Программирование на языке Java

Тема 30. Кодирование символов

Кодирование символов

Текстовый файл

- на экране (символы)
- в памяти – двоичные коды

1000001_2	1000010_2	1000011_2	1000100_2
65	66	67	68

В файле хранятся не изображения символов, а их числовые коды в двоичной системе!

А где же хранятся изображения?

Кодирование символов

1. **Сколько символов** надо использовать одновременно? **256** или 65536 (UNICODE)
2. **Сколько места** надо выделить **на символ**:

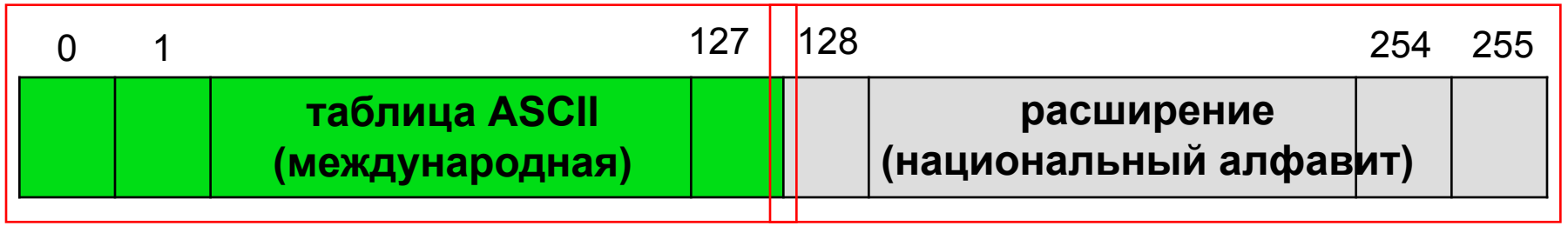
256 = 2^8 ⇒ 8 бит на символ

3. Выбрать **256 любых символов** (или 65536) - **алфавит**.
4. Каждому символу – **уникальный код 0..255** (или 0..65535). Таблица символов:

коды	65	66	67	68		
…	A	B	C	D	…	

5. Коды – в **двоичную систему**.

Кодировка 1 байт на символ



ASCII = *American Standard Code for Information Interchange*

0-31 управляющие символы:

7 – звонок, 10 – новая строка, 13 – возврат каретки, 27 – Esc.

32 пробел

знаки препинания: `. , : ; ! ?`

специальные знаки: `+ - * / () {} []`

48-57 цифры **0..9**

65-90 заглавные латинские буквы **A-Z**

97-122 строчные латинские буквы **a-z**

Кодовая страница (расширенная таблица ASCII)

для русского языка:

CP-866 для системы *MS DOS*

CP-1251 для системы *Windows* (Интернет)

КОИ8-Р для системы *UNIX* (Интернет)

Кодировка UNICODE (UTF-16)

- ***Java***
- **16 бит на символ**
- **65536** или 2^{16} символов в одной таблице

можно одновременно использовать символы разных языков (Интернет)

размер файла увеличивается **в 2 раза**

Программирование на языке Java

Тема 31. Символьный тип данных

Символьный тип данных `char`

Символ – минимальная единица текстовой информации.

Примеры символов: @, u, 8, ф, ®, µ

Символьный тип данных **`char`** относится к целочисленному типу без знака, т.к. в памяти хранится не сам символ, а его код в кодировке Unicode.

Символьный тип данных `char`

Особенности `char`

- является единственным беззнаковым примитивным типом данных;
- имеет длину 2 байта (65536 символов);
- задается с помощью конструкции `\u` и 4 шестнадцатеричных цифр от `'\u0000'` до `'\uFFFF'`;
- также можно задать с помощью символа, заключенного в одинарные кавычки, например `'A'`;
- инициализировать **`char`** можно и как символьный, и как численный литерал;
- может участвовать в арифметических действиях.

Пример

Значения переменных совпадают

```
char letter1 = '\u0041';
char letter2 = 'A';
letter1 == letter2 // true
```

Пример

```
public static void main(String args[]) {
   char a = '\u00B5';
      char b = '\u00A4';
      System.out.printf("%c%c\n", a, b);
}
```

```
µ¥
```

%c – спецификатор формата символьного типа char

Инкремент/декремент и `char`

Операторы инкремента и декремента можно применять к переменным типа **`char`** для получения следующего или предыдущего символы в кодировке Unicode.

```
char letter = 'a';
System.out.println(++letter);
```

Выведет символ b

Примеры

```
char c1 = 10, char c2;
c2 = 'A'; // латинская буква A
          // (‘\u0041’, код 65)
int i = c1 + c2 - 'B';
```

9

65

66

```
char c = 'A';
print(c);
print(c + 1);
print("c=" + c);
print('c' + '=' + c);
```

A

66

c=A

225

Код ‘c’ (99) + код ‘=’(61) + код ‘A’ (65)=225

Пример

Задача. Вывести на экран код символа ‘Y’.

```
public static void main(String args[]) {
   char ch1 = 'Y';
System.out.print((int)ch1);
}
```

Пример

Что будет выведено на экране?

```
public static void main(String args[]) {
   char ch = 'd';
   ch -= 3;
   System.out.println(ch);
}
```

a

Специальные символы

Экранированный символ	Описание	Unicode
`\b`	Возврат	`\u0008`
`\t`	Табуляция	`\u0009`
`\n`	Перевод строки	`\u000A`
`\r`	Возврат каретки	`\u000D`
`\\`	Обратный слеш	`\u005C`
`\"`	Двойная кавычка	`\u0022`

Чем плох массив символов?

Это массив символов:

```
char A[] = {'A', 'B', 'C', 'D'};
```

Для массива:

- каждый символ – отдельный объект;
- массив имеет длину N, которая задана при объявлении, ее нельзя изменить.

Что нужно:

- обрабатывать последовательность символов как единое целое;
- строка должна иметь переменную длину

Программирование на языке Java

Тема 32. Строки

Объекты и классы

Объекты – это сущности, которые содержат данные (переменные) и поведение (методы).

Класс – тип объекта.

Примеры:

- класс `String` представляет объекты, в которых хранится текст;
- класс `Scanner` представляет объекты, предназначенные для чтения данных из консоли, файла и других источников.

Объект String

String – объекты, в которых хранится строка, состоящая из символов.

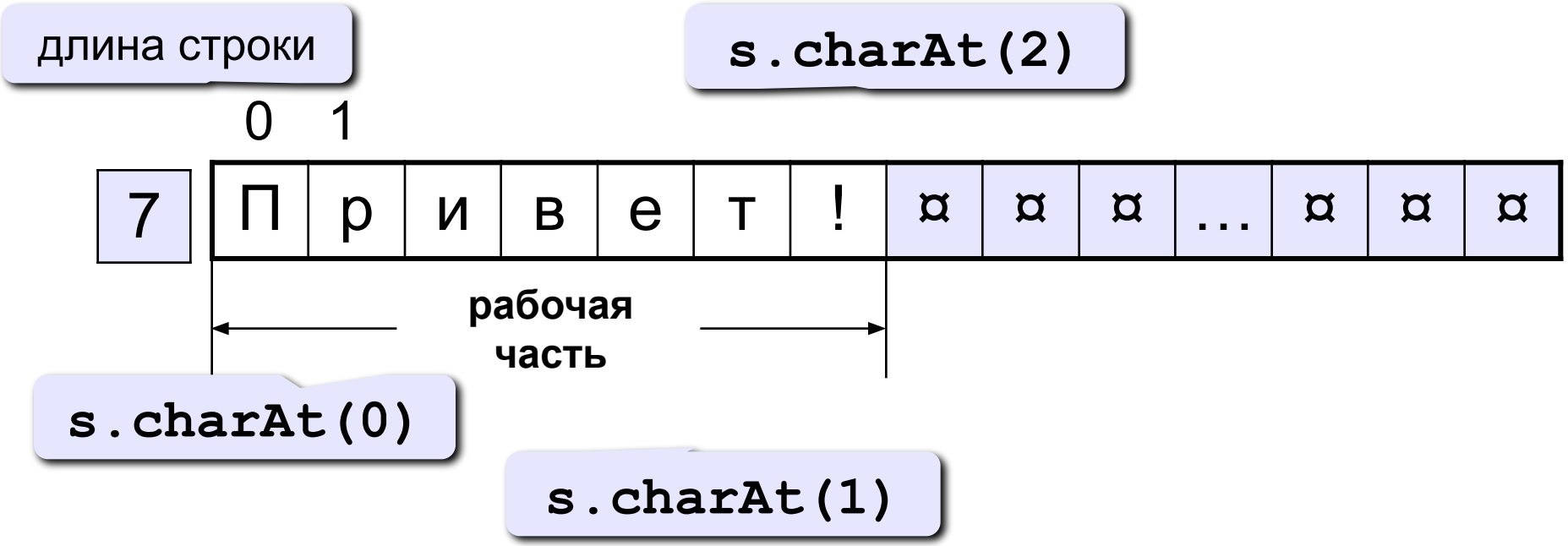
В отличие от большинства других объектов строку можно создать без оператора **new**.

Для обозначения строк используются двойные кавычки

```
String s = new String("Привет!");
```

```
String s = "Привет!";
```

Символьные строки



Длина строки:

```
int n = s.length();
```

Ввод строки с клавиатуры. Кодировка

Для ввода кириллических символов с клавиатуры в IDE NetBeans потребуется указать кодировку ввода.

Вместо

```
Scanner in = new Scanner (System.in);
```

Укажем

```
Scanner in = new Scanner (System.in, "cp1251");
```

Кодировка windows-1251

Ввод строки с клавиатуры. Методы

Для ввода строки используются методы:

- `next()` – возвращает последовательность символов до специального символа-разделителя (пробел, табуляция, перевод строки и т.п);
- `nextLine()` – возвращает следующую строку ввода до символа перевод строки (`\n`).

Ввод строки с клавиатуры. Пример

```
Scanner in = new Scanner (System.in);
System.out.println("Введите строку 1: ");
String str1 = in.nextLine();
System.out.printf("%s\n", str1);
System.out.println("Введите строку 2: ");
String str2 = in.next();
System.out.printf("%s\n", str2);
```

```
Введите строку 1:
фыва олдж
фыва олдж
Введите строку 2:
фыва олдж
фыва
```

Ввод символа с клавиатуры

```
System.out.print("Введите символ: ");
char ch = in.next().charAt(0);
System.out.printf("%c\n", ch);
```

Символьные строки

Задача: ввести строку с клавиатуры и подсчитать сколько букв «л» в нее входит.

ввод строки

```
System.out.print("Введите строку");
String s = in.nextLine();
int count = 0;
for (int i = 0; i < s.length(); i++)
   if (s.charAt(i) == 'л')
   count++;
System.out.println(count);
```

длина строки

Задания

Задача: **Ввести символьную строку и проверить совпадает ли первый символ строки и последним.**

Пример:

```
Введите строку:
АБВГДЕ
Результат:
Не совпадают.
```

Пример:

```
Введите строку:
КАЗАК
Результат:
Совпадают.
```

Задания

Задача: Ввести символьную строку и проверить, является ли она палиндромом (палиндром читается одинаково в обоих направлениях).

Пример:

```
Введите строку:
АБВГДЕ
Результат:
Не палиндром.
```

Пример:

```
Введите строку:
КАЗАК
Результат:
Палиндром.
```

Конкатенация строк

Для объединения двух строк используется метод `concat()` или оператор `+`.

```
String s1 = "abc";
String s2 = "123";
String s3 = s1.concat(s2);
String s4 = s2.concat(s1);
```

abc123

123abc

То же самое с использованием оператора `+`.

```
String s1 = "abc";
String s2 = "123";
String s3 = s1 + s2;
String s4 = s2 + s1;
```

Эквивалентность объектов

Оператор `==` сравнивает объекты по ссылке, поэтому часто такое сравнение возвращает значение `false`, даже если, например, строки состоят из одних и тех же символов.

Объекты нужно сравнивать с помощью метода `equals`.

Технически это метод, который возвращает значение типа `boolean` (логический метод), и для каждого типа объектов выполняет свои проверки.

Логические методы для строк

`equals(String s)` – полное сравнение строк

`equalsIgnoreCase(String s)` – сравнение, игнорирующее регистр символов

`startsWith(String s)` – проверка, что строка начинается с символов другой строки

`endsWith(String s)` – проверка, что строка заканчивается символами другой строки

`contains(String s)` – проверка, что строка содержит другую подстроку

Эквивалентность строк – 1

Задача: дано две строки, нужно проверить совпадают ли они.

```
String s1 = "abc";
String s2 = new String("abc");
if (s1 == s2)
   print("Строки совпадают");
```

Условие никогда не выполнится

Эквивалентность строк – 2

Задача: дано две строки, нужно проверить совпадают ли они.

```
String s1 = "abc";
String s2 = new String("abc");
String s3 = "AbC";
println (s1.equals(s2));
println (s1.equals(s3));
println (s1.equalsIgnoreCase(s3));
```

true

false

true

Сравнение строк – 1

Одна строка меньше другой строки, если она расположена перед ней в лексикографическом порядке.

“2 be or to B”

“2 B be or to”

Для сравнения строк используется метод

`compareTo()` – возвращает числовое значение, которое трактуется по следующим правилам:

<0 – вызывающая строка меньше строки из параметра

>0 – вызывающая строка больше строки из параметра

=0 – строки эквивалентны

Сравнение строк – 2

```
String s1 = “abc”;
String s2 = “AbC”;
print (s1.compareTo(s2));
```

32

Получение подстроки – 1

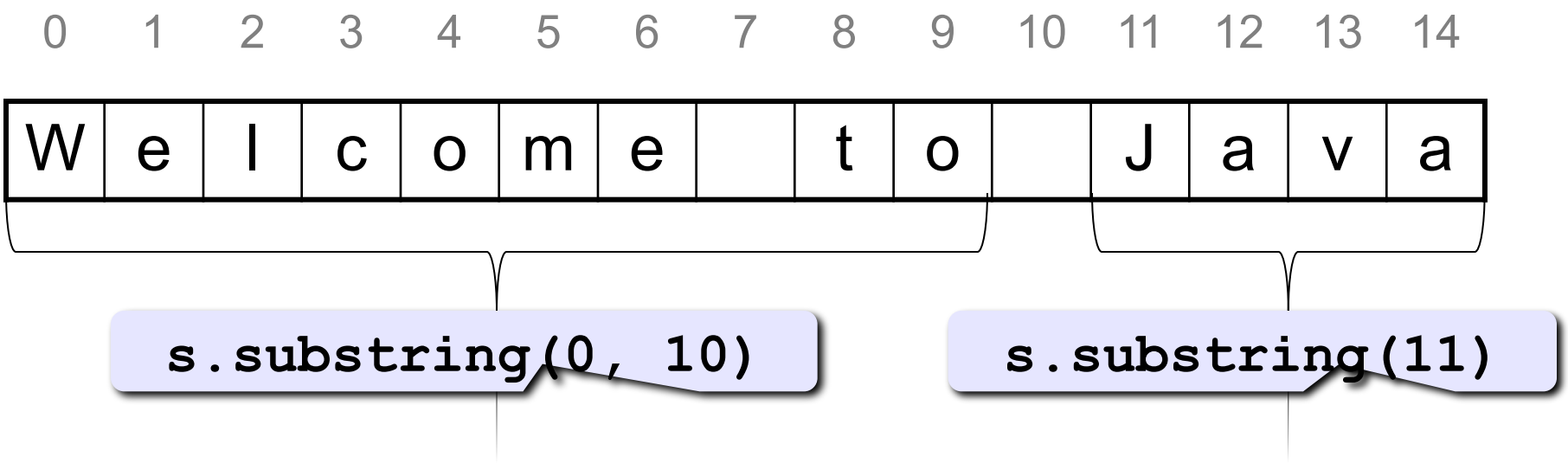
Получить подстроку некоторой строки можно с помощью метода **`substring()`**. Две формы вызова:

`substring(int beginIndex)` – возвращает подстроку, начинающуюся с индекса **`beginIndex`** и до конца строки.

`substring(int beginIndex, int endIndex)` – возвращает подстроку, начинающуюся с индекса **`beginIndex`** и заканчивающуюся в **`endIndex`**.

Получение подстроки – 2

```
String s = "Welcome to Java";
print (s.substring(0, 10));
print (s.substring(11));
```



Преобразование строк

`toLowerCase()` – возвращает новую строку, в которой все символы преобразованы к нижнему регистру;

`toUpperCase()` – возвращает новую строку, в которой все символы преобразованы к верхнему регистру;

`trim()` – возвращает новую строку, в которой удалены все начальные и конечные пробельные символы.

```
String s = "\t Welcome \n";
print (s.toLowerCase());
print (s.toUpperCase());
print (s.trim());
```

"\t welcome \n"

"\t WELCOME \n"

"Welcome"

Преобразование строк

Методы преобразований создают и возвращают новую строку, а не изменяют текущую

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

"Welcome"

Чтобы изменить переменную, нужно использовать оператор присваивания

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

"WELCOME"

Замена строк

`replace(char old, char new)` – возвращает новую строку, в которой все символы **`old`** заменены на символ **`new`**;

`replaceAll(String old, String new)` – возвращает новую строку, в которой все подстроки **`old`** заменены на подстроку **`new`**.

```
String s = "Welcome";
print (s.replace('e', 'A'));
print (s.replaceAll("el", "AB"));
```

"WAlcomA"

"WABcome"

Разбиение на подстроки

split(String delimiter) – возвращает массив строк, содержащих подстроки «разбитые» по разделителю **delimiter**.

```
String s = "Java#HTML#CSS";
String[] tokens = s.split("#");
for (int i = 0; i < tokens.length; i++)
    println(tokens[i]);
```

```
Java
HTML
CSS
```

Поиск подстрок – 1

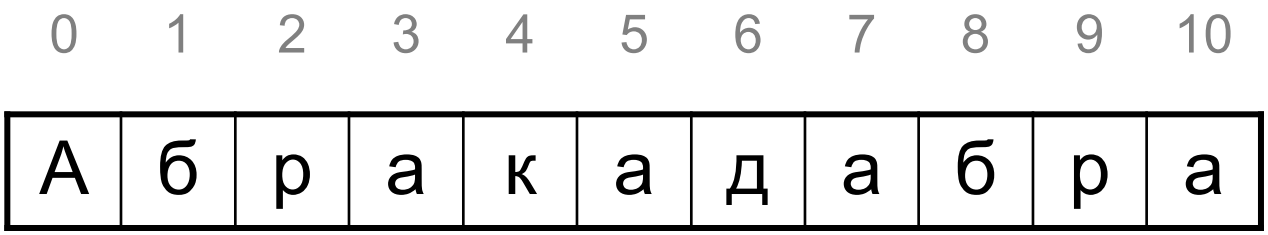
Методы, которые позволяют выполнить поиск в строке определенного символа или подстроки:

`indexOf()` – ищет первое вхождение символа или подстроки

`lastIndexOf()` – ищет последнее вхождение символа или подстроки

Оба метода возвращают позицию в строке, где символ или подстрока были найдены, либо значение -1 в случае неудачи.

Поиск подстрок – 2




```
String s = “Абракадабра”;
print (s.indexOf(‘a’));
print (s.lastIndexOf(‘a’));
print (s.indexOf(“ра”));
print (s.lastIndexOf(“ра”));
print (s.indexOf(“тра”));
```

Строки как параметры методов

```
public static void main(String[] args) {
    sayHello(“Alice");
    String person = “Bob";
    sayHello(person);
}
public static void sayHello(String name) {
    System.out.println(“Hello, " + name);
}
```

char vs. String

'a' – это символ (**char**), "a" – это строка (**String**)

String – это **объект**, он содержит методы:

```
String s = "h";
s = s.toUpperCase();          // 'H'
char first = s.charAt(0);     // 'H'
```

char – **примитивный**, от него нельзя вызвать методы

```
char c = 'h';
c = c.toUpperCase();  // ERROR
```

```
print(s + 1);
print(c + 1);
print(s + s);
print(c + c);
```

Задача – 1

Задача. В текстовом файле data.txt представлены данные об абитуриентах в следующем формате:

id Фамилия Имя Отчество Балл1 Балл2 Балл3

```
56 Иванов Иван Иванович 69 40 78
23 Петров Александр Петрович 79 90 88
```

Вывести данные из файла на консоль в виде таблицы

```
+---+-------------+----+----+----+
| id|     ФИО     | Б1 | Б2 | Б3 |
+---+-------------+----+----+----+
| 56|Иванов И.И.  |  69|  40|  78|
+---+-------------+----+----+----+
```

Задача – 2

Задача. Шифр Цезаря – простой способ шифрования, при котором каждый символ в открытом тексте заменяется символом, находящимся на некотором постоянном числе позиций левее или правее него в алфавите.

Например, при сдвиге 3 получаем

АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ

ГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВ

Напишите метод, который принимает в качестве параметров исходное сообщение и сдвиг, а возвращает сообщение, зашифрованное шифром Цезаря. При этом символы, отличные от кириллических, оставить без изменения.

Метод `format`

Класс **`String`** включает статический метод **`format`** для создания форматированных строк

```
String.format(<форматная_строка>,
   парметр1, ..., параметрk)
```

```
String s = String.format(
        "%7.2f%6d%-4s", 45.556, 14, "AB");
System.out.println(s);
```

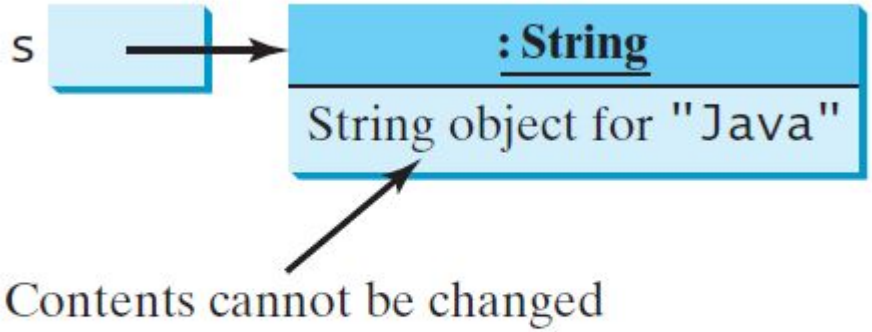
Неизменяемые строки

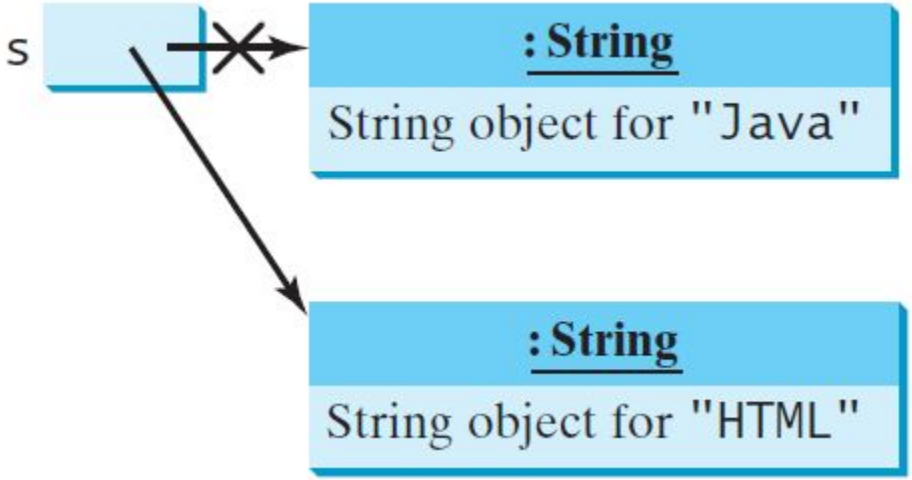
Класс **`String`** представляет **неизменяемые (immutable)** строки, т.е. их содержимое нельзя изменить.

Нет

Изменит ли следующий код содержимое строки?

```
String s = "Java";
s = "HTML";
```





Классы `String` и `StringBuffer`

Класс **`String`** представляет неизменяемые последовательности символов постоянной длины.

Если нужно работать с **изменяемыми строками (mutable)** , можно воспользоваться классом **`StringBuffer`**.

```
StringBuffer s = new StringBuffer("abc");
```

Работа с символами в StringBuffer

Для получения доступа к символу используется метод `charAt()`.

Для установки значения символа – метод `setCharAt()`

```
StringBuffer s = new StringBuffer("abc");
println(s.charAt(0));
s.setCharAt(0, '1');
println(s.charAt(0));
println(s);
```

```
a
1
1bc
```