JAVA PROGRAMMING BASICS

Module 2: Java Object-oriented Programming

Training program

1. Classes and Instances
2. The Methods
3. The Constructors
4. Static Elements
5. Initialization sections
6. Package
7. Inheritance and Polymorphism
8. Abstract classes and Interfaces
9. **String processing**
10. Wrapper classes for primitive types
11. Exceptions and Assertions
12. Nested classes
13. Enums
14. Generics
15. Collections
16. Method overload resolution
17. Multithreads
18. Core Java classes
19. Object Oriented Design
20. Functional Programming

Module contents

- String processing
  - The String class
  - Operations with Strings
  - Immutable String in Java
  - The StringBuilder and StringBuffer
  - String pool in Java
  - The StringTokenizer class
  - The Regular Expressions in Java
  - The Scanner class

Module contents

- String processing
  - The String class
  - Operations with Strings
  - Immutable String in Java
  - The StringBuilder and StringBuffer
  - String pool in Java
  - The StringTokenizer class
  - The Regular Expressions in Java
  - The Scanner class

The String class 1/4

- Strings, which are widely used in Java programming, are a sequence of characters.
- In the Java programming language, strings are objects.
- The Java platform provides the String class to create and manipulate strings.

- String greeting = **"Hello!"**;

String s5 = "\u041f\u0440\u0438\u0432\u0435\u0442\u01fe"; //ПриветǾ

Text Blocks

- Text block starts and ends with a “”” (three double-quote marks) followed by optional whitespaces and a newline.
- Inside the text block, we can freely use newlines and quotes **without the need for escaping line breaks**.

```
String page = """
        <html>
        <head>
        <meta charset="UTF-8">
        <title>Simple Servlet</title>
        </head>
        <body>
        <h2>Simple Servlet at %s</h2>
        <br/>%s
        </body>
        </html>
        """
```

New line required

Quote escape does not required

Since JDK 15

The String class 2/4

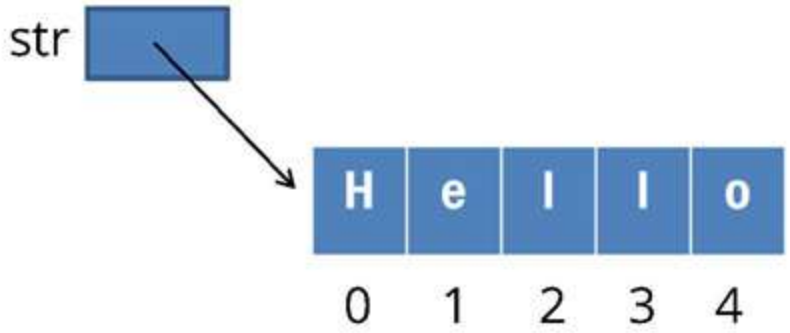
- String constructors:

1. String()
2. String(String value)
3. String(char[] value)
4. String(byte[] bytes)

- String s1 = **new** String();
- String s2 = **new** String(**"Hello"**);

The String class 3/4

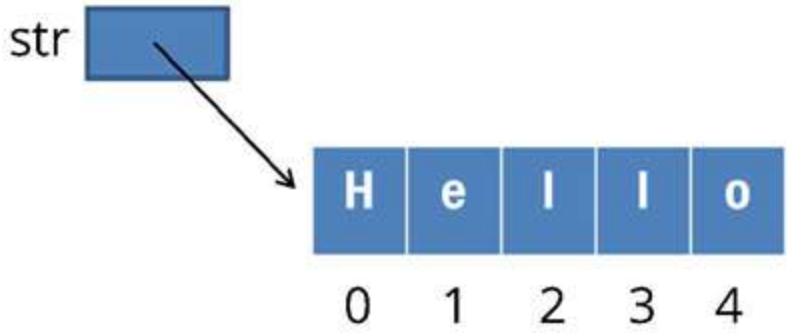
```
1. public static void main(String[] arg) {
2.     char[] helloArray = {'H', 'e', 'l', 'l', '\u0048'};
3.     String str = new String(helloArray);
4.     System.out.println(str);
5. }
```



String(**char[]** value, **int** offset, **int** count);

The String class 3/4

```
1. public static void main(String[] arg) {
                        throws UnsupportedEncodingException {
2.     byte [] helloArray = {0x48, 0x65, 0x6C, 0x6C, 0x6F};
3.     String str = new String(helloArray, "UTF-8" );
4.     System.out.println(str);
5. }
```



The String class 4/4

1. The length() method returns the number of characters contained in the string object

```
2. public static void main(String[] arg) {
3.     String str = "Hello World!!!";
4.     int len = str.length();
5.     System.out.println("String Length is : " + len);
6. }
```

Console output:
String Length is : 14

See StringInit

The String class

Before Java 9

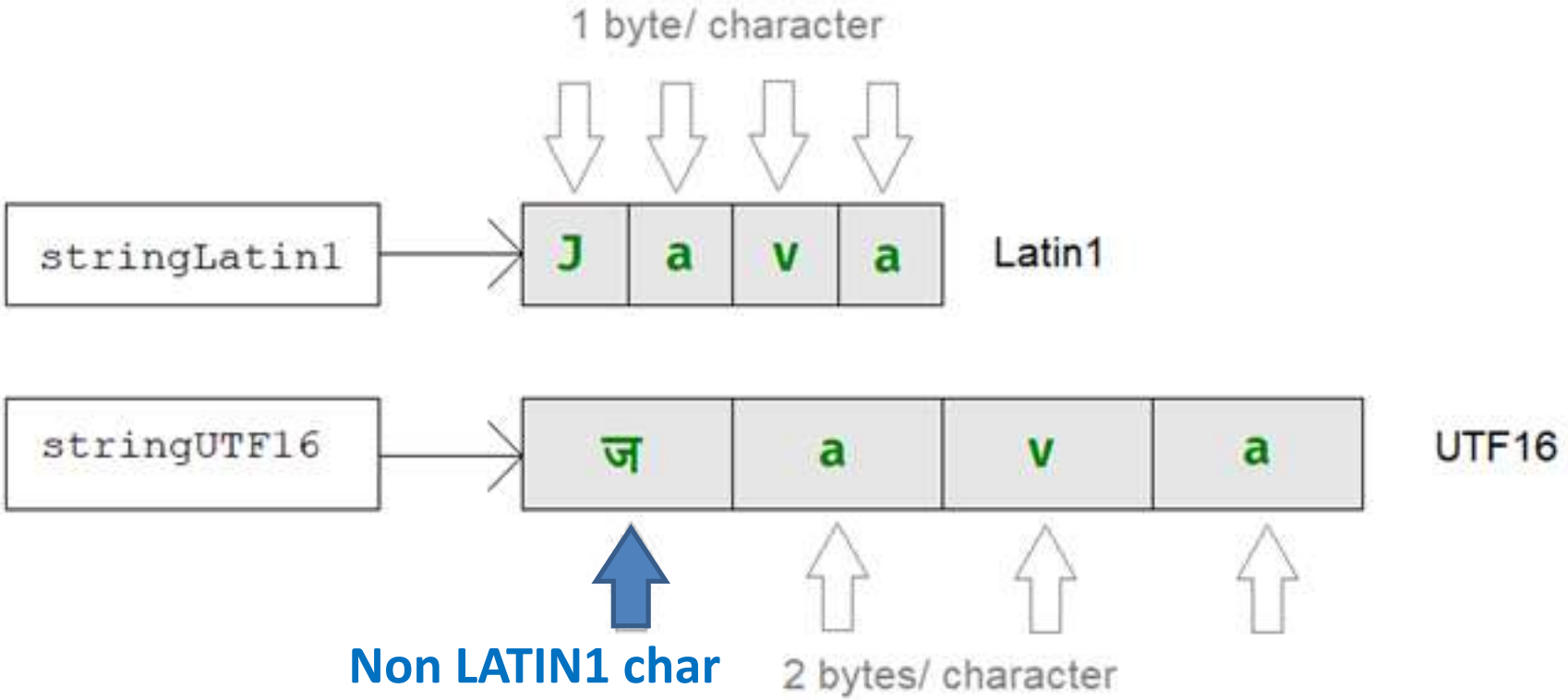
```
public final class String implements java.io.Serializable,
                                     Comparable<String>, CharSequence {

private final char value[];        //2 bytes per character
...
```

Since Java 9 - **Compact Strings**

```
public final class String implements java.io.Serializable,
Comparable<String>, CharSequence, Constable, ConstantDesc {
@Stable
private final byte[] value;        //1-2 bytes per character
/*The identifier of the encoding used to encode the bytes in value.
The supported values in this implementation are LATIN1 and UTF16*/
private final byte coder;          //LATIN1 = 0, UTF16 = 1
...
```

`String` are a major component of heap usage (approx. 20%) and *most* `String` objects contain only ISO-8859-1 or Latin-1 characters.

The String class



With Compact Strings, only the implementation details have changed for class String starting Java 9. **There are no API changes for class String.**

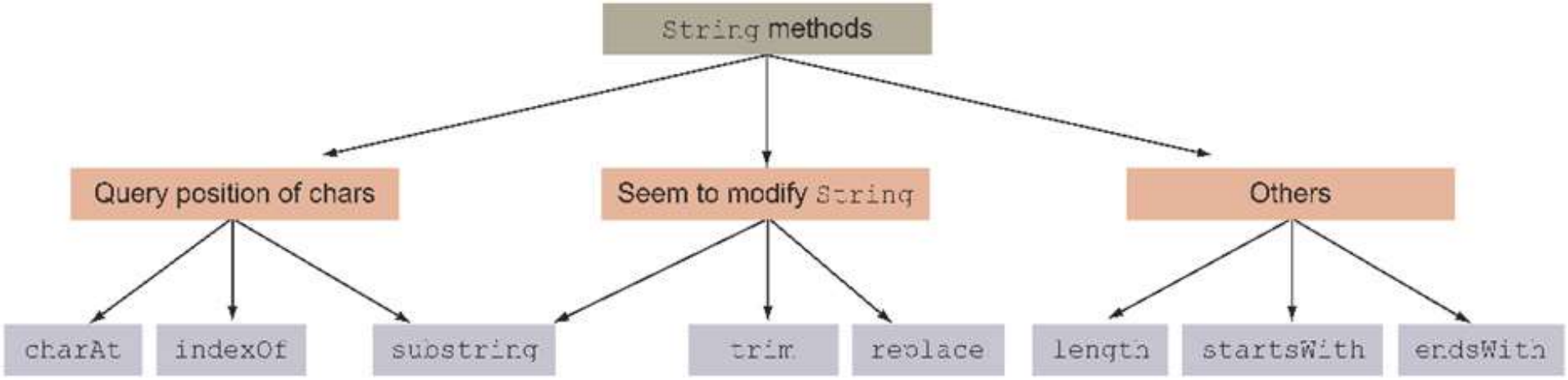
See StringInit

Module contents

- String processing
  - The String class
  - Operations with Strings
  - Immutable String in Java
  - The StringBuilder and StringBuffer
  - String pool in Java
  - The StringTokenizer class
  - The Regular Expressions in Java
  - The Scanner class

Operations with Strings 1/6

- The String class has a number of methods for examining the contents of strings, finding characters or substrings within a string, changing case, and other tasks.



String class basic methods
- Query position of chars

```
char charAt(int index)
int indexOf(String str)
int indexOf(String str, int fromIndex)
int lastIndexOf(String str)
int lastIndexOf(String str, int fromIndex)
int length()
```

String class basic methods
- Seem to modify String

String concat(String str)
String replace(**char** oldChar, **char** newChar)
String replace(CharSequence target,
CharSequence replacement)
String replaceAll(String regex, String replacement)
String substring(**int** beginIndex)
String substring(**int** beginIndex, **int** endIndex)
String repeat(**int** count)
String[] split(String regex)
String intern()
...

The methods return a NEW string because of String object is immutable

String class basic methods
- Seem to modify String

...
String toLowerCase()
String toString()
String toUpperCase()
String trim()

```
public final class String
   implements java.io.Serializable, Comparable<String>, CharSequence,
              Constable, ConstantDesc {

  private final byte[] value;
...
}
```

The methods return a NEW string because of String object is immutable

String class basic methods - Other

boolean isBlank()
boolean isEmpty()
boolean matches(String regex)
boolean endsWith(String suffix)
boolean equals(Object anObject)
boolean equalsIgnoreCase(String anotherString)
boolean startsWith(String prefix)
boolean startsWith(String prefix, **int** toffset)
...

String class basic methods
- Other

...

```
int compareTo(String anotherString)
int compareToIgnoreCase(String str)
char[] toCharArray()
void getChars(int srcBegin, int srcEnd,
                    char[] dst, int dstBegin)
byte[] getBytes(String charsetName)
static String valueOf(primitive data type x)
```

See StringOperation

String methods chaining

2 whitespaces at the end

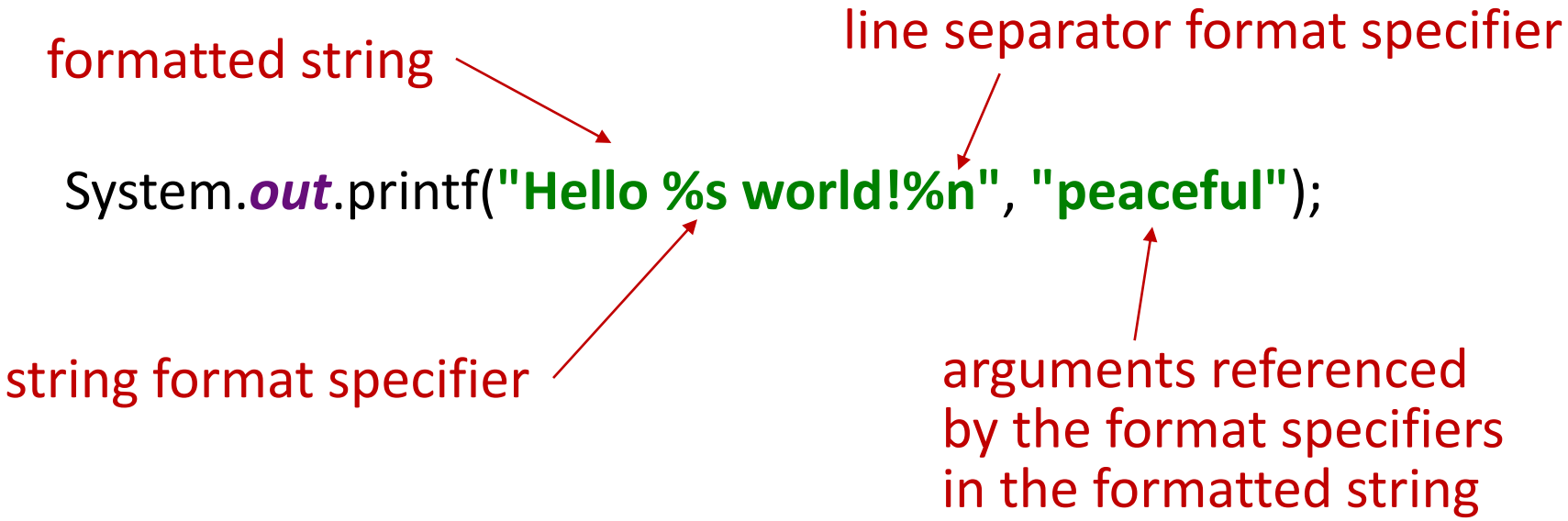
```
String result = "Sunday  ".replace(' ', 'Z').trim().concat("M n");

System.out.println(result);                    //SundayZZM n
```

The methods are evaluated from left to right.
The first method to execute in this example is replace, not concat.

See StringOperation

Print Formatted String to Console

- The **printf(String format, Object ... args)** method of the **java.io.PrintStream** class used to print formatted strings using various format specifiers.
- **printf()** uses the **java.util.Formatter** class to parse the format string and generate the output.



Output:
Hello peaceful world!

Print Formatted String to Console

Format specifiers:

%c - character
%d - decimal (int, long, ...) number (base 10)
%e - exponential floating-point number
%f - floating-point number (float, double)
%i - integer (base 10)
%o - octal number (base 8)
%s - String
%u - unsigned decimal (integer) number
%x - number in hexadecimal (base 16)
%t - formats date/time
%% - print a percent sign
%n - line separator

See PrintFormattedString

Print Formatted String to Console

Numbers formatting:

- You can set number width and precision for number formatting:
  %widthd - for decimal (integer) number, e.g. %15d
  %0widthd - for decimal (integer) number with leading zeros, e.g. %04d
  %width.precisionf - for floating-point number, e.g. %10.3f
  %width.precisione - for exponential floating-point number, e.g. %12.5e
- Numbers (and other type data) are left-justified by default
- To right justify use "minus" sign,
  e.g. %-12.5e or %-20s

See PrintFormattedString

Print Formatted String to Console

Date/time formatting:

- You can use special character for date time values formatting:

A/a - full/abbreviated day of week
B/b - full/abbreviated month
d - formats a two-digit day of the month
m - formats a two-digit month
Y/y - full year/last two digits of the year
j - day of the year

H, M, S - Hours, Minutes, Seconds
L, N – to represent the time in milliseconds and nanoseconds accordingly
p – AM/PM
z – prints out the difference from GMT

See PrintFormattedString

Module contents

- String processing
  - The String class
  - Operations with Strings
  - Immutable String in Java
  - The StringBuilder and StringBuffer
  - String pool in Java
  - The StringTokenizer class
  - The Regular Expressions in Java
  - The Scanner class

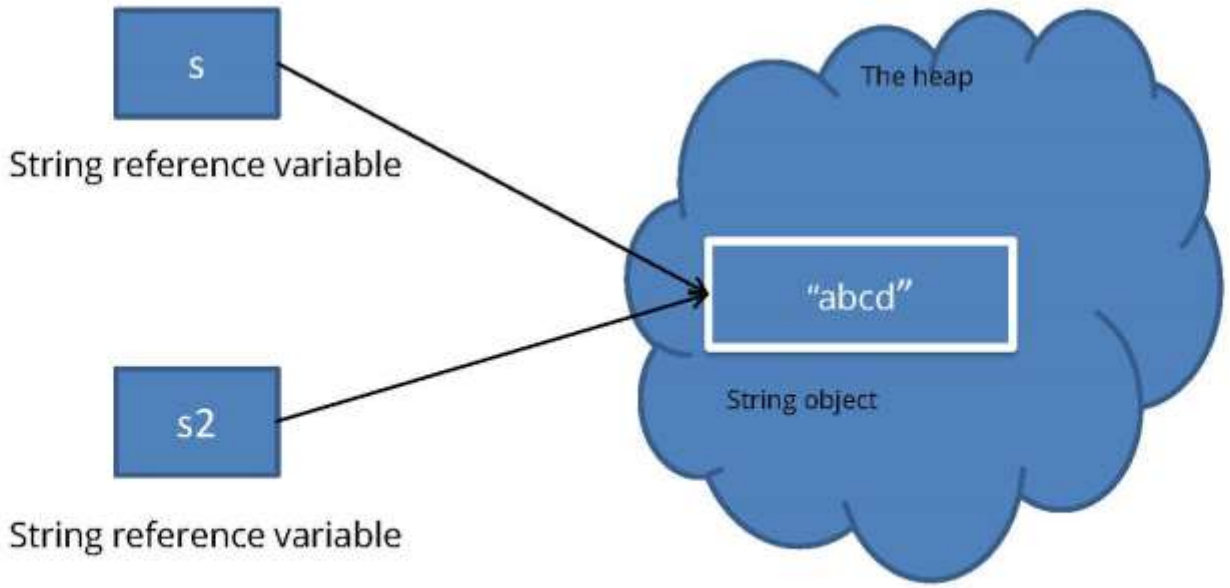
Immutable String in Java 1/3

- The String class is immutable, so that once it is created a String object cannot be changed.

```
public final class String
    implements java.io.Serializable, Comparable<String>, CharSequence {

    /**
     * The value is used for character storage.
     *
     * @implNote This field is trusted by the VM, and is a subject to
     * constant folding if String instance is constant. Overwriting this
     * field after construction will cause problems.
     *
     * Additionally, it is marked with {@link Stable} to trust the contents
     * of the array. No other facility in JDK provides this functionality (yet).
     * {@link Stable} is safe here, because value is never null.
     */
    @Stable
    private final byte[] value;
```

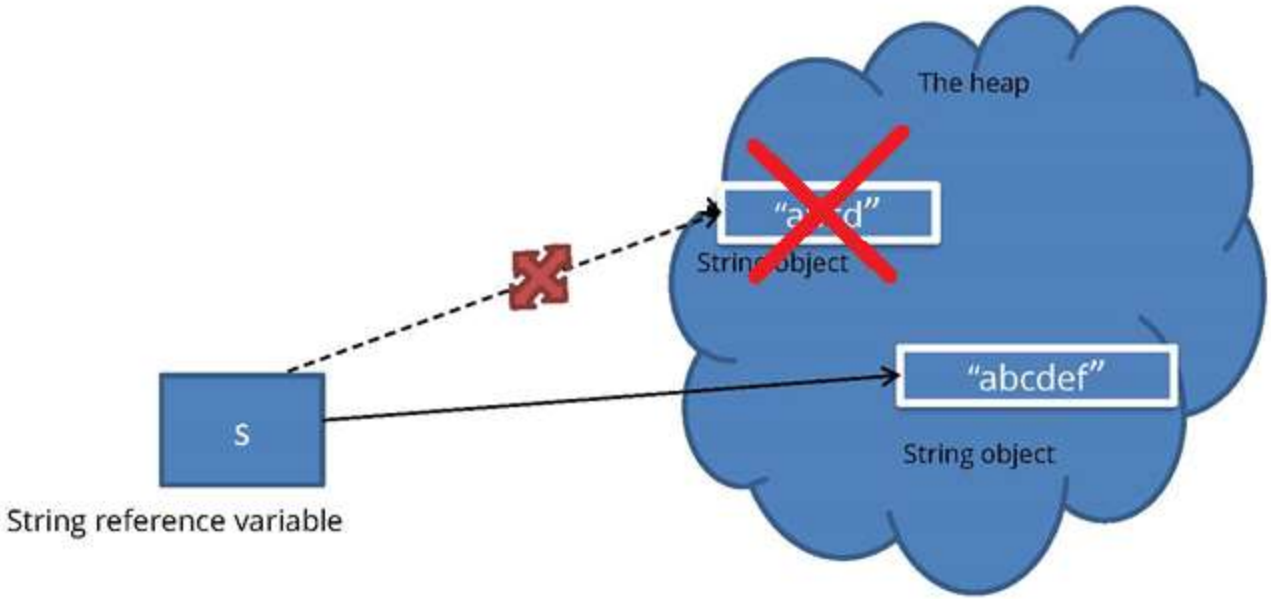
Immutable String in Java 2/3

1. String s = **"abcd"**;
2. String s2 = s;



Immutable String in Java 3/3

1. String s = **"abcd"**;
2. s = s+**"ef"**;



Module contents

- String processing
  - The String class
  - Operations with Strings
  - Immutable String in Java
  - The StringBuilder and StringBuffer
  - String pool in Java
  - The StringTokenizer class
  - The Regular Expressions in Java
  - The Scanner class

The StringBuilder and StringBuffer 1/6

- String str = "A0";
  for (int i = 1; i < 7; i++){
  str+="P"+i;
  }
  System.*out*.println(str);

- Each time creating new Object in heap
- Bad for performance

- **Objects in heap:**
- A0
- A0P1
- A0P1P2
- A0P1P2P3
- A0P1P2P3P4
- A0P1P2P3P4P5
- A0P1P2P3P4P5P6

See StringBuilderOperation

StringBuilder/StringBuffer

```
public final class StringBuilder
    extends AbstractStringBuilder
    implements java.io.Serializable, Comparable<StringBuilder>, CharSequence
{
```

```
abstract class AbstractStringBuilder implements Appendable, CharSequence {
    /**
     * The value is used for character storage.
     */
    byte[] value;
```

no **final** modificator  mutable String

StringBuffer - since JDK 1.0

StringBuilder - since JDK 5

StringBuilder/StringBuffer basic methods

- Method modified StringBuilder/StringBuffer (synchronized for StringBuffer)

StringBuilder **append**(String s)
StringBuilder **insert**(**int** offset, String s)
StringBuilder **delete**(**int** startIndex, **int** endIndex)
StringBuilder **deleteCharAt**(**int** index)
StringBuilder **replace**(**int** startIndex, int endIndex,
String str):
StringBuilder **reverse**()

See StringBuilderOperation

StringBuilder/StringBuffer basic methods

- Other methods

```
int capacity()
ensureCapacity(int minimumCapacity)
charAt(int index)
int indexOf(String str)
int indexOf(String str, int fromIndex)
int lastIndexOf(String str)
int lastIndexOf(String str, int fromIndex)
int length()
String substring(int beginIndex)
String substring(int beginIndex, int endIndex)
String toString()
int compareTo(StringBuilder another)
```

See StringBuilderOperation

StringBuilder/StringBuffer basic methods

StringBuilder append(boolean b)
StringBuilder append(char c)
StringBuilder append(char[] str)
StringBuilder append(char[] str, int offset, int len)
StringBuilder append(double d)
StringBuilder append(float f)
StringBuilder append(int i)
StringBuilder append(long lng)
StringBuilder append(CharSequence s)
StringBuilder append(CharSequence s, int start, int end)
StringBuilder append(Object obj)
StringBuilder append(String str)
StringBuilder append(StringBuffer sb)

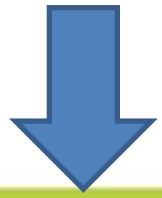
StringBuffer insert(**int** offset, boolean b):
similarly!

See StringBuilderOperation

The StringBuilder and StringBuffer 5/6

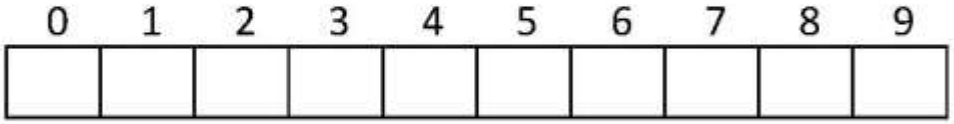
```
1. StringBuilder sb = new StringBuilder(10);
2. sb.append("Hello...");
3. char c = '!';
4. sb.append(c);// append a character
5. sb.insert(8, " Java");// Insert a string at index 5
6. sb.delete(5, 8);// Delete substring at index 5 to 8
7. System.out.println(sb);
```

See StringBuilderOperation

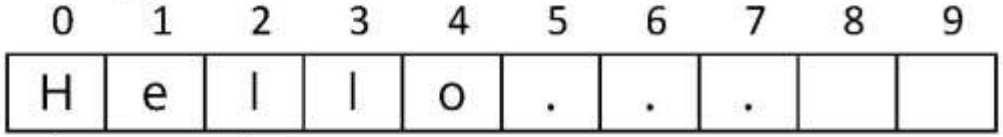


The StringBuilder and StringBuffer 6/6

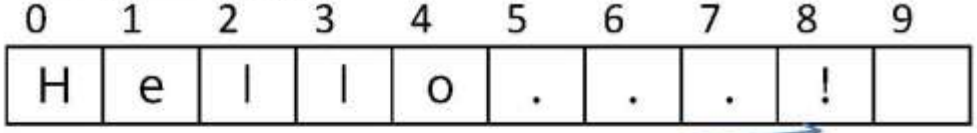
StringBuilder sb = new StringBuilder(10);



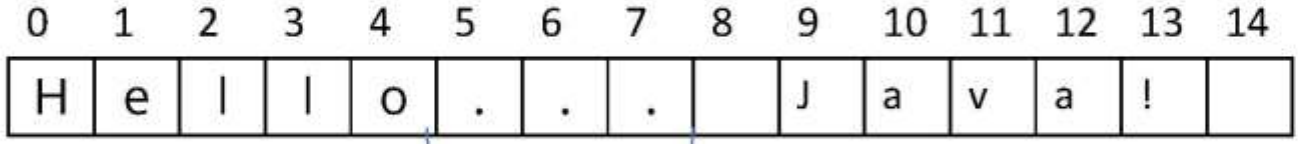
sb.append("Hello...");



sb.append('!');



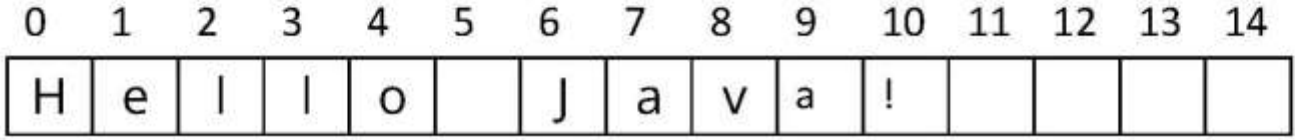
sb.insert(8," Java");



" Java"

whitespace

sb.delete(5,8);



The StringBuilder and StringBuffer 2/6

- **StringBuffer** - thread-safe, mutable sequence of characters

- **StringBuffer():** creates an empty string buffer with the initial capacity of 16.
- **StringBuffer(String str):** creates a string buffer with the specified string and capacity string.length() + 16.
- **StringBuffer(int capacity):** creates an empty string buffer with the specified capacity as length.

See StringBufferBuilderPerformance

The StringBuilder and StringBuffer 3/6

- **StringBuilder** – non-thread-safe, mutable sequence of characters
- The Java **StringBuilder** class is same as StringBuffer class except that it is non-synchronized.

StringBuilder(): creates an empty string Builder with the initial capacity of 16.
StringBuilder(String str): creates a string Builder with the specified string. and capacity string.length() + 16.
StringBuilder(int length): creates an empty string Builder with the specified capacity as length.

See StringBufferBuilderPerformance

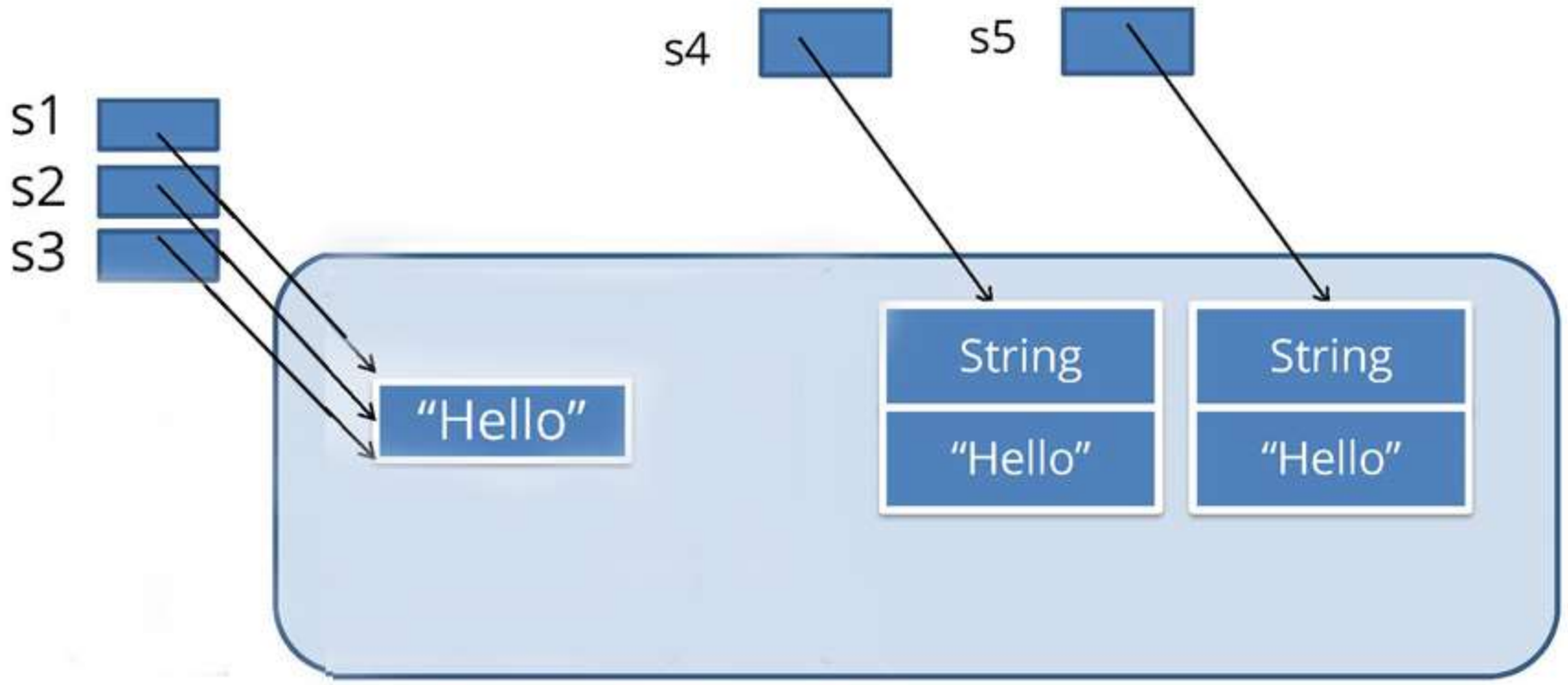
Module contents

- String processing
  - The String class
  - Operations with Strings
  - Immutable String in Java
  - The StringBuilder and StringBuffer
  - String pool in Java
  - The StringTokenizer class
  - The Regular Expressions in Java
  - The Scanner class

String pool in Java 1/4

- Java has provided a special mechanism for keeping the String literals - in a so-called string common pool.
- If two string literals have the same contents, they will share the same storage inside the common pool.
- The JVM can make this optimization only because String is immutable.

String pool in Java 3/4



"Common pool" for String literals

"Heap"

Method intern()

- Because the == operator checks for identity, all it has to do is compare two pointers, and obviously **this will be much faster than equals()**.
- So if you're going to compare the same strings multiple times, you can get a significant performance advantage by object identity checking instead of character comparison.
- When the **intern()** method is invoked, if the pool does not contain a string equal to this String, this String object is added to the pool and a reference to this String object is returned.
- Otherwise the string from the pool is returned.

See StringPool

String concatenation

- Strings are resolved to eliminate the concatenation operator if possible because of no extra object created.
- When Strings are created with the help of **String literals and '+' operator**, they get concatenated at compile time (*Compile-Time Resolution of Strings*).
- Compiler eliminates the concatenation operator, optimizes the string and send it to String pool.

```
String s1 = "Hello " + "World!";                    //CTR
String s2 = "Hello World!";
System.out.println(s1 == s2);                       //true
```

String concatenation

- When Strings are created with the help of **String literals along with variables and '+' operator**, they get concatenated at runtime only, as the value of the variables cannot be predicted beforehand (*Run-Time Resolution of Strings*).
- This causes extra objects to be generated without optimization with String pool.

```
String s3 = "Hello ";
/*s4 is not treated as literal*/
String s4 = s3 + "World!";          //RTR
System.out.println(s4 == s2);       //false
```

See StringConcatenation

Module contents

- String processing
  - The String class
  - Operations with Strings
  - Immutable String in Java
  - The StringBuilder and StringBuffer
  - String pool in Java
  - The StringTokenizer class
  - The Regular Expressions in Java
  - The Scanner class

The Regular Expressions in Java 1/7

- **Regular expression** (abbreviated regex or regexp) is a sequence of characters that define a search pattern, mainly for use in pattern matching with strings, or string matching, i.e. "find and replace"-like operations.
- The java.util.regex package primarily consists of three classes: **Pattern, Matcher, and PatternSyntaxException**.
- Regular Expression can be used to search, edit or manipulate text.

java.util.regex

since JDK 1.4

The Regular Expressions in Java 2/7

- The Java **Pattern** class (java.util.regex.Pattern), is the main access point of the Java regular expression API.
- A **Pattern** object is a compiled representation of a regular expression.
- Pattern pattern = Pattern.*compile*(**".*http://.*"**);
- Pattern pattern = Pattern.*compile*(**".*http://.*"**,

  for multi-line searched text → Pattern.*DOTALL*);

Char class: "." - any character, ".*" - any number of any chars

The Regular Expressions in Java 3/7

- The Java **Matcher** class (**java.util.regex.Matcher**) is used to search through a text for multiple occurrences of a regular expression.
- You can also use a **Matcher** to search for the same regular expression in different texts.

1. Matcher matcher = pattern.matcher("**searched in string http://mycompany.com**");
2. **boolean** matches = matcher.matches();

Pattern.*matches(...)*

public static boolean matches(String regex, CharSequence input)

See StringRegex

Matcher methods

```
boolean matches();
boolean find();
boolean find(int start);
int start();
int start(int group)
int end();
int end(int group)
String group();
String group(int group)
```

true - if **entire** input sequence matches with pattern

true - if **part or entire** input sequence matches with pattern

Returns the start/end index of the previous match.

For a matcher **m** with input sequence **s**, the expressions **m.group()** and **s.substring(m.start(), m. end())** are equivalent.

group

```
String patternString = "(This is the text)";
```

See StringRegex

The Regular Expressions in Java 4/7

- **Character Classes**
- . Dot, any character
- \d A digit: [0-9]
- \D A non-digit: [^0-9]
- \s A whitespace character: [\t\n\x0B\f\r]
- \S A non-whitespace character: [^\s]
- \w A word character: [a-zA-Z_0-9]
- \W A non-word character: [^\w]
- \b - word boundary, e.g. "\\b(\\w{2})\\b"

In Java, you will need to “double escape” these backslashes.

```
String pattern = "\\b\\d \\D \\W \\w \\S \\s\\b";
```

The Regular Expressions in Java 5/7

- **Quantifiers**
- * Match 0 or more times
- + Match 1 or more times
- ? Match 1 or 0 times
- {n} Match exactly n times
- {n,} Match at least n times
- {n,m} Match at least n but not more than m times

See StringRegex

The Regular Expressions in Java

- **Quantifiers - greedy vs reluctant**
- A greedy quantifier - matches as much as possible from the input string (until last match).
- A reluctant or "non-greedy" quantifier - matches as little as possible (first match only).
- Add ? at the end of quantifier to make it reluctant (non-greedy).

See StringRegex

The Regular Expressions in Java 6/7

- **Meta-characters**
- \ Escape the next meta-character (it becomes a normal/literal character)
- ^ Match the beginning of the line
- $ Match the end of the line (or before newline at the end)
- | Alternation (‘or’ statement)
- () Grouping
- [] Custom character class, [^] is inversion

The RegExp for Ukrainian alphabet is: "[А-Яа-яёЁЇїІіЄєҐґ']"

See StringRegex

The Regular Expressions in Java

- **Characters escaping**
- Java characters that have to be escaped by additional back slash \ in regular expressions are:

  \ . [ ] { } ( ) < > * + - = ! ? ^ $ |

- Two of the closing brackets (] and }) only have to be escaped after opening the same type of bracket.
- In [ ]-brackets some characters (like + and -) do sometimes work without escape.

See StringRegex

The Regular Expressions in Java - String methods

- public boolean matches(String regex)
- public String replaceFirst(String regex, String replacement)
- String replaceAll(String regex, String replacement)
- public String[] split(String regex, int limit)
- public String[] split(String regex)

- Since JDK 8 String class has method:

  public static String **join**(CharSequence delimiter,
  CharSequence... elements)

  to construct a sequence of characters separated by a delimiter.

See StringRegex

Module contents

- String processing
  - The String class
  - Operations with Strings
  - Immutable String in Java
  - The StringBuilder and StringBuffer
  - String pool in Java
  - The StringTokenizer class
  - The Regular Expressions in Java
  - The Scanner class

The Scanner class

- The java.util.Scanner class is a simple text scanner, which can parse strings and primitive types using regular expressions.
- Scanner can get text data from the String instance, console input, text file, et al.
- Scanner breaks its input into tokens using a delimiter pattern, which is whitespace by default.
- A scanning operation may block waiting for input.
- A Scanner is not safe while multithreaded use without external synchronization.

See ScannerCreation

since JDK5

The Scanner basic methods

public Scanner useDelimiter(String pattern)

public boolean hasNext()

public boolean hasNext(String pattern)

public String next()

public boolean hasNextLine()

public boolean hasNextLine(String pattern)

public String nextLine()

public boolean hasNextType()

public type nextType()

type = boolean, byte, short, int, long, float, double, BigInteger, BigDecimal

public String findInLine(String pattern)

public MatchResult match()

public void close()

See ScannerMethods