кінець рядок, зазначений в параметрі методу.

```
"Hello".concat(" World") -> "Hello World"
```

Методу `replace()` в якості параметрів задаються два символи. Всі символи, що збігаються з першим, замінюються в новій копії рядка на другий символ.

```
"Hello".replace('l', 'w') -> "Hewwo"
```

Пара методів `toLowerCase()`, `toUpperCase()` перетворює все символи початкового рядка в нижній і верхній регістр, відповідно:

```
"Hello".toLowerCase() -> "hello"
"Hello".toUpperCase() -> "HELLO"
```

І, нарешті, метод `trim()` видаляє з рядка всі початкові та кінцеві пробіли.

```
"    Hello World    ".trim () -> "Hello World"
```

Часто зручним є використання методу `split(String regex)`, що повертає масив рядків, на які розбивається вихідний рядок, з якого викликається цей метод, при цьому роздільником служить рядок `regex`. Наприклад, наступна програма

```
String s = "Переселення";
String[] substrings = s.split("е");
for(String substr: substrings) {
    System.out.println(substr);
}
```

роздрукує підрядки, розділені буквою "е", що є елементами масиву `substrings[]`:

```
П р с л ння
```

Часто буває потрібно розбити рядок на слова, при цьому роздільником між словами може бути пробіл, символ табуляції, переведення рядка, вертикальна табуляція, прогон сторінки і повернення каретки - так звані *пробільні символи (whitespace symbols)*. Для цього випадку існує рядок регулярного виразу `"\\s+"`, знак `+` вказує, що роздільником можуть бути кілька поспіль символів пробілу. Приклад програми, що розбиває рядок на слова, виглядає наступним чином:

```
String s = "Привіт мій світ!";
String[] words = str.split("\\s+");
for (String word: words) {
    System.out.println(word);
}
```

Додаткову інформацію щодо регулярних виразів, які можна використовувати в якості роздільників, можна прочитати в Регулярні вирази в Java.

Конструктори класів StringBuffer / StringBuilder

Об'єкт StringBuffer/StringBuilder можна створити без параметрів, при цьому в ньому буде зарезервовано місце для розміщення 16 символів без можливості зміни довжини рядка. Існу мождивість передачі конструктору цілого числа, для того, щоб явно задати необхідний розмір буфера. І, нарешті, є можливість передачі конструктору рядка, при цьому він буде скопійованаий в об'єкт і додатково в ньому буде зарезервовано місце ще для 16 символів. Поточну довжину об'єкта StringBuffer/StringBuilder можна визначити, викликавши метод `length()`, а для

визначення всього місця, зарезервованого під рядок в об'єкті StringBuffer / StringBuilder, потрібно скористатися методом `capacity()`. Нижче наведено приклад, який це пояснює:

```
StringBuffer sb = new StringBuffer("Hello");
System.out.println("buffer = " + sb);
System.out.println("length = " + sb.length());
System.out.println("capacity = " + sb.capacity());
```

Виведння цієї програми демонструє, що в екземплярі `sb` об'єкта StringBuffer для маніпуляцій з рядком зарезервовано додаткове місце.

```
C:\>java StringBufferDemo
buffer = Hello
length = 5
capacity = 21
```

Якщо після створення об'єкта StringBuffer необхідно зарезервувати в ньому місце для певної кількості символів, для установки розміру буфера моєна скористатися методом `ensureCapacity()`. Це буває корисно, коли Ви заздалегідь знаєте, що доведеться додавати до буферу багато невеликих рядків.

Якщо Вам знадобиться в явному вигляді встановити довжину рядка в буфері, скористайтеся методом `setLength()`. Якщо Ви задасте значення, більше, ніж довжина рядка, що міститься в об'єкті, цей метод заповнить кінець нового, розширеного рядка символами з кодом нуль.

Перетворення даних методом valueOf

Якщо Ви маєте справу з будь-яким типом даних і хочете вивести значення цього типу у вигляді рядка, спочатку доведеться перетворити це значення в текстовий рядок. Для цього використовується метод `valueOf()` класу String. Цей статичний метод визначений для будь-якого існуючого в Java типу даних (всі ці методи суміщені, тобто використовують одне і те ж ім'я). Завдяки цьому не складає труднощів перетворити в рядок значення будь-якого типу.

Робота з класами-обгортками

Примітивні типи в Java, на відміну від об'єктів, використовуються з міркувань продуктивності. Застосування об'єктів для цих значень додає небажані накладні витрати, навіть в разі найпростіших обчислень. Незважаючи на те, що примітивні типи забезпечують виграш в продуктивності, бувають випадки, коли необхідно об'єктне уявлення. Наприклад, не можна передати в метод примітивний тип за посиланням. Крім того, багато які з стандартних структур даних, реалізованих в Java, оперують з об'єктами, тому неможливо використовувати ці структури даних для збереження примітивних типів. Щоб впоратися з такими (і подібними) ситуаціями, Java пропонує *класи-обгортки (wrappers)* примітивних типів, які представляють собою класи, що вміщують примітивний тип в об'єкт.

Обгортки для примітивних типів - це розміщені в пакеті java.lang класи Byte, Short, Integer, Long, Float, Double, Character і Boolean. Ці класи надають методи, що дозволяє в повній мірі інтегрувати примітивні типи в ієрархію об'єктних типів Java.

Слід зазначити, що всі класи оголошені з модифікатором `final`, який забороняє перевизначення їх членів, і реалізують інтерфейс Comparable (можуть порівнюватися і сортуватися). А чисельні типи успадковують абстрактний клас java.lang.Number з абстрактними методами перетворення одного числового типу в інший. Крім загальних для всіх обгорток методів, кожен клас-обгортка має корисні методи для конкретного типу, який він обертає. Наприклад, в класі Integer існують корисні константи (розмір в байтах, бітах, максимально-можливе і мінімально-можливе значення) і методи перетворення в рядок бітів, в рядок символів шестнадцатеричної і вісімкової систем обчислення, перевертання і зсуву рядка бітів числа, декодування зі строкового подання числа в десяткову, шістнадцяткову і вісімкову системи числення і інші. Часто використовується метод парсинга строкового подання цілого числа в десятковій системі числення `parseInt(String s)`. У нижче наведеному фрагменті коду демонструється використання констант і методів класу Integer:

```
System.out.println("Integer constatnts:\n"
                 + "Integer size in bytes: " + Integer.BYTES
                 + ", Integer size in bits: " + Integer.SIZE
                 + ", Integer primritive type is: "
                 + Integer.TYPE
                 + "\nInteger MIN_VALUE = " + Integer.MIN_VALUE
                 + ", Integer.MAX_VALUE = "
                 + Integer.MAX_VALUE);
Integer intNum = new Integer(5);
System.out.println(intNum + " = "
                            + Integer.toBinaryString(intNum));
System.out.println("Reverse value of " + intNum + " is "
           + Integer.reverse(intNum) + " = "
           + Integer.toBinaryString(Integer.reverse(intNum)));
int bitsToRotate = 2;
System.out.println("Rotate left " + intNum + " by "
           + BitsToRotate + " bits is: "
           + Integer.toBinaryString(Integer.rotateLeft(intNum,
                                bitsToRotate)) + " = "
           + Integer.rotateLeft(intNum, bitsToRotate));
System.out.println("Value of " + intNum + " is "
                                  +Integer.decode("0x5"));
String s1 = "12";
String s2 = "21";
System.out.println("s1 + s2 = " + (Integer.parseInt(s1)
                            + Integer.parseInt(s2)));
```

Виведення на екран результату роботи програми:

```
Integer constatnts:
Integer size in bytes: 4, Integer size in bits: 32, Integer
primritive type is: int
Integer MIN_VALUE = -2147483648, Integer.MAX_VALUE = 2147483647
5 = 101
```

```
Reverse value of 5 is -1610612736 =
10100000000000000000000000000000
Rotate left 5 by 2 bits is: 10100 = 20
Value of 5 is 5
s1 + s2 = 33
```

Клас-обгортка Character містить корисні методи для роботи з символами. Наведений нижче фрагмент коду ілюструє їх використання:

```
String s = "The Character instatnce size is " + Character.SIZE
         + " bits.";
char[] letters = s.toCharArray();
int letterCnt = 0;
int digitCnt = 0;
int upperCaseLetterCnt = 0;
int lowererCaseLetterCnt = 0;
int whitespaceCnt = 0;
for (char letter: letters) {
    if (Character.isAlphabetic(letter)) {
        letterCnt++;
    }
    if (Character.isDigit(letter)) {
        digitCnt++;
    }
    if (Character.isWhitespace(letter)) {
        whitespaceCnt++;
    }
    if (Character.isUpperCase(letter)) {
        upperCaseLetterCnt++;
    }
    if (Character.isLowerCase(letter)) {
        lowererCaseLetterCnt++;
    }
}
System.out.println("The string: \" "+ s +" \ " has:"
                 + "\nTotal symbols: " + letters.length
                 + "\nletters: " + letterCnt + ", digits:"
                 + DigitCnt + ", whitespaces:" + whitespaceCnt
           + "\nupper case letters:" + upperCaseLetterCnt
           + ", Lower case letters:" + lowererCaseLetterCnt);
System.out.println("The string: \" "+ s + " \ "in reverse case"
                 + "is:");
for (int i = 0; i <letters.length; i++) {
    if (Character.isUpperCase(letters[i])) {
        System.out.print(Character.toLowerCase(letters[i]));
    } else {
        System.out.print(Character.toUpperCase(letters[i]));
    }
}
```

Програма виводить наступний результат:

```
The string: "The Character instatnce size is 16 bits." has:
```

```
Total symbols: 40
letters: 31, digits: 2, whitespaces: 6
upper case letters: 2, lower case letters: 29
The string: "The Character instatnce size is 16 bits."↲
in reverse case is:
tHE cHARACTER INSTATNCE SIZE IS 16 BITS.
```

В Java 5 були введені технології автопакування-автораспакування, що усувають необхідність ручного перетворення типів для класів-обгорток і примітивних типів. *Автопакування (Autboxing)* - автоматичне перетворення з примітивного типу в об'єкт класу-обгортки, *автораспакування (Unboxing)* - автоматичне перетворення з об'єкта класу обгортки в примітивний тип. Для того щоб оцінити переваги цієї технології, порівняйте два фрагмента коду - без використання і з використанням автопакування-автораспакування:

Код, необхідний при використанні версій JDK до JDK 5:

```
/*Перетворення від int до Integer – ручне пакування*/
Integer intObj = Integer.valueOf(5);
/*Перетворення від Integer до int – ручне розпакування*/
int x1 = intObj.intValue();
int x2 = intObj.intValue() + 7;
```

Код, необхідний при використанні версій JDK 5 і вище:

```
/*Перетворення від int до Integer - автоупаковка*/
Integer intObj = 5;
/*Перетворення від Integer до int - автораспаковка*/
int x1 = intObj;
int x2 = intObj + 7;
```

Видно, що в другому випадку код простіше.

2. Завдання

1. Розробіть програму, що обробляє рядки і підрядки, відповідно до завдання в наведеній нижче таблиці (номер варіанта вибирайте за формулою V = (№ mod 16) +1, де № - Ваш порядковий номер в журналі академгрупи).

V	Завдання	V	Завдання
1	Для довільного цілого числа N виведіть на екран рядок довжиною N символів, який складається з символів Q і Ж, які повторюються, починаючи з Q	9	Для довільнго рядка-речення виконайте заміну місцями його слів: 1-е - з останнім, 2-е - з передостаннім і т.д. Якщо кількість слів непарна, то середнє слово не змінюється
2	Для довільних цілих чисел N1 і N2 і рядків S1 і S2 виведіть на екран новий рядок, що містить N1 перших символів рядка S1 і N2 останніх символів рядка S2	10	Для довільного рядка-речення виведіть на екран слова цього рядка в зворотному порядку (передбачити можливість поділу слів більш ніж одним пропуском)

V	Завдання	V	Завдання
3	Для довільних рядків S1 і S2 виведіть на екран новий рядок, рівний рядку S1, з якого вилучені всі підрядки, що збігаються з S2. Якщо таких підрядків немає, то вивести S1 без змін	11	Для довільних рядків S1 і S2 виведіть на екран інформацію, чи міститься рядок S2 в рядку S1. У разі якщо міститься, виведіть номер позиції, починаючи з якої S2 міститься в S1
4	Для довільних рядків S1, S2 і S3 виведіть на екран новий рядок, що збігається з S1, але в якому всі входження рядка S2 замінені на S3	12	Для довільного рядка-речення виведіть його слова в порядку неспадання їх довжин
5	Для рядка, що представляє собою повний шлях до файлу, виведіть на екран виділену з рядка назву останнього каталогу (без символів "\"). Якщо файл міститься в кореневому каталозі, то виведіть символ "\"	13	Для довільних рядків-речень S1 і S2 виведіть на екран рядок, який по черзі містить слова з S1 і S2, перше слово - з S1. Слова з порядковим номером, що перевищує кількість слів коротшого рядка, виводити без зміни порядку
6	Для довільних рядків S1 і S2 і довільного символу виведіть на екран рядок S1, у якому перед кожним символом, що збігається з введеним, вставлений рядок S2	14	Для довільного рядка-речення виведіть на екран слова, які починаються і закінчуються однією і тією ж буквою
7	Для довільного рядка виведіть на екран підрядок, розташований між першою і другою точками вихідного рядка. Якщо в рядку менше двох точок, то вивести весь вихідний рядок	15	Для довільного рядка S і цілого числа N, більшого довжини рядка S, виведіть на екран рядок довжиною N, що містить повторювані рядки S
8	Для довільного рядка-речення виведіть на екран слова цього рядка, що містять дві і більше літери I в довільному регістрі	16	Для довільного рядка-речення виведіть на екран такий же рядок, але кожне слово якого має починатися з великої літери

2. Розробіть програму, що обробляє символи, відповідно до завдання в наведеній нижче таблиці (Номер варіанта вибирайте за формулою V = (№ mod 16) +1, де № - Ваш порядковий номер в журналі академгрупах). Введення даних в програму організовуйте через аргументи командного рядка.

V	завдання	V	завдання
1	Для довільного рядка виведіть на екран кількість великих букв, маленьких букв і цифр, які містяться у цьому рядку	9	Для довільного рядка S і цілого числа N> 0 виведіть на екран рядок, у якому видалений кожен N-ий символ
2	Для довільного речення обрахувати кількість входжень у нього кожної з літер	10	Для довільного рядка виведіть на екран рядок, у якому для кожного слова вихідного рядка видалені усі наступні входження першої літери цього слова
3	Дано два слова. Вивести тільки ті букви, які зустрічаються в обох словах лише один раз	11	Для довільного рядка S і цілого числа N > 0 виконайте шифрування рядка, збільшивши на N значення байта, відповідного символів рядка. Виведіть зашифровану рядок на екран

V	завдання	V	завдання
4	Дано три довільних слова. Виведіть тільки ті букви для кожного зі слів, які не повторюються у інших словах	12	Для довільного рядка S і цілого числа N > 0 виведіть на екран новий рядок, рівний за довжиною S. При цьому якщо довжина рядка S більше N, то відкинути перші символи, якщо довжина рядка S менше N, то на початку рядка додати символи точки
5	Для довільного рядка-речення виведіть на екран найдовше слово в рядку (якщо таких слів кілька, то вивести перше з них)	13	Дано довільне слово. Поміняйте місцями його букви, які стоять на парних і непарних позиціях: 1 з 2, 3 з 4 і т.і. Якщо кількість букв у слові непарна, останню літеру залиште незмінною
6	Для довільного рядка виведіть на екран кількість голосних букв, що містяться у рядку	14	Дано рядок-речення. Вивести на екран однакові сусідні символи
7	Для довільного рядка-речення із зайвими пробілами між словами виведіть на екран такий же рядок тільки з одним пробілом між словами і з відсутністю пробілів перед першим словом і після останнього	15	Для довільного рядка виконайте його шифрування, помістивши спочатку всі символи, розташовані на парних позиціях, а потім, в зворотному порядку, всі символи, розташовані на непарних позиціях (наприклад, рядок "Програма" перетвориться на "ргаамроП"). Виведіть зашифрований рядок на екран
8	Для довільної рядка, що містить відкриваючу та закриваючу круглу дужку, виведіть на екран число 0, якщо для кожної відкриваючої дужки є одна закриваюча. В іншому випадку виведіть номер позиції, у якій розташована перша дужка без пари	16	Для довільного рядка-речення і цілого числа k (0 < k < 10) виконайте шифрування рядка шляхом циклічної заміни кожної букви на букву того ж регістра, розташовану в алфавіті на k-й позиції після літери, шо шифрується (наприклад, для k = 2 "А" перейде в "В", "а" →"в", "Б" → "Г", "я" → "б" і т.і.). Букву "Ґ" та апостроф в алфавіті не враховувати, розділові знаки і пропуски не змінювати. Виведіть на екран зашифрований рядок

3. Виконайте перевизначення методу `toString()` для розробленого Вами в лабораторній роботі "Принципи об'єктно-орієнтованого підходу до програмування" класу так, щоб при виведенні на екран об'єктів класу роздруковувалися їх властивості, визначенні у завданні.
4. Розробіть програму, використовуючи класи-огортки, відповідно до завдання у наведеній нижче таблиці (Номер варіанта обирайте за формулою V = (№ mod 16) +1, де № - Ваш порядковий номер у журналі академгрупи). Введення даних у програму організовуйте через аргументи командного рядка, по-можливості продемонструйте автоупаковку-автораспаковку примітивних типів.

V	Завдання	V	Завдання
1	Розробіть метод перетворення довільного цілого числа в його двійкове подання з виведенням усіх 32 біт числа	9	Організуйте випадкове генерування позитивного 2-байтового числа з виведенням на екран його значення у шістнадцятковій системі і подальшим перетворенням і виведенням його десяткового значення
2	Організуйте введення символьного вигляду цілого числа. Виведіть на екран набір символів, що містить цифри цього числа у зворотному порядку	10	Для довільного цілого числа виведіть на екран його вісімкове і шістнадцяткове значення
3	Для довільного цілого числа виведіть на екран кількість одиничних бітів у його двійковому вигляді і в двійковому вигляді його бітової інверсії	11	Для довільного `long` числа виведіть на екран його знак
4	Організуйте введення рядка, що відповідає двійковому запису довільного байта, і виведіть рядок, який зображує десятковий запис цього байта	12	Для довільного дійсного числа `double` виведіть на екран двійкове значення його знака, експоненти і мантиси (з неявним одиничним бітом) відповідно до специфікації IEEE-754
5	Організуйте введення довільного рядка і виведіть на екран інформацію, чи є він записом цілого числа, чи є він записом дійсного числа або він не є записом числа	13	Організуйте введення рядка, що представляє дійсне число, і цілого числа N > 0. Виведіть на екран набір символів, що відповідають першим N цифрам дробової частини цього дійсного числа (без округлення)
6	Для довільного цілого числа вкажіть номер позиції його максимального і мінімального одиничного біта	14	Організуйте введення рядка. Виведіть на екран шіснадцяткові коди його першого та останнього символів
7	Для довільного цілого числа виведіть його двійкове і десяткове значення після циклічного зсуву на 1, 2, 4 і 8 біт вліво	15	Для довільного `long` числа вкажіть номер позиції його максимального і мінімального одиничного біта
8	Для довільного дійсного числа `float` виведіть на екран двійкові значення його знака, експоненти і мантиси (з неявним одиничним бітом) відповідно до специфікації IEEE-754	16	Для довільного цілого числа виведіть його двійкове і десяткове значення після циклічного зсуву на 1, 2, 4 і 8 біт вправо

5. Додайте до вихідного коду документовані коментарі та надайте вихідний код розроблених методів, а також скріншоти з результатами роботи програм до звіту.

3. Контрольні питання

1. Який клас Java являє рядок? Яким чином у ньому організована незмінюваність рядку?
2. Які класи Java представляють змінювані рядки? Яким чином у них організована змінність рядків? Назвіть відмінності цих класів.
3. Опишіть конструктори класу String.

4. Наведіть приклад завдання рядка з використанням масиву, у якому елементи задані як символи і як коди символів таблиці *UTF-8*.
5. Наведіть приклад завдання рядки з використанням масиву байтів із зазначенням назви кодування символів, за допомогою якої повинні інтерпретуватися байти.
6. За допомогою якого методу рядка можна з'ясувати кількість символів, що входять до нього?
7. Як можна виконати конкатенацію (об'єднання) декількох рядків? Який метод класів StringBuffer / StringBuilder її здійснює?
8. Наведіть приклад використання методу `insert` класів StringBuffer / StringBuilder.
9. Якщо створити об'єкт довільного класу `obj` і потім вивести його на екран оператором `System.out.println(obj)`, що буде виведено? Як забезпечити виведення атрибутів об'єкта зазначеним оператором?
10. Вкажіть, які з перерахованих методів: `charAt`, `setCharAt`, `replace`, `append`, `deleteCharAt`, `insert` містяться у класі String, а які у StringBuffer / StringBuilder. Поясніть відповідь.
11. Наведіть приклад використання методу, що витягує з рядка в символьний масив кілька поспіль символів і методу, що витягує всі символи рядка в символьний масив.
12. Наведіть приклад використання методу, що витягує всі символи рядка в байтовий масив.
13. За допомогою яких методів можна дізнатися, чи однакові рядки? Чому рівність рядків не можна виконувати аналогічно перевірці рівності значень примітивних типів оператором ==? Назвіть методи, що дозволяють порівнювати підрядок у заданому рядку із підрядком-параметром.
14. Опишіть алгоритм роботи методу `compareTo()` у класі String.
15. Опишіть використання методів `indexOf()` і `lastIndexOf()` для пошуку символів у рядках і підрядках.
16. Опишіть призначення наступних методів класу `String`: `substring`, `concat`, `replace`, `trim`, `toLowerCase`, `toUpperCase`, `split`.
17. Як можна регулювати розмір буфера для введення символів рядка в об'єкти класів StringBuffer / StringBuilder.
18. Перерахуйте методи класів StringBuffer / StringBuilder для роботи з символами, що дозволяють витягати символ з певної позиції, додавати символ у певну позицію, додавати символ у кінець рядка, видаляти символ з певної позиції, замінювати символ у певній позиції.
19. За допомогою якого методу можна перетворити до рядка числове значення.
20. Опишіть переваги використання примітивних та об'єктних типів.
21. Дайте визначення класам-огорткам і перерахуйте їх.
22. Який інтерфейс реалізують всі класи огортки? Який абстрактний клас успадковують огортки числових типів?

23. Назвіть відомі Вам методи класів-огорток, що дозволяють перетворювати значення-рядки до значень примітивних типів і навпаки, значення примітивних типів – до рядків.
24. Назвіть відомі Вам константи і методи класу java.lang.Integer, що працюють з бітами.
25. Назвіть відомі Вам методи класу java.lang.Character, що працюють з рядками і символами.
26. Поясніть, як працює технологія автоупакуваня-авторозпакування.

4. Література

1. Васильєв О. Програмування мовою Java. Тернопіль: Навчальна книга - Богдан, 2020. 696 с.
2. Horstmann C. S. Core Java, Volume I: Fundamentals. 12-th Ed. "Addison-Wesley", 2022. 1197 p.
3. Learning the Java Language. The Java™ Tutorials. Oracle Java documentation site. URL: https://docs.oracle.com/javase/tutorial/java/TOC.html (дата звернення 08.08.2023).
4. IntelliJ IDEA IDE URL: https://www.jetbrains.com/idea/ (дата звернення 08.18.2023).
3. Класи-огортки для примітивних типів. Блог Pro Java -http://pr0java.blogspot.com/2015/05/blog-post_4.html
4. Регулярні вирази в Java. Стаття блогу Quizful. - http://www.quizful.net/post/Java-RegExp