СТРОКИ

Строка - это объект (ссылочный тип), представляющий собой упорядоченную последовательность символов.

В Java строки являются объектами класса java.lang.String.

Строки неизменны (immutable objects), можно только создать новую строку на основе существующей.

Перейти к описанию класса String в документации

До 9 версии Java строки хранятся в кодировке UTF-16 как массив char[] .
Каждый символ занимает 2 байта.

Начиная с 9 версии (компактные строки), строки хранятся как массив byte[] в кодировке LATIN-1 (символы занимают по 1му байту) или UTF-16 (символы занимают 2 байта). При этом класс String хранит информацию о кодировке.

Создание строк

Для создания новой строки можно использовать один из конструкторов класса String, либо напрямую присвоить переменной ссылку на значение в двойных кавычках.

```
1 String poolMessage = "Строка добавлена в пул строк";
2 String heapMessage = new String("Строка находится в 'куче'");
```

Пул строк - список объектов класса String.

Значение poolMessage попало в пул. Если аналогично создать другую строку с тем же значением, произойдет поиск по пулу строк. Если там уже найдена такая строка, то второй переменной присваивается ссылка на уже созданный объект (на который указывает и poolMessage). При этом сравнение ссылок вернет true результат.

```
1 String poolMessage = "Строка добавлена в пул строк"; // строка попала
2 // вернулась ссылка на существующий объект
3 String poolMessage2 = "Строка добавлена в пул строк";
4 System.out.println(poolMessage == poolMessage2); // true
```

Если же создавать строки через вызов конструктра с использованием оператора new, строки не попадают в пул. Каждый раз происходит создание нового объекта. Создание new String("Строка находится в 'куче'"); приведет к созданию еще одного объетка. Сравнение ссылок вернет false результат.

```
1 // ссылка на первый объект
2 String heapMessage = new String("Строка находится в 'куче'");
3 // ссылка на второй объект
4 String heapMessage2 = new String("Строка находится в 'куче'");
5 System.out.println(heapMessage == heapMessage2); // false
```

Вычисление строк во время выполнения

Строки, вычисленные путем конкатенации во время выполнения, создаются заново и поэтому различаются.

```
1 String name = "Алексей";
2 String surname = "Петров";
3 System.out.println(name + " " + surname == "Алексей Петров"); // false
```

Вычисление строк во время компиляции

Строки, вычисленные с помощью константных выражений, вычисляются во время компиляции, а затем обрабатываются так, как если бы они были литералами.

```
1 String fullName1 = "Алексей Петров";
2 String fullName2 = "Алексей" + " " + "Петров";
3 System.out.println(fullName1 == fullName2); // true
```

Методы строк

Многие методы строк принимают в качестве параметра или возвращают в качестве результата индекс символа. Индексация начинается с нуля. Последний элемент имеет индекс - length() — 1. Если переданный в параметр индекс выходит за пределы строки, то методы генерируют исключение java.lang.IndexOutOfBoundsException.

Некоторые методы принимают в качестве параметров начальный индекс и конечный индекс. В этом случае начальный индекс включается в отрезок, а конечный индекс исключается.

Методы equals() / equalsIgnoreCase()

Сравнение строк

```
firstString.equals(String secondString)
firstString.equalsIgnoreCase(String secondString)
```

Сравнивают строку firstString со строкой secondString сначала по размеру,
если он одинаковый - посимвольно.
Метод equalsIgnoreCase сравнивает без учета регистра.
Возвращают true или false.

Методы compareTo() / compareToIgnoreCase()

Сравнение строк

```
firstString.compareTo(String secondString)
firstString.compareToIgnoreCase(String secondString)
```

Сравнивают firstString с secondString в лексикографическом порядке (номера Unicode).
Возвращают значение типа int.
Положительное - firstString больше secondString.
Отрицательное - firstString меньше secondString.
0 - строки равны.

Часто используемые методы строк

startsWith(String start) / endsWith(String end) - сравнение начала / конца строки

substring(int beginIndex) / substring(int beginIndex, int endIndex) - возвращают подстроку, начинающуюся с beginIndex до endIndex или конца строки

split(String regex) - разбивает строку на массив строк по регулярному выражению

contains(String s) - ищет подстроку s в строке, возвращает true или false

trim() - возвращает строку, в которой убраны пробелы в начале и в конце строки

replaceAll(String regex, String replacement) - возвращает строку, в которой все вхождения подстрок согласно регулярному выражению заменены на replacement

```java
// замена replace / replaceAll
String courseName = "Курс Java Junior Developer";
courseName = courseName.replace("Java", "Python");
System.out.println(courseName); // Курс Python Junior Developer

courseName = "Курс Java Junior Developer";
courseName = courseName.replaceAll("\\s", "_");
System.out.println(courseName); // Курс_Java_Junior_Developer

// метод split: массив из строки
String namesStr = "Kotlin, Java, Python";
String[] names = namesStr.split(", ");
System.out.println(Arrays.toString(names)); // [Kotlin, Java, Python]

// метод join: строка из массива
namesStr = String.join("! ", names);
System.out.println(namesStr); // Kotlin! Java! Python
```

Конкатенация строк

```java
String userName = "Александр";
String userSurname = "Петров";

// 1 вариант: +
String concatStr = userName + " :: " + userSurname;
System.out.println(concatStr); // Александр :: Петров

// 2 вариант: строка.concat()
concatStr = userName.concat(" :: ").concat(userSurname);
System.out.println(concatStr); // Александр :: Петров

// 3 вариант: String.join(раздельтель, строка1, строка2, строкаN);
concatStr = String.join(" :: ", userName, userSurname);
System.out.println(concatStr); // Александр :: Петров
```

Классы StringBuffer и StringBuilder

Класс StringBuffer представляет расширяемые и доступные для изменений последовательности символов, позволяет вставлять символы и подстроки в любое место существующей строки.

Класс StringBuilder идентичен классу StringBuffer и обладает большей производительностью. Однако он не синхронизирован, поэтому его не нужно использовать в тех случаях, когда к изменяемой строке обращаются несколько потоков.

Методы классов StringBuffer / StringBuilder предпочтительнее использовать для конкатенации в циклах

```
 1 String str2 = "Begin";
 2 StringBuilder sb = new StringBuilder();
 3 sb.append(str2);
 4 for (int i = 0; i < 1000; i++) {
 5     // на каждой итерации не будет создаваться новый объект
 6     sb.append(" Промежуток ").append(i).append(" ");
 7 }
 8 sb.append(" Окончание ");
 9 // StringBuffer / StringBuilder создают строку
10 // только при вызове метода toString()
11 System.out.println(sb.toString());
```