Лабораторная работа № 4

Тема: Строковые данные и классы работы с ними

Цель: Изучение методов классов строковых данных и их эффективности

Теоретическая часть

Строчные данные достаточно часто используются в разных программах. Для работы с этими данными используются классы String, StringBuffer и StringBuilder. Объекты класса String являются неизменяемыми (immutable), т.е. при изменении строки в действительности старая строка не изменяется, а создаётся новая. В отличие от класса String специальные классы StringBuffer и StringBuilder допускают изменения в исходной строке. Все три класса String, StringBuffer, StringBuilder определены в пакете java.lang и доступны автоматически без объявления импорта. В этих классах реализуют интерфейс CharSequence. Ниже приведены несколько примеров создания строчных данных различными методами.

Пример создания строки:

```
String aboutCity = "Zaporizhzhya is industrial city";
```

Пример создания массив строк:

```
String[] collaborators = {"student", "aspirant", "professor"};
```

Создание пустого объекта класса String:

```
String str = new String();
```

Создание строки через массив символов:

```
char[] chars = { 's', 't', 'u', 'd', 'e', 'n', 't'};
String str = new String(chars);
```

Всего в классе String имеется 13 конструкторов [https://docs.oracle.com/javase/8/docs/api/java/lang/String.html], которые различаются как источником данных, так и начальными параметрами образования объектов. Например, следующий конструктор, позволяет задать диапазон символьного массива, который используется для создания объекта, для чего указывается начало диапазона и количество символов для использования:

```
char[] chars = {'s', 't', 'u', 'd', 'e', 'n', 't'};
String str = new String(chars, 0, 4);
```

В результате строковая переменная str будет содержать значение «stud».

Также имеется два конструктора, которые позволяют создать объект класса String из объекта классов StringBuffer или StringBuilder:

```
String(StringBuffer объект_StrBuf)
String(StringBuilder объект_StrBuild)
```

Для строковых данных знак плюс (+) означает конкатенацию строк (объединение строк). Например,

```
String first_name = "Іван";
String second_name = "Сидоренко";
String full_name = first_name + " " + second_name;
// получится Іван Сидоренко. Пробел используется для разделения слов
```

В языке Java если один из операндов в выражении содержит строку, то другие операнды также должны быть строковыми. Поэтому компилятор Java приводит переменные к строковому представлению, даже если они не являются строками. Например,

```
int digit = 4;
String paws = " лапы";
String aboutcat = digit + paws;
// в результате сложения числа и строки получим строку «4 лапы»
```

В классе String имеется более 60 методов, которые позволяют получать свойства строчных данных, выполнять операции их изменения, сравнения, извлечения как их частей, так и отдельных символов и т. п. (https://docs.oracle.com/javase/8/docs/api/java/lang/String.html).

Пример использования методов класса String:

```
public class JavaApplication8 {
   public static void main(String[] args) {
      byte[] str_byte;
      char[] chars = {'s','t','u','d','e','n','t'};
      String HW = "Hello";
      str_byte = HW.getBytes();
      System.out.println("Hello - "+HW.length());
      for(int i=0;i<HW.length();i++){System.out.println("byte["+i+"]="+str_byte[i]);}
      String a = new String(chars,0,7);
      System.out.println(">> "+a);
      str_byte = a.getBytes();
      System.out.println("student - "+a.length());
      for(int i=0;i<a.length();i++){System.out.println("byte["+i+"]="+str_byte[i]);}
   }
}
```

Изучите работу примера и дайте пояснения используемым в примере методам.

В классе String предусмотрена возможность форматирования строки, т. е. её образование по определенному формату. Эта возможность реализуется при помощи метода format(), который имеет следующий синтаксис:

```
public static String format(String format, Object... args)
```

Метод возвращает форматированную строку в соответствии со строкой заданных форматов (String format) и аргументов (Object... args). Строка формата - это строка, которая может содержать фиксированный текст и один или несколько встроенных спецификаторов формата. Каждый спецификатор формата начинается со знака % и заменяется соответствующим аргументом. Завершает спецификатор формата - символ преобразования, который обозначает тип форматируемого значения: f — число с плавающей точкой, s — символьную строку, d — десятичное целое число. Общий синтаксис спецификатора формата:

```
%[argument_index$][flags][width][.precision]conversion
```

где argument_index — десятичное целое число, которое указывает позицию аргумента в списке аргументов. Первый аргумент обозначается как "1$", второй — "2$" и т. д.;

flags — набор символов, которые модифицируют вывод, например, флаг ‘+’ требует вывод чисел с указанием знака (для положительных ‘+’, для отрицательных ‘-’);

width — положительное целое число, которое указывает на минимальное количество символов, которое должно быть в выводе;

precision — количество знаков точности представления;

conversion — символ преобразования (см. табл. ниже).

Символ преобразования	Назначение	Пример
d	Десятичное целое число	234
x или X	Шестнадцатеричное целое число	8c, 8C
o	Восьмеричное целое число	237
f или F	Число с плавающей точкой фиксированной точности	14.3
e или E	Число с плавающей точкой в экспоненциальной форме	1.23e+01 2.345E-03
g или G	Число с плавающей точкой в общей форме	
a или A	Шестнадцатеричное число с плавающей точкой	0x1.fccdp3 0X1.FCCDP3

s или S	Символьная строка	Hello, WORLD
c или C	Символ	J, j
b или B	Логическое значение	False, FALSE True, TRUE
h или H	Хеш-код	42628b2
t или T	Дата и время (устаревшее)	
%	Знак процента	
n	Разделитель строк (зависит от конкретной платформы)	

Для обозначения представления форматов чисел используются — f, g, e. Выполните следующий пример, для того чтобы увидеть различия в этих форматах:

```
System.out.printf("%10.8g\n", -123.456);
System.out.printf("%10.8e\n", -123.456);
System.out.printf("%10.8f\n", -123.456);
System.out.println();
System.out.printf("%10.8g\n", -0.0000123456);
System.out.printf("%10.8e\n", -0.0000123456);
System.out.printf("%10.8f\n", -0.0000123456);
System.out.println();
System.out.printf("%10.8g\n", -123456.);
System.out.printf("%10.8e\n", -123456.);
System.out.printf("%10.8f\n", -123456.);
```

Форматированный вывод также позволяет организовывать строку для вывода. Например, строки могут быть подобными и различаться только в некоторой их части. Проанализируйте вывод строк следующего примера и дайте ему пояснения:

```
String strBarsik = "Барсик";
String strPaws = "четыре";
String strTail = "один";
int year = 5;
String strCat = "У кота по имени %1$s %2$s лапы, %3$s хвост. Ему %4$s лет.";
String strFinal = String.format(strCat, strBarsik, strPaws, strTail, year);
System.out.println(strFinal);
strBarsik = "Васька";
year = 6;
strFinal = String.format(strCat, strBarsik, strPaws, strTail, year);
System.out.println(strFinal);
```

Класс **String** представляет собой неизменяемые последовательности символов постоянной длины и частое использование объектов класса занимают много места в памяти. Класс **StringBuffer** представляет

расширяемые и доступные для изменений последовательности символов, позволяя вставлять символы и подстроки в существующую строку и в любом месте. Данный класс гораздо экономичнее в плане потребления памяти и настоятельно рекомендуется к использованию.

Существует четыре конструктора класса:

1. StringBuffer() - резервирует место под 16 символов без перераспределения памяти
2. StringBuffer(int capacity) - явно устанавливает размер буфера
3. StringBuffer(String string) - устанавливает начальное содержимое и резервирует 16 символов без повторого резервирования
4. StringBuffer(CharSequence cs) - создаёт объект, содержащий последовательность символов и резервирует место ещё под 16 символов

Методы класса StringBuffer

length() - метод, который позволяет получить текущую длину объекта.

```
StringBuffer sb = new StringBuffer("студент");
System.out.print("Длина: " + sb.length());
```

capacity() - метод, который позволяет получить текущий объём выделенной памяти.

```
StringBuffer sb = new StringBuffer("студент");
System.out.print("Объём памяти: " + sb.capacity());
```

Определите объем, который в памяти занимает слово.

ensureCapacity() - метод, который позволяет предварительно выделить место для определённого количества символов. Это позволяет ускорить операции при добавлении большого количества небольших строковых данных.

setLength(int length) - устанавливает длину строки, значение должно быть неотрицательным.

Можно извлечь значение отдельного символа с помощью метода **charAt(int index)** и установить новое значение символа с помощью метода **setCharAt(int index, char ch)**, указав индекс символа **index** и его значение **ch**.

```
StringBuffer sb = new StringBuffer("Кит");
sb.setCharAt(1, 'o');
System.out.print(sb.toString());
```

getChars() - метод, который позволяет скопировать подстроку из объекта класса StringBuffer в массив. Массив должен быть достаточного размера для приёма нужного количества символов указанной подстроки.

append() этот метод позволяет присоединить представление любого другого типа данных. Метод имеет несколько перегруженных версий:

```
StringBuffer append(String string)
StringBuffer append(int number)
StringBuffer append(Object object)
```

Например:

```
String str1 = "У кота ";
String str2 = " лапы";
int paws = 4;
StringBuffer sb = new StringBuffer(20);
sb.append(str1).append(paws).append(str2);
System.out.print(sb.toString());
```

Метод **insert()** вставляет одну строку в другую. Также можно вставлять значения других типов, которые будут автоматически преобразованы в строки. Вам надо указать индекс позиции, куда будет вставляться строка.

```
StringBuffer sb = new StringBuffer("Я в университете");
sb.insert(2, "учусь ");
System.out.print(sb.toString());
```

Метод **reverse()** позволяет изменить порядок символов на обратный.

```
StringBuffer sb = new StringBuffer("арбуз блокнот окно");
sb.reverse();
System.out.print(sb.toString());
```

Метод **delete(int index1, int index2)** удаляет последовательность символов, начиная с символа с индексом **index1** и до символа с индексом **index2-1**, т. е. символ с индексом **index2** уже не удаляется. Метод **deleteCharAt(int index)** удаляет один символ из указанной позиции.

Метод **replace(int index1, int index2, String str)** позволяет заменить один набор символов на другой. Нужно указать начальный и конечный индекс и строку замены.

```
StringBuffer sb = new StringBuffer("арбуз блокнот окно");
sb.replace(6, 13, "балкон");
System.out.println(sb.toString());
```

Метод **substring()** позволяет получить часть содержимого. Есть две формы метода. В первом случае позволяется указать индекс позиции, начиная с которой получается подстрока, во втором — указывается начальный и конечный индексы, и получается определенная часть текста строки.

```
StringBuffer sb = new StringBuffer("арбуз блокнот окно");
sb.replace(6, 13, "балкон");
System.out.println(sb.toString());
StringBuffer sb2 = new StringBuffer(sb.substring(6,13));
System.out.println(sb2.toString());
```

Еще один класс **StringBuilder** подобен классу **StringBuffer**, обладает большей производительностью, но он не синхронизирован. Поэтому его не следует использовать в тех случаях, когда к изменяемой строке обращаются несколько потоков.

Ниже приведен пример, позволяющий сравнить производительность методов конкатенации в классах **String, StringBuilder** и **StringBuffer** при создании строчных данных из случайных символов.

```
import java.util.Scanner;
/**
 * @author gorbenko
 */
public class ProbStringBuilder {

   public static void main(String[] args) {

      long t1,t2;
      int numsimb;
      Scanner sc=new Scanner(System.in);
      System.out.print("Какая должна быть длина последовательности? - ");
      numsimb=sc.nextInt();
      // Используем простой String
      String Key="";
      System.out.println(" Key = "+Key);
      t1=System.currentTimeMillis();
        for(int i=0; i<numsimb; i++) {
          Key+=(char)((int)(Math.random()*255));
        }
      t2=System.currentTimeMillis();
      System.out.println(" Key = "+Key);
      System.out.println("Время вычисления = "+(double)(t2-t1)/1000+" s");
      System.out.println("Длина строки = "+Key.length());

      // Используем StringBuilder
      StringBuilder Key2=new StringBuilder("");
      System.out.println(" Key = "+Key2);
```

```
        t1=System.currentTimeMillis();
          for(int i=0; i<numsimb; i++) {
            Key2.append((char)((int)(Math.random()*255)));
          }
        t2=System.currentTimeMillis();
        System.out.println(" Key = "+Key2.toString());
        System.out.println("Время вычисления = "+(double)(t2-t1)/1000+" s");
        System.out.println("Длина строки = "+Key2.length());

        // Используем StringBuffer
        StringBuffer Key3=new StringBuffer("");
        System.out.println(" Key = "+Key3);
        t1=System.currentTimeMillis();
          for(int i=0; i<numsimb; i++) {
            Key3.append((char)((int)(Math.random()*255)));
          }
        t2=System.currentTimeMillis();
        System.out.println(" Key = "+Key3.toString());
        System.out.println("Время вычисления = "+(double)(t2-t1)/1000+" s");
        System.out.println("Длина строки = "+Key3.length());
    }
}
```

Задания к работе

1. Изучите методы и примеры теоретической части лабораторной работы. Приведите примеры, демонстрирующие работу методов, сделайте скрин-шоты и дайте пояснения.

2. Изучите производительность методов конкатенации (последний пример теоретической части) на 10 значениях длины строк в диапазоне от 10000 до 100000 символов. Постройке графики зависимости времени выполнения конкатенации случайных символов в строку от ее длины. Проверьте зависит ли время выполнения методов конкатенации для объектов разных классов от последовательности их вызова.

3. Разработайте методы, которые позволяют выполнять сортировку для массива объектов классов String, StringBuffer, StringBuilder. Приведите примеры их работы и соответствующие скрин-шоты.

4. Подготовьте отчет.