Міністерство освіти і науки України
Сумський державний університет

4804 МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт
на тему «Робота з рядками в Java»
із дисциплін
«**Програмування мобільних комп'ютерних систем**»,
«**Прикладне програмування в телекомунікаційних системах**»,
«**Програмування мобільних пристроїв телекомунікацій**»
для студентів спеціальностей
171 «*Електроніка*»,
172 «*Телекомунікації та радіотехніка*»
денної форми навчання

Суми
Сумський державний університет
2020

Методичні вказівки до виконання лабораторних робіт на тему «Робота з рядками в Java» із дисциплін «Програмування мобільних комп'ютерних систем», «Прикладне програмування в телекомунікаційних системах», «Програмування мобільних пристроїв телекомунікацій» / укладач О.Є. Горячев – Суми: Сумський державний університет, 2020. – 33 с.

Кафедра електроніки і комп'ютерної техніки

1. Мета роботи

Метою роботи є набуття навичок програмування з використанням рядків мови Java.

2. Склад робочого місця.

IBM-сумісний персональний комп'ютер (ПК).

Програмне забезпечення: операційна система Windows, Java 2 SDK версії 6.0 і вище, пакет NetBeans IDE 6.1 і вище.

3 Короткі теоретичні відомості

3.1. Робота з рядками (класи String и StringBuffer)

Для зберігання та оброблення рядків у Java є два класи: **String** для незмінних рядків і **StringBuffer** – для рядків, які можуть змінюватися. Обидва класи розширюють клас **Object**. Вони розміщені в пакеті **java.lang**, тому їх не потрібно підключати за допомогою оператора import.

Рядкові літерали в Java, як і в C, розміщують в подвійних апострофах, наприклад, , **`"abc"`** задає строковий літерал abc.

Якщо всередині строкового літерала необхідно задати символ апострофа, він задається за допомогою символів \",наприклад, **`"it\"s"`** задає строковий літерал it"s.

3.1.1. Створення та ініціалізація об'єкта класу String

Ініціалізація об'єкта класу String може виконуватися як за допомогою оператора присвоювання змінної класу String строкової змінної або строкового літерала, наприклад:

```
String str = "Строка 1";
```

так і під час створення об'єкта за допомогою оператора new з використанням одного з конструкторів класу String, поданих у таблиці 1.

Єдина операція, яку можна використовувати для рядків, є операція зчеплення (конкатенація) двох або більше рядків – "+".

Довжина рядка може бути визначена за допомогою методу

```
public int length().
```

Рядки класу String можна змінювати, але під час кожної зміни довжини рядка створюється новий екземпляр рядка.

3.1.2. Створення та ініціалізація об'єкта класу StringBuffer

Клас StringBuffer схожий на клас String, але рядки, створені за допомогою цього класу можна модифікувати, тобто їх вміст і довжина можуть змінюватися. Під час зміни рядка класу StringBuffer програма не створює новий строковий об'єкт, а працює безпосередньо з вихідної рядком, тобто всі методи оперують безпосередньо з буфером, що містить рядок. Тому клас StringBuffer зазвичай використовується, коли рядок доводиться часто модифікувати зі зміною її довжини.

Таблиця 1. Конструктори класу String

Конструктор	Об'єкт, який він створює
`String()`	Порожний рядок
`String (String` *рядок*`)`	Новий рядок, який є копією *рядку*
`String(char[]` *масив*`)`	Рядок, створений з елементів *масиву*
`String(char[]` *масив*`, int` *початковий-індекс*`,, int` *довжина*`)`	Рядок, створюваний із фрагменту *масиву* символів, що починається з позиції *початкового-індексу* і заданої довжини
`String(byte[]` *масив*`)`	Рядок, створюваний із *масиву* байтів, з використанням кодування на даному комп'ютері за замовчуванням
`String(byte[]` *масив*`, int` *початковий-індекс*`, int` *довжина*`)`	Рядок, створюваний із фрагменту *масиву* байтів, що починається з позиції *початковий-індекс* і заданої *довжини*

Розміщення рядків в об'єкті **StringBuffer** виконується так. Для об'єкта **StringBuffer** задається розмір або ємність (capacity) буферної пам'яті для рядка. Рядок символів в об'єкті **StringBuffer**, характеризується своєю довжиною, яка може бути меншою або дорівнює ємності буфера. Якщо довжина рядка менша ніж ємність буфера, то решта довжини рядка заповнюється символом Unicode "**`\u0000`**". ". Якщо в результаті модифікації рядка його довжина стане більшою ніж ємність буфера, ємність буфера автоматично збільшується.

У класі StringBuffer є три конструктори, що дозволяють по-різному створювати об'єкти типу StringBuffer:

```
StringBuffer()
StringBuffer(int довжина)
StringBuffer(String строка)
```

Перший конструктор створює порожній об'єкт StringBuffer з ємністю буферної пам'яті в 16 символів, другий конструктор задає буфер з ємністю заданої довжини для зберігання рядка, а третій конструктор створює об'єкт StringBuffer з рядка з ємністю буфера, що дорівнює довжині рядка.

Методи визначення та встановлення характеристик рядка в класі StringBuffer наведені в таблиці 2.

Таблиця 2. Методи визначення та встановлення характеристик рядка в класі StringBuffer

Оголошення методу	Дія
`public int length()`	Повертає довжину рядка для об'єкта класу **`StringBuffer`**
`public int capacity()`	Повертає поточну ємність буферної області для об'єкта класу **`StringBuffer`**
`public void ensureCapacity(int` *ємність*`)`	Встановлює ємність буферної області для об'єкта класу **`StringBuffer`**
`public void setLength(int` *нова-довжина*`)`	Встановлює *нову-довжину* рядка. Якщо нова довжина більше старої, збільшуються довжини рядка і буфера, при цьому додаткові символи заповнюються нулями. Якщо нова довжина менше старої, символи в кінці рядка відкидаються, а розмір буфера не змінюється
`public String toString()`	Перетворює рядок **`StringBuffer`** у рядок **`String`**

3.1.3. Порівняння рядків

Оскільки в Java рядки є об'єктами, для порівняння рядків можна використовувати оператор "**==**" і метод

`public boolean equals (Object об'єкт),`

порівнює рядок, для якого викликається метод, з об'єктом. Результат буде true, лише якщо об'єкт є рядком і значення порівнюваних рядків рівні.

Використання оператора "**==**" для порівняння рядків може привести до неправильного результату, якщо порівнювані рядки різні об'єкти, тому більш доцільним є використання методу **equals()** (цей метод працює і для рядків String і для рядків StringBuffer).

Інші методи порівняння рядків, також повертають бульові значення, наведені в таблиці 3.

Метод **`int compareTo(String anotherString)`** – лексикографічно порівнює два рядки і повертає:

- 0, якщо рядки однакові по довжині і мають однакове значення,

- значення менше ніж 0, якщо в першій позиції, в якій символи рядків не однакові, код символу в першому рядку менше коду символу у другому рядку або довжина першого рядка менше ніж довжина другого рядка всі символи першого рядка рівні символів в тих же позиціях другого рядка.

- значення, більше ніж 0, якщо в першій позиції, в якій символи рядків не рівні код символу в першому рядку більше коду символу в другому рядку або довжина першого рядка більше довжини другого рядка, повертається.

Таблиця 3. Методи порівняння рядків класу String

Метод	Повертає `true`, коли
`equalsIgnoreCase(String` *рядок1*`)`	Рядок дорівнює *рядку1* незалежно від регістру символів
`startsWith(String` *префікс*`)`	Рядок починається з *префіксу*
`startsWith(String` *префікс*`,` `int` *початковий-індекс*`)`	Підрядок рядку *префікс*, починаючи з позиції *початковий-індекс*
`endsWith(String` *суфікс*`)`	Рядок закінчується рядком *суфікс*
`regionMatches(int` *початковий-індекс*`,` `String` *рядок1*`,` `int` *початковий-індекс1*`,` `int` *довжина*`)`	Підрядок в рядку, починаючи з позиції зі зміщенням *початковий-індекс*, відповідає підрядку *рядку1*, починаючи зі зміщення *початковий-індекс1*, і заданої *довжини*
`regionMatches(boolean` *без-обліку-регістра*`,` `int` *початковий-індекс*`,` `String` *рядок1*`,` `int` *початковий-індекс1*`,` `int` *довжина*`)`	Те саме, що і попередній метод, але ігнорує регістр символів, коли параметр *без-обліку-регістра* дорівнює `true`.

3.1.4. Пошук у рядках

Для пошуку символів або послідовностей символів (лише в рядках класу String) використовуються такі перевантажені методи **`indexOf()`,** наведені в таблиці 4.

Таблиця 4. Методи пошуку класу String

Оголошення методу	Значення, що повертається
`int indexOf(int` *символ*`)`	Перша позиція в рядку, в якій зустрічається *символ*
`public int indexOf(int` *символ*`,` `int` *початковий-індекс*`)`	Перша позиція в рядку, починаючи з позиції *початковий-індекс*, в якій зустрічається *символ*
`int indexOf(String` *рядок*`)`	Перша позиція в рядку, в якій зустрічається *рядок*
`public int indexOf(String` *строка*`,` `int` *початковий-індекс*`)`	Перша позиція в рядку, починаючи з позиції *початковий-індекс*, в якій зустрічається *рядок*

Для кожного методу **`indexOf()`** є відповідний метод **`lastIndexOf()`**, який починає пошук символу або рядка не з початку, а з кінця рядка. Якщо символ або рядок не знайдені в рядку, в якій проводиться пошук, методи **`indexOf()`** і **`lastIndexOf()`** повертають значення -1.

3.1.5. Витяг символів і підрядків із рядка

Витяг символів і підрядків з рядка, а також створення нових рядків на основі існуючих рядків виконується за допомогою методів, наведених у таблиці 5.

Таблиця 5. Методи маніпуляцій з рядками класу String

Метод	Значення, що повертається
`char charAt(int` *індекс***`)`**	Символ рядка в позиції *індекс*
`char[] toCharArray()`	Масив символів - копія рядка
`String substring(int` *початковий-індекс***`)`**	Підрядок початкового рядка, що починається з позиції *початковий-індекс* початкового рядка
`String substring(int` *начальный-индекс***`, int`** *конечный-индекс***`)`**	Підрядок початкового рядка, що починається в позиції *початковий-індекс* і закінчується в позиції *кінцевий-індекс-1* початкового рядка
`void getChars (int` *початковий-індекс***`, int`** *кінцевий-індекс***`, char[]`** *масив***`, int`** *індекс-вставки***`)`**	Значення, що повертається немає. Копіює частину рядка, починаючи з символу в позиції *початковий-індекс* і закінчуючи символом в позиції *індекс-вставки* + (*кінцевий-індекс* - *початковий-індекс) -1* у символьний масив, починаючи з позиції *індекс-вставки*

3.1.6. Модифікація рядків

Створення нових рядків на основі існуючих рядків виконується за допомогою методів класу String, наведених у таблиці 6.

Таблиця 6. Методи створення нових рядків класу String

Метод	Значення, що повертається
`String concat(String` *строка***`)`**	Вихідний рядок, у кінець якої додано *рядок*
`String toLowerCase()`	Вихідний рядок, перекладений у нижній регіст.
`String toUpperCase()`	Вихідний рядок, перекладений у верхній регістр
`String trim()`	Вихідний рядок, з якого виключені початкові і кінцеві пробільні символи
`String replace(char` *символ-1***`, char`** *символ-2***`)`**	Вихідний рядок, в якій всі символи *символ-1* замінені на *символ-2*
`public static String valueOf(`*тип ім'я***`)`**	Рядок, перетворений з примітивних типів даних. Допустимі типи параметра: **`boolean`**, **`char`**, **`int`**, **`long`**, **`float`**, **`double`** или **`char[]`**
`public static String valueOf(char[]` *масив***`, int`** *початковий-індекс***`, int`** *довжина***`)`**	Рядок, отриманий з фрагмента *масиву*, який починається в позиції *початковий-індекс* і заданої *довжини*

Методи класу StringBuffer, подані у таблиці 7, дозволяють безпосередньо модифікувати рядок.

Таблиця 7. Методи модифікації рядків класу StringBuffer

Метод	Дія
public void setCharAt(int *індекс*, **char** *символ*)	Поміщає в позиції *індекс* заданий *символ*
public void deleteCharAt(int *індекс*)	Видаляє символ у позиції *індекс*
public StringBuffer replace(int *початковий-індекс*, **int** *кінцевий-індекс*, **String** *рядок*)	Замінює фрагмент рядка, починаючи з позиції *початковий-індекс* і до позиції *кінцевий-індекс-1* рядок, а потім повертає змінений *рядок*
public StringBuffer append(тип ім'я**)**	Додає в кінець рядка дане заданого *типу* з заданим ім'ям і повертає змінений *рядок* (див. нижче). Допустимі *типи* параметра: **Object**, **boolean**, **char**, **int**, **long**, **float**, **double**, **String**, **StringBuffer** або **char[]**
public StringBuffer insert(int *початковий-індекс*, *тип ім'я*)	Вставляє в рядок в позиції *початковий-індекс* дані заданого типу з заданим ім'ям і повертає змінений рядок. Допустимі типи параметра: **Object**, **boolean**, **char**, **int**, **long**, **float**, **double**, **String**, **StringBuffer** или **char[]**
public StringBuffer append(char[] *масив*, **int** *початковий-індекс*, **int** *довжина*	Додає в рядок фрагмент символьного *масиву*, що починається з позиції *початковий-індекс* заданої довжини, і повертає змінений рядок
public StringBuffer insert(int *початковий-індекс-рядку*, **char[]** *масив*, **int** *початковий-індекс-масиву*, **int** *довжина*)	Вставляє в рядок в позиції *початковий-індекс-рядку* фрагмент символьного масиву, що починається з позиції *початковий-індекс-масиву* заданої довжини, і повертає змінений рядок

3.2. Регулярні вирази в Java

3.2.1. Основні відомості про регулярні вирази

Функції роботи з підрядками дозволяють виконати операції пошуку підрядків у рядку і заміни підрядків у рядку. Протее під час роботи з даними часто доводиться виконувати операції пошуку і заміни за досить складними алгоритмами, наприклад, знайти перше входження цифри в рядку. Хоча такі операції можна виконати, використовуючи функції роботи з рядками, умовні оператори і оператори циклу, в мові Java, як і в інших мовах програмування, існує більш зручний спосіб − використання регулярних виразів.

Регулярні вирази використовуються для вирішення таких завдань:

• перевірки даних на наявність деякої послідовності даних, заданих за допомогою певного зразка, званого шаблоном (**pattern**);

• заміни або видалення даних;

• вилучення деякої послідовності з даних.

Синтаксис регулярних виразів в Java в основному такий самий, як і в інших мовах програмування.

3.2.1.1. Синтаксис регулярного виразу

Регулярний вираз в мові Java є об'єктом класу String, тобто є рядком - послідовністю символів.

Символи в рядку можуть бути таких типів:

• алфавітно-цифрові символи, включаючи літери кирилиці;

• символ **'\\'** – зворотна коса риска (зворотний слеш);

• символ **'\0num'** – вісімкове число, де num – одна, дві або три вісімкові цифри;

• символ **'\xhh'** – код символа ASCII, де hh – дві шістнадцяткові цифри;

• символ **'\uhhhh'** – код символа Unicode, де hhhh – чотири шістнадцяткові цифри;

• символ табуляції (**'\t'** або **'\u0009'**);

• символ нового рядка (**'\n'** або **'\u000A'**);

• символ повернення каретки (**'\r'** або **'\u000D'**);

• символ переходу до нової сторінки (**'\f'** або **'\u000C'**);

• символ звукового сигналу (**'\a'** або **'\u0007'**);

• символ Escape (Esc) (**'\u001B'**);

• символ **'\cx'** – відповідає керуючому символу x (наприклад, \cM відповідає символу Ctrl+M або символу повернення каретки).

Деякі символи в регулярних виразах, так звані метасимволи, мають особливе значення. Метасимволами є такі символи:

^ $ ( ) \ | [{ ? . + *

Саме використання метасимволів забезпечує всю міць і гнучкість регулярних виразів.

Якщо в регулярному виразі необхідно розглядати метасимвол як звичайний символ, перед ним потрібно поставити символ '\\', наприклад, "**\\(**".

3.2.1.2. Операція альтернації

У регулярних виразах можна об'єднувати кілька шаблонів, так щоб знайдений рядок відповідав хоча б одному з них. Для вирішення такої проблеми служить операція альтернації, яка в регулярних виразах задається символом "|", наприклад шаблон "**Internet|Інтернет**" означає пошук у заданому рядку або рядка "Internet", або рядка "Інтернет".

3.2.1.3. Одиночний метасимвол

Метасимвол точка "." всередині регулярного виразу відповідає будь-якому одиночному символу, крім символу перекладу рядка.

3.2.1.4. Квантифікатори

Квантифікатори – це метасимволи, використовувані для вказівки кількісних відносин між символами в шаблоні і в шуканому рядку. Квантифікатор може бути поставлений після одиночного символу або після групи символів.

Найпростішим квантифікатором є метасимвол "+". Він означає, що символ, який стоїть перед ним, відповідає кільком таким символам, що йдуть підряд у рядку пошуку. Кількість символів може бути будь-якою (максимально великою в рамках відповідності шаблону), але повинен бути наявним хоча б один символ.

Дія метасимвола "*" схожа на дію метасимвола "+". Метасимвол "*" показує, що символ, який стоїть перед ним, трапляється нуль або більше разів.

Метасимвол "?" показує, символ, який стоїть перед ним, повинен траплятися або один раз, або не траплятися взагалі.

Якщо необхідно зазначити точну кількість повторень символу, можна скористатися конструкцією

`{n,m}`

Маємо, що **n** – мінімально допустима кількість повторень попереднього символу, **m** − максимально допустима кількість повторень. Один з параметрів n або m можна опустити.

Фактично квантифікатори "+", "*" і "?" є окремими випадками конструкції `{n, m}`: відповідно, `{1,}`, `{0,}` і `{0,1}`.

У регулярних виразах часто використовують поєднання метасимволов "*". Йому відповідають будь-які символи. За правилами оброблення регулярних виразів знаходиться найдовший рядок, який все ще задовольняє шаблону пошуку.

Якщо необхідно обмежити пошук, потрібно після квантифікатора (в тому числі і символу "?") поставити символ "?".

3.2.1.5. Класи символів

Для пошуку в регулярних виразах можна задавати також класи символів, укладені в квадратні дужки. Під час пошуку всі символи в класі розглядаються як один символ. Усередині класу можна задавати діапазон символів (коли такий діапазон має сенс), поміщаючи дефіс між кордонами діапазону. Всередині символьних класів більшість метасимволів втрачають свої значення і стають звичайними символами.

Якщо першим символом класу є знак вставки "^", то значення виразу інвертується. Іншими словами, такому класу відповідає будь-який символ, який не входить в клас.

Так як у класах символи "`]`", "^" і "`-`" мають спеціальне значення, для їх використання в класі існують певні правила:

• літерал "`^`" не повинен бути першим символом класу;

• перед літералом "`]`" повинен стояти символ зворотної косої риски;

• для приміщення в клас символу "`-`" мало або поставити його на першу позицію, або помістити перед ним символ зворотної косої риски.

3.2.1.6. Спеціальні символи

Найбільш поширені класи символів можна задати за допомогою таких спеціальних символів:

• **`\d`** – відповідає будь-якому цифрового символу (еквівалентно [0–9]);

• **`\D`** – відповідає будь-якому нецифровому символу (еквівалентно [^0–9]);

• **`\w`** – відповідає будь-якій латинській букві або цифрі (еквівалентно [A–Za-z0–9]);

• **`\W`** – відповідає будь-якому небуквеному (латинському) і цифровому символу (еквівалентно [^A–Za-z0–9]);

• **`\s`** – відповідає будь-якому символу пробілу (еквівалентно [\f\n\r\t\v]);

• **`\S`** – відповідає будь-якому непробільному символу (еквівалентно [^\f\n\r\t\v]).

Необхідно зазначити, що спеціальні символи \w і \w не можна використовувати для букв кирилиці, а також букв західноєвропейських алфавітів, відмінних від латинських букв. У цьому разі необхідно безпосередньо задавати діапазон символів, як це робиться для класів символів.

3.2.1.7. Анкери

За допомогою анкерів можна зазначити, в якому місці рядка повинно бути знайдено відповідність з шаблоном.

В Java визначені такі анкери:

• **`^`** – відповідає позиції на початку рядка;

• **`$`** – відповідає позиції в кінці рядка;

• **`\b`** – відповідає границі слова, тобто границі між словом і пробільним символом;

• **`\B`** – відповідає не на границі слова.

На жаль, анкери \b і \B діють лише для рядків, що складаються з латинських букв.

3.2.1.8. Групування елементів і зворотні посилання

Операція групування елементів, тобто вивід групи елементів у круглі дужки, дозволяє розглядати дану групу елементів як один елемент.

Якщо в регулярних виразах використовуються дужки, частини шуканого рядка, відповідні фрагментами в дужках, запам'ятовуються в спеціальних змінних $1 (перший фрагмент в дужках), $2 (другий фрагмент у дужках), $3 (третій фрагмент у дужках) тощо. Така операція називається захопленням (**capture**) змінної.

Для вкладених дужок змінні $1-$9 формуються в порядку появи лівої (що відкриває) дужки виразу.

Слід зазначити, що змінні модифікуються під час кожного успішного пошуку, незалежно від використання в регулярному виразі дужок. Крім того, значення цих змінних установлюються тоді і лише тоді, коли рядок повністю відповідає шаблону. В іншому випадку значення цих змінних дорівнюють порожньому рядку.

Змінні $1-$9 можна записувати і в вираженні шаблону в формі \i, де і – номер змінної. Змінну називає зворотнім посиланням.

Змінні $1-$9 не завжди необхідно використовувати як результат пошуку відповідності шаблоном. У разі, коли необхідно згрупувати будь-які символи шаблону, але не виконувати для них операції по визначенню відповідних змінних (не виконувати для них операцію захоплення), використовується така форма групування:

```
(?:символи)
```

Іншим видом групування є групування з «загляданням вперед». Групування в цьому випадку записується в такому вигляді:

```
шаблон(?=символи)
```

У цьому разі під час пошуку відповідності шаблоном враховуються символи, що задані в дужках і йдуть після шаблону, але в результат пошуку ці символи не входять. Такий пошук починається з позиції, з якої починаються символи в дужках, тобто результат «заглядання вперед» не враховується.

Друга форма групування з «загляданням вперед» записується в такому вигляді:

```
шаблон(?=символи)
```

На відміну від першої форми символи в дужках не повинні міститися у відповідності з шаблоном. У результаті пошуку ці символи не входять і результат «заглядання вперед» також не враховується.

Ггрупування «заглядання назад» записується в такому вигляді:

```
(? <= Символи) шаблон
```

У цьому разі при пошуку відповідності шаблону враховуються символи, що задані в дужках і йдуть перед шаблоном, але в результат пошуку ці символи не входять. Такий пошук починається з першої позиції після шаблону.

Друга форма угруповання з «загляданням назад» записується в такому вигляді:

```
(? <! Символи) шаблон
```

На відміну від першої форми символи в дужках не повинні міститися відповідно до шаблону. У результат пошуку ці символи не входять і результат «заглядання назад» також не враховується.

Для роботи з регулярними виразами в Java використовуються класи **Pattern**, **Matcher** і **PatternSyntaxException** пакета **java.util.regex**.

3.2.2. Клас Pattern

3.2.2.1. Створення об'єкта класу Pattern.

Об'єкт класу Pattern є відкомпільоване поданням шаблону регулярного виразу і створюється не за допомогою ключового слова new, а за допомогою статичних методів **compile()**класу Pattern.

Метод

`public static Pattern compile(String шаблон)`

повертає об'єкт класу Pattern для заданого в параметрі шаблону.

Метод

`public static Pattern compile(String шаблон, int прапорці)`

повертає об'єкт класу Pattern для заданого в параметрі шаблону з заданими прапорцями.

Прапорці (таблиця 8) подані в Java як статичні поля типу **public static final int** класу Pattern.

Таблиця 8. Прапорці класу Pattern

Прапор	Числове значення	Застосування
`CASE_INSENSITIVE`	`2`	Включає пошук відповідності без урахування верхнього або нижнього регістру, тобто рядки "`abc`", "`Abc`" і "`ABC`"будуть вважатися відповідними регулярному виразу "`abc`" (за відключеного прапорця шаблоном буде відповідати тільки перший рядок)
`UNICODE_CASE`	`64`	Якщо цей прапорець включений разом з прапорцем `CASE_INSENSITIVE`, то верхній і нижній регістри букв у коді Unicode не враховуються під час пошуку відповідності, тобто рядки "`рядок`", "`Рядок`" і "`РЯДОК`" будуть вважатися відповідними регулярному виразу "рядок" (за відключеного прапорця `UNICODE_CASE` і включеному прапорці `CASE_INSENSITIVE` шаблоном буде відповідати за виключенням окремих виразів, що містять латинські букви)
`UNIX_LINES`	`1`	Під час включення цього прапорця тільки символ "`\n`" враховується як символ закінчення рядка, в якій виконується пошук відповідності

Продовження таблиці 8.

MULTILINE	**8**	Якщо всередині рядка, в якій виконується пошук відповідності, є символи "**\n**", то вважається що рядок складається з декількох рядків (якщо прапорець вимкнений, то вважається, що пошук відповідності проводиться в одному рядку, незалежно від наявності "**\n**")
LITERAL	**16**	Усі символи шаблону, включаючи метасимволи, розглядаються як звичайні символи (якщо прапорець вимкнений, метасимвол у рядку обробляються під час компіляції)
DOTALL	**32**	Якщо в шаблоні є метасимвол ".", То йому буде відповідати будь-який символ, включаючи символ "**\n**" (якщо прапорець вимкнений, метасимвол "." відповідатиме будь-якому символу, виключаючи символ "**\n**")
COMMENTS	**4**	У рядку шаблону допустимі пробіли і коментарі, що починаються з символу "**#**" до кінця рядка (під час компіляції шаблону пробіли і коментарі будуть проігноровані)
CANON_EQ	**128**	Під час пошуку відповідності буде враховуватися відповідність між кодом символу і самі символом, тобто під час включення прапорця латинська буква "**a**" буде відповідати коду Unicode цієї букви "**\u00E5**" в шаблоні

Прапорці можна включати безпосередньо в шаблоні, використовуючи таку синтаксичну форму:

(?рядок-символів)

де символи в рядку-символів можуть мати одне з таких значень

i – для прапорця CASE_INSENSITIVE;

d – для прапорця UNIX_LINES;

m – для прапорця MULTILINE;

s – для прапорця DOTALL;

u – для прапорця UNICODE_CASE;

x – для прапорця COMMENTS.

3.2.2.2. Методи класу Pattern

Методи класу Pattern наведені у таблиці 9.

Таблиця 9. Методи класу Pattern

Метод	Дія та повернення значення
`public static boolean matches(String` *шаблон*`, CharSequence` *рядок-пошуку*`)`	Перевіряє відповідність *шаблону рядку-пошуку* і повертає значення **`true`**, якщо рядок пошуку відповідає шаблону і **`false`** - в протилежному разі
`public String pattern()`	Повертає рядок шаблону для об'єкта класу **`Pattern`**
`public int flags()`	Повертає числове значення прапорця для об'єкта класу **`Pattern`**; (якщо задано кілька прапорців, повертає суму їх числових значень)
`public static String quote(String` *рядок*`)`	Повертає строковий шаблон для заданого *рядка* (див. поле **`LITERAL`**).
`public String[] split(CharSequence` *рядок-пошуку*`)`	Створює з *рядка-пошуку* масив, розділений на елементи за шаблоном, заданим в об'єкті класу **`Pattern`**
`public String[] split(CharSequence` *рядок-пошуку*`, int` *межа*`)`	Створює з *рядка-пошуку* масив, розділений на елементи за шаблоном, заданим в об'єкті класу **`Pattern`**, і з заданою в параметрі *межа* кількістю елементів (якщо значення параметра більше або дорівнює кількості елементів, або менше 0, виводяться всі елементи, якщо воно менше кількості елементів ? всі відповідності, що залишилися, виводяться в останньому елементі масиву)
`public String toString()`	Повертає рядкове представлення відкомпільованого шаблону

3.2.3. Клас Matcher

3.2.3.1. Створення об'єкта класу Matcher

Клас Matcher забезпечує виконання пошуку або заміни відповідності заданому об'єктом класу Pattern шаблоном.

Об'єкт класу Matcher створюється за допомогою методу

`public Matcher matcher(CharSequence рядок-пошуку)` класу Pattern для *рядка-пошуку*.

Рядкове представлення об'єкту класу Matcher можна отримати за допомогою методу

`public String toString()`.

3.2.3.2. Операції з регіонами

Пошук відповідності виконується в підрядку початкового рядка, що називається регіоном (**region**). За замовчуванням регіоном є вся послідовність символів, що вводиться.

Методи для операцій з регіонами наведено у таблиці 10.

Таблиця 10. Методи для операцій з регіонами класу Matcher

Метод	Дія і повернення значення
public Matcher region(int *початковий-індекс*, **int** *кінцевий-індекс***)**	Встановлює межі регіону і повертає об'єкт класу **Matcher** для підрядка, що починається з *початкового-індексу* і закінчується індексом, на одиницю меншим, ніж *кінцевий-індекс*
public int regionStart()	Отримує і повертає індекс початку регіону
public int regionEnd()	Отримує і повертає індекс закінчення регіону
public Matcher useAnchoringBounds(boolean *прапорець***)**	Якщо параметр встановлено *прапорець* в **true**, то для завдання шаблону на кордонах регіону можна використовувати анкери "**^**" і "**$**". В іншому разі (якщо *прапорець* встановлений в **false**) відповідність анкерів "**^**" і "**$**"на кордонах регіону не перевіряється. За замовчуванням значення *прапорця* одно **true**
public Matcher useTransparentBounds(boolean *прапорець***)**	Якщо параметр встановлено *прапорець* в **true**, межі регіону є прозорими для шаблонів, що містять граничні умови, а також «заглядання вперед» і «заглядання назад», в зокрема і за межі регіону. В іншому разі (якщо *прапорець* встановлений в **false**) межі регіону будуть непрозорими, тобто операції з символами за межами регіону заборонені. За замовчуванням значення *прапорця* дорівнює **false**
public boolean hasAnchoringBounds()	Перевіряє, чи є межі анкерними. Під час виконанні умови метод повертає **true**, в іншому разі – **false**
public boolean hasTransparentBounds()	Перевіряє, чи є межі прозорими. Під час виконанні умови метод повертає **true**, в іншому разі – **false**

3.2.3.3. Методи пошуку відповідників

Методи пошуку відповідників наведені у таблиці 11.

Таблиця 11. Методи пошуку відповідностей класу Matcher

Метод	Дія та повернення значення
public Matcher region(int *початковий-індекс*, **int** *кінцевий індекс***)**	Встановлює межі регіону й повертає об'єкт класу **Matcher** для підрядка, що починається з *початкового-індексу* і закінчується індексом, на одиницю меншим, ніж *кінцевий індекс*
public int regionStart()	Отримує і повертає індекс початку регіону
public boolean matches()	Виконує для об'єкта класу **Matcher** пошук на відповідність всього регіону, починаючи з початку регіону. Повертає **true**, якщо відповідність знайдено і **false** – в іншому разі.

Продовження таблиці 11.

public boolean lookingAt()	Виконує для об'єкта класу **Matcher** пошук, починаючи з початку регіону на наявність шаблону в регіоні, але необов'язково відповідності усього регіону шаблоном. Повертає **true**, якщо відповідність знайдено і **false** – в іншому разі
public boolean find()	Виконує для об'єкта класу **Matcher** пошук, починаючи з початку регіону або, якщо попередній виклик методу був успішним, і об'єкт класу **Matcher** не був скинутий, з першого символу після знайденого попередньої відповідності. Повертає **true**, якщо відповідність знайдено і **false** – в іншому разі
public boolean find(int *початковий-індекс***)**	Виконується так само, як і метод **find()** без параметрів , але пошук починається не з початку регіону, а заданого *початкового-індексу*. Якщо необхідно знайти всі відповідності шаблону у рядку, починаючи з *початкового-індексу*, цей метод можна використовувати лише для пошуку першої відповідності. Всі решта відповідності визначаються за допомогою методу **find()** без параметрів
public String group()	Повертає попереднє, знайдене за допомогою методів **matches()**,**lookingAt()** або **find()**,відповідність
public String group(int *номер-группы***)** –	Повертає попереднє, знайдене за допомогою методів **matches()**,**lookingAt()** або **find()**,відповідність для заданого *номера -групи* (групи нумеруються зліва направо, починаючи з **1**; якщо задано **0**, метод виконується так само, як метод **group()** без параметрів)
public int start()	Повертає початковий індекс в рядку пошуку попередньої відповідності, знайденого за допомогою методів **matches()**,**lookingAt()** або **find()**
public int start(int *номер-группы***)**	Повертає початковий індекс у рядку пошуку попередньої відповідності, знайденої за допомогою методів **matches()**,**lookingAt()** або **find()**,для заданого *номера-групи* (групи нумеруються зліва направо, починаючи з **1**; якщо задано **0**, метод виконується аналогічно попередньому методу **start()** без параметрів)
public int end()	Повертає кінцевий індекс рядку пошуку попередньої відповідності, знайденої за допомогою методів **matches()**,**lookingAt()** або **find()**
public int end(int *номер-групи***)**	Повертає кінцевий індекс рядку пошуку попередньої відповідності, знайденої за допомогою методів **matches()**,**lookingAt()** або **find()**,для заданого *номера-групи* (групи

Закінчення таблиці 11.

`ublic int groupCount()`	Повертає кількість груп в об'єкті класу **`Matcher`**, знайдених з допомогою методів **`matches()`**,**`lookingAt()`** або **`find()`**
`ublic Pattern pattern()`	Повертає шаблон, який відповідає об'єкту класу **`Match`**
`ublic Matcher sePattern(Pattern` *новий-шаблон*`)`	Замінює шаблон для об'єкта класу **`Match`** на *новий-шаблон*
`ublic boolean hitEnd()`	Повертає **`true`**, якщо пошук відповідності дійшов до закінчення рядка пошуку і **`false`** – в іншому разі
`ublic boolean requireEnd()`	Повертає **`true`** і відповідність було знайдено, то подальший пошук може призвести до втрати відповідності. Якщо метод повертає **`false`** і відповідність було знайдено, то подальший пошук не призведе до втрати відповідності. Якщо відповідності не знайдено, метод не має значення

3.2.3.4. Методи заміни

Методи класу Matcher дозволяють не лише виконати пошук у рядку за заданим шаблоном, а й замінити знайдені відповідності заданими послідовностями символів – рядками заміни.

Методи заміни наведено у таблиці 12.

Таблиця 12. Методи заміни класу Matcher

Метод	Дія і значення, яке повертається
`public String replaceFirst(String` *рядок-заміни*`)`	Замінює лише першу відповідність у рядку пошуку *рядком-заміни* і повертає змінений рядок.
`public String replaceAll(String` *рядок-заміни*`)`	Замінює всі відповідності в рядку пошуку *рядком-заміни* і повертає змінений рядок.
`public Matcher appendReplacement(StringBuffer` *новий-рядок*`,` **`String`** *рядок-заміни*`)`	Формує *новий-рядок* за таким алгоритмом: • пересилає символи рядка пошуку в *новий-рядок*, починаючи з кінцевої позиції (append position) до символу на одиницю меншого, ніж символ, який визначається методом **`start()`** об'єкта **`Match`**; • потім до нового рядка додається *рядок-заміни*; після цього кінцева позиція у новому рядку стає рівною позицією, яка визначається методом **`end()`** об'єкта **`Match`**. На початку перегляду і заміни значення кінцевої позиції дорівнює **`0`**

Продовження таблиці 12.

public StringBuffer appendTail(StringBuffer *новий-рядок***)**	Пересилає символи рядка пошуку в *новий-рядок*, починаючи з кінцевої позиції і до кінця рядка пошуку. Цей метод використовується разом з методом **appendReplacement()** для завершення процесу пошуку і заміни в рядку.
public static String quoteReplacement(String *рядок-заміни*)	Перетворює рядок заміни в строкову форму, в якій метасимволи, такі як **"$"**, розглядаються як звичайні символи (діє аналогічно методу **quote()** класу **Pattern**) і повертає змінений рядок.
public Matcher reset()	Скидає всі збережені в об'єкті класу **Matcher** дані, установлює кінцеву позицію в **0**, регіоном стає весь рядок пошуку (однак на стан анкерного кордону та межі прозорості скидання не впливає) і повертає змінений об'єкт класу **Matcher**;
public Matcher reset(CharSequence *рядок-пошуку***)**	Виконує ті самі дії, що і метод **reset()** без параметрів, але задає для об'єкта нову послідовність символів пошуку.

3.2.4. Клас PatternSyntaxException

Клас PatternSyntaxException кидає виняток, якщо регулярний вираз (шаблон) містить синтаксичну помилку.

Методи цього класу наведено у таблиці 13.

Таблиця 13. Методи класу PatternSyntaxException

Метод	Повернення значення
public String getPattern()	Шаблон, що містить помилку
public String getDescription()	Опис помилки
public int getIndex()	Позиція помилки символу в шаблоні
public String getMessage()	Повідомлення про помилку, яке містить всі перераховані вище компоненти: опис помилки та її індекс, шаблон, що містить помилку і візуальну індикацію індексу помилки усередині шаблону

3.2.5. Методи класу String для роботи з регулярними виразами

Для роботи з регулярними виразами в класі String визначено методи, наведені у таблиці 14.

3.3. Методи в Java

Для визначення методів використовується такий формат:

```
  повертаючий-тип ідентифікатор-методу (параметри)
{
        тіло-методу
}
```

Таблиця 14. Методи для роботи з регулярними виразами класу String

Метод	Дія і повернення значення
public boolean matches(String *шаблон***)**	Якщо об'єкт класу **String** відповідає шаблону, повертає значення **true**, в іншому разі повертає **false** (діє аналогічно методу **matches()** класа **Pattern**)
public String[] split(String *шаблон***)**	Створює для об'єкта класу **String** масив рядків, розділений на елементи за заданим *шаблоном* (діє аналогічно відповідним методом **split()** класа **Pattern**)
public String[] split(String *шаблон***, int** *межа***)**	Створює для об'єкта класу **String** масив рядків, розділений на елементи за заданим *шаблоном,* і з заданим параметром *межа* кількістю елементів (якщо значення більше або дорівнює кількості елементів, або менше 0, виводяться всі елементи, якщо менше кількості елементів – всі залишилися відповідності виводяться в останньому елементі масиву) (діє аналогічно відповідним методом **split()** класа **Pattern**)
public String replaceFirst(String *шаблон***, String** *рядок-заміни*)	Замінює в об'єкті **String** перше відповідність *шаблону* на *рядок-заміни* і повертає змінену рядок (діє аналогічно методу **replaceFirst()** класа **Match**)
public String replaceAll(String *шаблон***, String** *рядок-заміни*)	Замінює в об'єкті **String** усі відповідності *шаблону* на *рядок-заміни* і повертає змінену рядок (діє аналогічно методу **replaceAll()** класа **Match**).

Повертаючий-тип визначає тип даних, які повертає метод під час виклику (повертається відповідь на повідомлення). Якщо метод не повертає ніякого значення, то повертаючий-тип має значення **void**. **Ідентифікатор-методу** визначає ім'я методу, а **параметри** – список параметрів, які необхідно передати методу під час його виклику. Параметри в списку відокремлюються один від одного комами, і кожен параметр має такий вигляд:

```
тип ідентифікатор
```

де тип визначає тип значення, а ідентифікатор – ім'я параметра (область дії цього імені – лише всередині тіла методу). Якщо параметри відсутні, після ідентифікатор-методу зазначаються порожні дужки.

Ідентифікатор методу формується за тими самими правилами, що і ім'я змінної, тобто перша буква імені повинна бути великою, а інші слова, складові ім'я, починаються з великої літери, наприклад method1 або myMethod.

Тіло-методу містить оператори, що реалізують дії, які виконуються цим методом.

Якщо тип значення, що повертається, не **void**, у тілі методу повинен бути хоча б один оператор

```
return вираз;
```

де тип виразу повинен збігатися з типом значення, що повертається. Цей оператор повертає результат обчислення виразу в точку виклику методу.

Якщо тип значення, що повертається – void, повернення з методу виконується після виконання останнього оператора тіла методу, або в результаті виконання пропозиції

```
return;
```

(таких пропозицій в тілі методу може бути кілька).

Тип заданого аргументу в Java повинен строго відповідати типу параметра в оголошенні методу.

У мові Java в межах одного класу можна визначити два або більше методів, які спільно використовують одне і те саме ім'я, але мають різну кількість параметрів. Коли це відбувається, методи називають перевантаженими, а про процес кажуть як про перевантаження методів (**method overloading**).

Коли метод викликається, то за кількістю параметрів та/або їх типами середовище виконання Java визначає, яку саме версію перевантаженого методу потрібно викликати (тип значення, що повертається до уваги не береться, хоча, в принципі, він теж може відрізнятися у різних версій перевантажених методів).

За замовчуванням метод, як і змінна, доступний лише класам в тому самому пакеті (наборі класів), що і початковий клас. Якщо перед її повертаючим типом заданий модифікатор доступу **public**, то метод є глобальним і доступний будь-яким об'єктам, а модифікатор **private** означає, що метод доступний у тому класі, в якому він був оголошений, тобто метод інкапсульований у цьому класі. Модифікатор **final** означає, що метод не можна перевизначити в дочірніх класах.

4. Порядок виконання роботи

Напишіть програму на мові Java за одним із наведених нижче варіантів. Присвоювання значень рядків виробляється за допомогою завдання аргументів у командному рядку під час виконання програми (якщо аргумент містить прогалини, він повинен бути укладений у подвійні апострофи, наприклад, "Рядок 1"). Перевірка відповідності рядка заданому шаблону виконується за допомогою регулярних виразів. Вихідні дані та одержані результати виводяться на екран з пояснювальними повідомленнями. Аналіз та/або зміна аргументів і виведення результату виконуються в двох окремих методах.

Варіант 1

Аналіз типів аргументів, які задаються під час запуску програми. Якщо аргумент є правильним цілим числом (шаблон: одна або декілька цифр, першим символом може бути або цифра, або знак "+" або "-"), то тип

аргументу "Integer", інакше "String". Програма виводить кількість заданих аргументів, для кожного аргументу, його тип і значення.

Варіант 2

Аналіз типів аргументів, які задаються під час запуску програми. Якщо аргумент є десятковим числом з цілою і дробовою частиною (шаблон: складається з однієї або декількох цифр, однією десяткової точки, яка може бути на початку, в середині або в кінці числа, і, крім того, першим символом числа може бути знак "+" або "-"), то тип аргументу "Decimal", інакше "String". Програма виводить кількість заданих аргументів, для кожного аргументу, його тип і значення.

Варіант 3

Аналіз типів аргументів, які задаються під час запуску програми. Якщо аргумент є числом з плаваючою точкою (шаблон: складається з мантиси – одна або кілька цифр, можливо, із знаком "+" або "-", яка може містити десяткову крапку на початку, середині або наприкінці, а також порядку – цілого числа зі знаком "+" або "-" або без знаку, роздільником між мантисою і порядком служить символ " e " або символ " E "), то тип аргументу "Float", інакше "String". Програма виводить кількість заданих аргументів, для кожного аргументу, його тип і значення.

Варіант 4

Аналіз типів аргументів, які задаються під час запуску програми. Якщо аргумент є цілим вісімковим числом (шаблон: складається з цифр від 0 до 7, причому першою цифрою повинен бути 0), то тип аргументу "Octal", інакше "String". Програма виводить кількість заданих аргументів, для кожного аргументу, його тип і значення.

Варіант 5

Аналіз типів аргументів, які задаються під час запуску програми. Якщо аргумент є цілим шістнадцятковим числом (шаблон: складається з цифр від 0 до 9 і літер A(a), B(b), C(c), D(d), E(e),F(f), перед числом повинні стояти символи "0X " або "0x "), то тип аргументу "Hexadecimal", інакше "String". Програма виводить кількість заданих аргументів, для кожного аргументу, його тип і значення.

Варіант 6

Аналіз типів аргументів, які задаються під час запуску програми. Якщо аргумент є цілим двійковим числом (шаблон: одна і більше цифр 0 і 1), то тип аргументу "Binary", інакше "String". Програма виводить кількість заданих аргументів, для кожного аргументу, його тип і значення.

Варіант 7

Аналіз типів аргументів, які задаються під час запуску програми. Якщо аргумент має вигляд "ім'я=значення", то він є ключовим параметром (тип "Keyed"), якщо аргумент має вигляд "- значення " або "/значення", то він є опцією (тип "Optional") і якщо має вигляд "значення", то є безпосереднім параметром (тип "Immediate"). Шаблон для значення: одна або декілька цифр і літер (включаючи літери кирилиці). Програма виводить кількість заданих аргументів, для кожного аргументу, його тип і значення (для ключових параметрів додатково виводиться ім'я параметра).

Варіант 8

Аналіз типів аргументів, які задаються під час запуску програми. Якщо аргумент має вигляд "значення", то він є поодиноким параметром (тип "Single"), якщо аргумент має вигляд "значення, значення,... ", то він є списком (тип "List"). Програма виводить кількість заданих аргументів, для кожного аргументу, його тип і значення (для кожного параметра-списку під час виведенні його значення перетворюється в масив типу String, кожен елемент якого містить елемент списку і виводиться в циклі за елементами).

Варіант 9

Аналіз типів аргументів, які задаються під час запуску програми. Якщо аргумент є правильним ідентифікатором Java (шаблон: складається з латинських літер, цифр та символу "$" та "_", вважаються буквами, і, крім того, перший символ є буквою), то його тип "Identifier", якщо аргумент є ключовим словом Java (для прикладу задати кілька ключових слів Java, "if", "for", "while", "do" і "else"), то його тип "Keyword", інакше його тип вважається "Illegal". Програма виводить кількість заданих аргументів, для кожного аргументу його тип і значення.

Варіант 10

Аналіз аргументів, які задаються під час запуску програми. Програма шукає найбільший загальний підрядок, що міститься у введених аргументах (шаблон аргументу: рядок або латинських літер, або літер кирилиці). Програма виводить кількість заданих аргументів, їх значення та найбільший загальний підрядок або повідомлення про те, що спільного підрядка немає.

Варіант 11

Аналіз аргументів, які задаються під час запуску програми. Програма визначає, які символи містяться у введених аргументах (наприклад, аргументи "abc", "cf", "bfc" містять символи "abcf"). Шаблон аргументу: рядок або латинських літер, або букв кирилиці. Програма виводить кількість заданих аргументів, значення аргументів і рядок символів, що містяться в аргументах.

Варіант 12

Аналіз аргументів, які задаються під час запуску програми. Програма видаляє з аргументів всі повторювані символи, крім одного (наприклад, аргумент "abcadc", перетворюється в "abcd"). Шаблон аргументу: рядок або латинських літер, або букв кирилиці. Програма виводить кількість заданих аргументів, їх значення і перетворені значення аргументів.

Варіант 13

Аналіз аргументів, які задаються під час запуску програми. Програма визначає рядок символів, що трапляються лише в одному з аргументів (наприклад, для аргументів "agc", "cf", "bfc" такий рядком буде рядок "gb"). Шаблон аргументу: рядок або латинських літер, або букв кирилиці. Програма виводить кількість заданих аргументів, їх значення і знайдену рядок символів або повідомлення про те, що таких символів немає.

Варіант 14

Аналіз аргументів, які задаються під час запуску програми. Програма визначає, скільки разів символ трапляється у введених аргументах (наприклад, для аргументів agc", "cf", "bfc" символи "a", "g" и "b" трапляються один раз, символ "f" – два рази і символ "c" – три рази). Шаблон аргументу: рядок або латинських літер, або букв кирилиці. Програма виводить кількість заданих аргументів, їх значення і для кожного символу – частоту його повторення в аргументах.

Варіант 15

Аналіз аргументів, які задаються під час запуску програми. Програма сортує введені аргументи (шаблон аргументу: рядок латинських літер) за першим символом аргументу. Програма виводить кількість заданих параметрів, їх значення і список аргументів відсортованих.

Варіант 16

Аналіз аргументів, які задаються під час запуску програми. Програма визначає тип аргумент – ціле число або рядок (шаблон: цілим числом вважається рядок, який містить цифри і, крім того, першим символом рядка може бути цифра, знак "+" або "-"). Потім серед аргументів-чисел шукається максимальне число. Програма виводить кількість заданих аргументів, їх значення, кількість аргументів-чисел і значення максимального числа.

Варіант 17

Аналіз аргументів, які задаються під час запуску програми. Програма визначає, чи є серед введених аргументів однакові аргументи і кількість їх повторення. Шаблон аргументу: рядок або цифр, або латинських літер. Програма виводить кількість заданих аргументів, значення повторюваних

аргументів і кількість їх повторення або повідомлення про те, що повторюваних аргументів немає.

Варіант 18

Аналіз аргументів, які задаються під час запуску програми. Програма переставляє введені аргументи в порядку зростання їх довжини. Шаблон аргументу: рядок або цифр, або латинських літер, або букв кирилиці. Програма виводить кількість заданих аргументів, їх значення, а також список значень аргументів у порядку зростання їх довжини.

Варіант 19

Аналіз аргументів, які задаються під час запуску програми. Програма визначає, які з введених аргументів містять рядок, що задається як перший параметр. Шаблон аргументу: рядок або цифр, або латинських літер, або букв кирилиці. Програма виводить кількість заданих аргументів (без урахування першого аргументу) та аргументи, які містять заданий підрядок або повідомлення про те, що цей рядок не міститься у введених аргументах.

Варіант 20

Аналіз аргументів, які задаються під час запуску програми. Програма видаляє з масиву введених аргументів усі повторювані аргументи, крім одного (наприклад, з аргументів "ab", "cd", "ab" будуть залишені аргументи "ab" і "cd"). Шаблон аргументу: рядок латинських літер. Програма виводить кількість заданих аргументів, їх значення, а також кількість різних аргументів і їх значення.

Варіант 21

Аналіз аргументів, які задаються під час запуску програми. Програма визначає тип аргумент – ціле число або рядок (шаблон: цілим числом вважається рядок, який містить цифри і, крім того, першим символом рядка може бути цифра, знак "+" або "-"). Потім аргументи-числа сортуються в порядку зростання їх значення. Програма виводить кількість заданих аргументів, їх значення, кількість аргументів-чисел і їх значення в порядку зростання.

Варіант 22

Перетворення аргументів, які задаються під час запуску програми. Програма визначає тип аргументу – двійкове число без знака або рядок (шаблон: двійковим числом без знака вважається рядок, який містить одну або більше цифр 0 та 1). Введені аргументи-числа перетворюються в шістнадцяткові числа (кожні чотири цифри двійкового числа перетворюються в одне шістнадцяткове, тому, за необхідності, значення аргументу додаються нулі до довжини, кратної 4). Програма виводить

кількість заданих аргументів, їх значення, а також кількість аргументів-чисел і їх шістнадцяткові значення.

Варіант 23

Перетворення аргументів, які задаються під час запуску програми. Програма визначає тип аргумент – шістнадцяткове число без знака (шаблон: шістнадцятковим числом без знака вважається рядок, який містить цифри від 0 до 9 та букви A(a), B(b), C(c), D(d), E(e),F(f)) або рядок. Введені аргументи-числа перетворюються в двійкові числа. Програма виводить кількість заданих аргументів, їх значення, а також кількість аргументів-чисел і їх двійкові значення.

Варіант 24

Перетворення аргументів, які задаються під час запуску програми. Програма перетворює російські та латинські літери в аргументах у верхній регістр (якщо вони є малими). Шаблон аргументу: або рядки латинських літер, або рядки літер кирилиці. Програма виводить кількість заданих аргументів, їх значення, а також нові значення аргументів.

Варіант 25

Перетворення аргументів, які задаються під час запуску програми. Програма визначає тип аргументу – шістнадцяткове число без знаку (шаблон: шістнадцятковим числом без знака вважається рядок, який містить цифри від 0 до 9 та букви A(a), B(b), C(c), D(d), E(e),F(f)) або рядок. Введені аргументи-числа перетворюються у десяткові числа (кожна і-та цифра шістнадцяткового числа перетвориться в десяткове число N_i за формулою $N_i=16^{n-i-1}$, де n – кількість цифр у числі; i = (0,n) – індекс цифри в числі, шукане число є сумою всіх N_i). Програма виводить кількість заданих аргументів, їх значення, а також кількість аргументів-чисел та їх десяткові значення.

Варіант 26

Аналіз аргументів, які задаються під час запуску програми. Аргумент має такий формат:

ім'я-типу-або-ім'я-класу.ім'я змінної або ім'я-методу

Аргумент являє собою звернення до змінної або виклик методу для об'єкта чи класу в Java. Шаблон аргументу: латинські літери, цифри та символи "$" і "_", вважаються буквами, і, крім того, перший символ є буквою. Якщо перший символ звернення – заголовна буква, то виводиться тип "Static", якщо рядкова буква – виводиться "Object". Якщо ім'я змінної або методу починається з малої літери, якщо ім'я містить символи "("і ")", то виводиться тип обігу "Method", інакше виводиться тип обігу "Variable". Якщо будь-яка з наведених умов не виконується, то виводиться тип обігу "Illegal". Програма виводить кількість заданих аргументів, для кожного аргументу, його тип, тип обігу та значення.

Варіант 27

Аналіз аргументів, які задаються під час запуску програми. Програма розглядає аргументи, що вводяться як імена файлів MS DOS, визначає тип файла за його розширення ("Програма " – розширення .exe або .com, " Документ Word " – розширення .doc або " Текстовий файл " – розширення . txt), і виробляє перевірку правильності завдання імені файла за таким шаблоном:

1) в імені файла не повинно бути більше 8 символів, а розширення імені (якщо воно є) – не більше ніж 3 символи.

2) в імені файлу не повинно бути російських букв, прогалин, одиночних і подвійних апострофів, а також символів "-", "_", "$", "#", "&", "@", "!", "%", "(", ")", "{", "}", "~", "." та "*".

Програма виводить кількість заданих аргументів, і типи файлів для кожного аргументу (якщо ім'я файлу неправильне, виводиться тип "Неправильне ім'я ", якщо тип не збігається з переліченими вище типами – виводиться тип "Неправильний тип ").

Варіант 28

Аналіз типів аргументів, які задаються під час запуску програми. Якщо аргумент є правильним ідентифікатором мови С (шаблон: складається з латинських літер, цифр та символу і "_", що вважається буквою, і, крім того, перший символ є буквою), то його тип "Identifier", якщо параметр є ключовим словом С (для прикладу задати кілька ключових слів С, "if", "for", "while", "do" и "else "), то його тип "Keyword", інакше його тип вважається "Illegal". Програма виводить кількість заданих аргументів, для кожного аргументу, його тип і значення

Варіант 29

Аналіз типів аргументів, які задаються під час запуску програми. Якщо аргумент є числовим літералом, тобто починається з цифри, то визначається його тип ("Integer" або "Real"), якщо аргумент укладено в одиночні апострофи і містить один символ, то його тип – "Character", якщо аргумент укладений у подвійні апострофи, то його тип – "String". Якщо не одна з умов не виконується, то тип аргументу – "Identifier". Програма виводить кількість заданих аргументів, для кожного аргументу, його тип і значення.

Варіант 30

Аналіз аргументів, які задаються під час запуску програми. Програма розглядає аргументи, що вводяться (аргументи можуть містити пробіли) як імена файлів Windows і визначає тип файлу за його розширенням ("Програма" – розширення .exe або .com, "Документ Word" – розширення . doc або "Текстовий файл" – розширення .txt), а потім переводить ім'я файла

(яке в Windows довгим ім'ям) коротке ім'я (або ім'я MS DOS) за таким алгоритмом:

1) якщо є розширення файла, і воно містить більше трьох символів, в ньому залишаються перші три символи.

2) з імені файлу видаляються символи, неприпустимі в імені файла MS DOS (пробіли, одиночні й подвійні апострофи, а також символи "-", "_", "$", "#", "&", "@", "!", "%", "(", ")", "{", "}", "~", "." та "*").

3) якщо довжина одержаного імені менше або дорівнює 6 символів, одержане ім'я береться в якості імені файлу, інакше як ім'я файлу беруться перші 6 символів, після яких ставлять символи "~1".

Програма виводить кількість заданих аргументів, і типи файлів для кожного аргументу (якщо тип не збігається з переліченими вище типами – виводиться тип "Невірний тип ").

5. Приклад виконання завдання 4

Аналіз типів аргументів, що задаються під час запуску програми. Якщо аргумент є правильним прізвищем з ініціалами (шаблон: заголовна буква кирилиці, одна або кілька малих літер кирилиці, пробіл, заголовна буква, символ ".", Заголовна буква, символ "."). Програма виводить кількість вихідних аргументів, список вихідних аргументів, кількість правильних прізвищ, відсортованих за першою літерою прізвища і список неправильних аргументів.

Текст програми:

```
package stringsearch;

import java.util.regex.*;
public class Main {
// Оголошення масиву правильних аргументів
static String [] correctName;
// Оголошення масиву неправильних аргументів
static String [] wrongName;
// Правильні імена є
static boolean hasCorrectNames = true;
// Неправильних імен немає
static boolean hasWrongNames = false;
public Main () {
}
// Метод перевірки аргументів
public static void argsTest (String [] args) {
// Завдання шаблона для аргументу
Pattern namePattern =
Pattern.compile ( "^ [А-Я] [а-я] + [А-Я] \\. [А-Я] \\.");
// Завдання початкового кількості правильних прізвищ у рядку аргументів
```

```
int correctNameNumber = 0;
// Визначення кількості правильних аргументів у циклі
for (int i = 0; i <args.length; i ++) {
// Створення об'єкта класу Matcher для поточного аргументу
        Matcher nameMatcher = namePattern.matcher (args [i]);
// Якщо поточний аргумент - правильне прізвище
        if (nameMatcher.matches ())
// Збільшення кількості правильних прізвищ на 1
        correctNameNumber ++;
}
if (correctNameNumber == 0)        // Якщо правильних імен немає
hasCorrectNames = false;           // Установка перемикача
else
// Завдання розмірності масиву правильних імен
correctName = new String [correctNameNumber];
// Якщо не всі імена - правильні
if (correctNameNumber <args.length) {
hasWrongNames = true;       // Установка перемикача
 // Завдання розмірності масиву неправильних імен
wrongName = new String [args.length - correctNameNumber];
}
// Початковий індекс масиву правильних елементів
int correctArrayIndex = 0;
// Початковий індекс масиву неправильних елементів
int wrongArrayIndex = 0;
// Заповнення масивів правильних і неправильних елементів в циклі
for (int i = 0; i <args.length; i ++) {
// Створення об'єкта класу Matcher для поточного аргументу
Matcher nameMatcher = namePattern.matcher (args [i]);
// Якщо поточний аргумент - правильна прізвище
if (nameMatcher.matches ()) {
// Додавання поточного аргументу в масив правильних  аргументів
correctName [correctArrayIndex] = args [i];
// Збільшення індексу масиву  правильних аргументів на 1
correctArrayIndex ++;
}
else {
// Додавання поточного аргументу в масив неправильних аргументів
wrongName [wrongArrayIndex] = args [i];
// Збільшення індексу масиву неправильних аргументів на 1
wrongArrayIndex ++;
}
}
}
// Метод виведення аргументів
public static void argsPrint (String [] args) {
// Вивід кількості аргументів
System.out.println ( "Кількість аргументів:" + args.length);
// Вивід першого аргументу як рядку
```

```
String allArgs = args [0];
// Цикл по іншим аргументам
for (int i = 1; i <args.length; i ++)
       allArgs + = "," + args [i];
// Вивід вихідних аргументів
System.out.println ( "Вихідні аргументи: \ n" + allArgs);
// Якщо є правильні аргументи
if (hasCorrectNames) {
// Вивід заголовка
System.out.println ( "Кількість правильних прізвищ:" +
correctName.length);
// Виклик методу сортування прізвищ
correctName = nameSort (correctName);
// Вивід правильних прізвищ в циклі
for (int i = 0; i <correctName.length; i ++)
// Вивід поточного прізвища
       System.out.println ( "" + correctName [i]);
}
// Якщо є неправильні аргументи
if (hasWrongNames) {
 // Вивід заголовка
System.out.println ( "Неправильні аргументи:");
// Вивід неправильних аргументів у циклі
for (int i = 0; i <wrongName.length; i ++)
// Вивід поточного аргументу
System.out.println ( "" + wrongName [i]);
}
}
// Метод сортування прізвищ за першою літерою
public static String [] nameSort (String [] name) {
// Цикл по масиву прізвищ
for (int i = 0; i <name.length - 1; i ++) {
for (int j = i; j <name.length - 1; j ++) {
// Якщо код букви в ій позиції більше коду в (i + 1)й позиції
if (name [j] .charAt (0)> name [j + 1] .charAt (0)) {
String nameIValue = name [j];      // Запам'ятовування і-го прізвища
name [j] = name [j + 1];    // Перестановка місцями і-го та (i + 1)-го прізвищ
name [j + 1] = nameIValue; // Запам'ятовування (i + 1)-го прізвища
}
}
}
return name;                    // Повернення відсортованого масиву
}

public static void main (String [] args) {
if (args.length == 0) {            // Якщо не задані аргументи
// Виведення повідомлення про помилку
System.out.println ( "Не задані аргументи");
System.exit (1);                    // Вихід з програми
```

```
}
argsTest (args);              // Виклик методу перевірки аргументів
argsPrint (args);             // Виклик методу виведення аргументів
}
}
```

Приклад виведення:

Рядок аргументів:
"Сидоров С.С." "Іванов І.І." "Петров П.П." "Smith J." "Jones J."
Кількість аргументів: 5

Вихідні аргументи:
Сидоров С.С., Іванов І.І., Петров П.П., Smith J., Jones J.

Кількість правильних прізвищ: 3
Іванов І.І.
Петров П.П.
Сидоров С.С.

Неправильні аргументи:
Smith J.
Jones J.

6. Зміст звіту

Текст і вивід програми, а також рядок аргументів вікна Project Properties з аргументами виклику для свого варіанта.

7. Питання для самоконтролю

1. Як оголошуються та ініціалізуються рядки у мові Java? Які види параметрів можуть бути задані в конструкторі класу String?
2. Як визначається довжина рядка і як виконується конкатенація рядків в мові Java?
3. Як виконується розміщення рядка в об'єкті клас StringBuffer? Які конструктори визначені для класу StringBuffer?
4. Як визначається та встановлюється ємність буферної пам'яті та довжина рядка для об'єкта класу StringBuffer?
5. Які методи порівняння рядків визначені в класі String?
6. Які методи класу String виконують пошук в рядках?
7. Які методи вилучення рядків і підрядків рядка визначені в класі String?
8. Які методи модифікації рядків визначені в класі String?
9. Як в Java можна створити рядки з примітивних типів?

10. Які методи модифікації рядків визначені в класі StringBuffer?
11. Які типи символів можна задавати шаблон регулярного виразу?
12. Як визначається операція альтернації в шаблоні регулярного виразу?
13. Як задати будь-який одиночний символ шаблону регулярного виразу?
14. Як діють квантифікаторів "+", "*" та "?" в шаблоні регулярного виразу?
15. Як задається точне число повторень символу у шаблоні регулярного виразу?
16. Як визначається клас символів в шаблоні регулярного виразу?
17. Як задаються класи символів за допомогою спеціальних символів у шаблоні регулярного виразу?
18. Які анкери можна задати шаблон регулярного виразу?
19. Як задається групування символів і як захоплюються символи шаблону регулярного виразу?
20. Як задається групування символів без захоплення в шаблоні регулярного виразу?
21. Як задається групування символів із «загляданням уперед» у шаблоні регулярного виразу?
22. Як задається групування символів із «загляданням назад» у шаблоні регулярного виразу?
23. Що містить і як створюється об'єкт класу Pattern у Java?
24. Які прапорці та якими способами можна задати шаблон регулярного виразу в Java?
25. Які методи визначені для класу Pattern у Java та які дії вони виконують?
26. Що містить і як створюється об'єкт класу Match у Java?
27. Які операції з регіонами визначені в класі Match у Java та які дії вони виконують?
28. Які методи пошуку відповідності визначені в класі Match у Java та які дії вони виконують?
29. Які методи заміни визначені в класі Match у Java та які дії вони виконують?
30. В яких випадках генерується виключення класу PatternSyntaxException і які методи визначені в цьому класі?
31. Які методи роботи з регулярними виразами визначені в класі String у Java, та які дії вони виконують?
32. Який формат визначення методу у Java?
33. Як задається тип значення, що повертається у визначенні методу у Java?
34. Для яких цілей використовують пропозицію return у Java?
35. Як визначаються перевантажені методи у Java і як вони задаються?
36. Які модифікатори можна задати для методів у Java?

Електронне видання

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт
на тему «Робота з рядками в Java»
із дисциплін
«**Програмування мобільних комп'ютерних систем**»,
«**Прикладне програмування в телекомунікаційних системах**»,
«**Програмування мобільних пристроїв телекомунікацій**»
для студентів спеціальностей
171 «*Електроніка*»,
172 «*Телекомунікації та радіотехніка*»
денної форми навчання

Відповідальний за випуск А. С. Опанасюк
Редактор Н. М. Мажура
Комп'ютерне верстання О. Є. Горячева

Формат 60x84/16. Ум. друк. арк. 1,16. Обл.-вид. арк. 1,09.

Видавець і виготовлювач
Сумський державний університет,
вул. Римського-Корсакова, 2, м. Суми, 40007
Свідоцтво суб'єкта видавничої справи ДК № 3062 від 17.12.2007.