Chapter 11: Strings

Computer Fundamentals and Introduction to Programming

String · Index · Methods · Comparison · Formatting

Sercan KÜLCÜ · Giresun University

© Sercan Külcü. All rights reserved. · Interactive HTML version of these slides: https://sercankulcu.github.io/files/computer/slides/Chapter_11_Strings.html

Learning Outcomes

Representation

Explain the difference between String and char.

Processing

Use indices and the basic methods.

Comparison

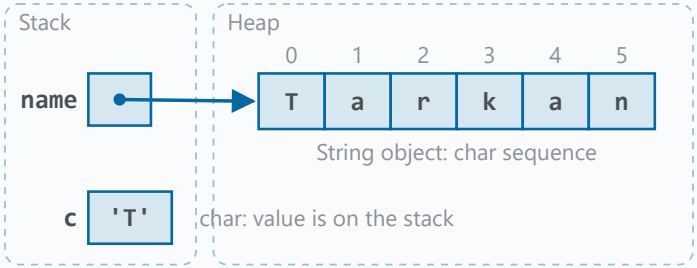
Tell content comparison from reference comparison.

Output

Produce formatted text with printf.

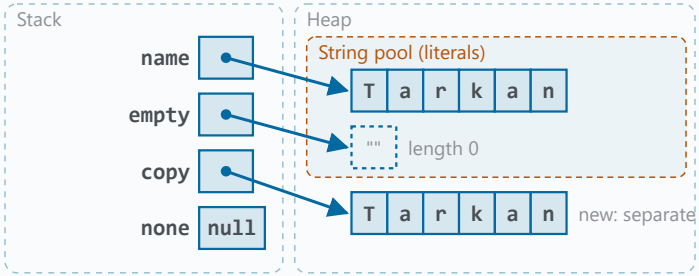
What Is a String?

- `String` is a `java.lang` class that represents a sequence of characters (no import needed).
- String literals use double quotes (`"Tarkan"`), a `char` uses single quotes (`'T'`).
- `String` is not a primitive type but an **immutable** object; the variable holds a reference to it.



Creating Strings

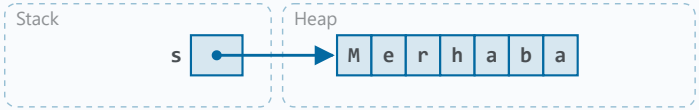
```
String name = "Tarkan";
String empty = "";
String copy = new String(name);
String none = null;
```



- A string literal is usually preferred over `new`.
- `""` is a real object of length 0; `null` refers to no object at all (`none.length()` → `NullPointerException`).

Immutability

- Once created, a String's contents **never change**.
- Concatenation and conversion produce a **new** String; the old object stays as it was.
- To use the result, assign the returned value to a variable.
- An object no variable refers to is reclaimed by the garbage collector.

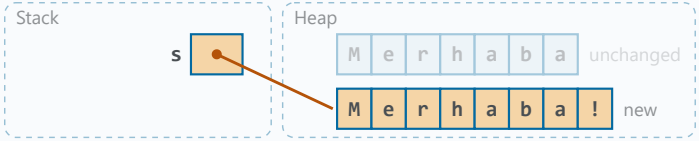



```
String s = "Merhaba";
```

Step 1 / 2

Immutability

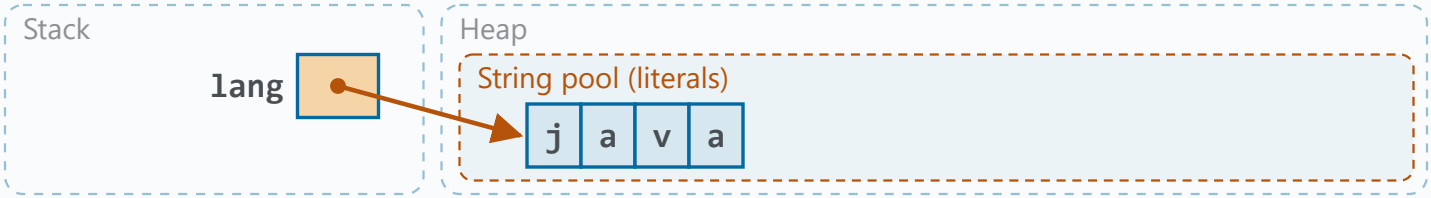
- Once created, a String's contents **never change**.
- Concatenation and conversion produce a **new** String; the old object stays as it was.
- To use the result, assign the returned value to a variable.
- An object no variable refers to is reclaimed by the garbage collector.



`s = s + "!";` → a new object; the arrow moves to it, the old object is unchanged.

Step 2 / 2

Immutability: `toUpperCase()`



`lang` refers to the `"java"` object in the pool.

Step 1 / 5

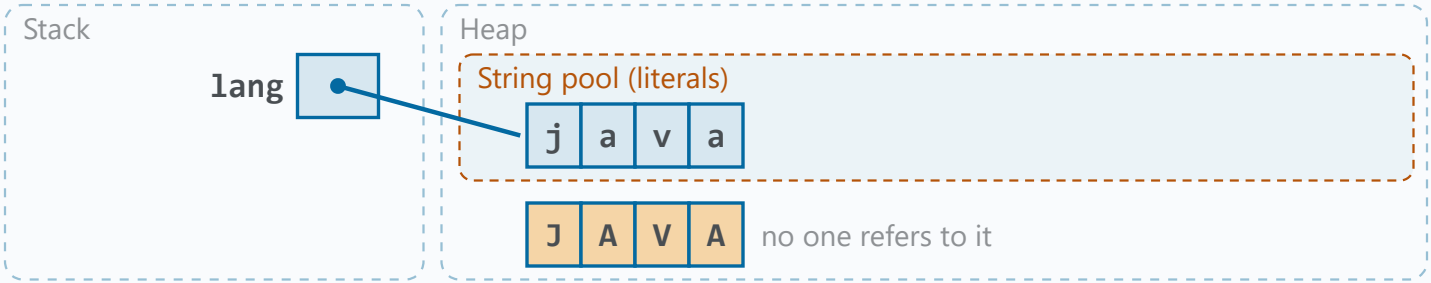
```
▶ String lang = "java";
2 lang.toUpperCase();
3 System.out.println(lang); // java
4 lang = lang.toUpperCase();
5 System.out.println(lang); // JAVA
```

VARIABLES

`lang = "java"`

OUTPUT

Immutability: `toUpperCase()`



`toUpperCase()` produces a **new** `"JAVA"` object, but the result is assigned to nothing → it is lost. `lang` still refers to the old object.

Step 2 / 5

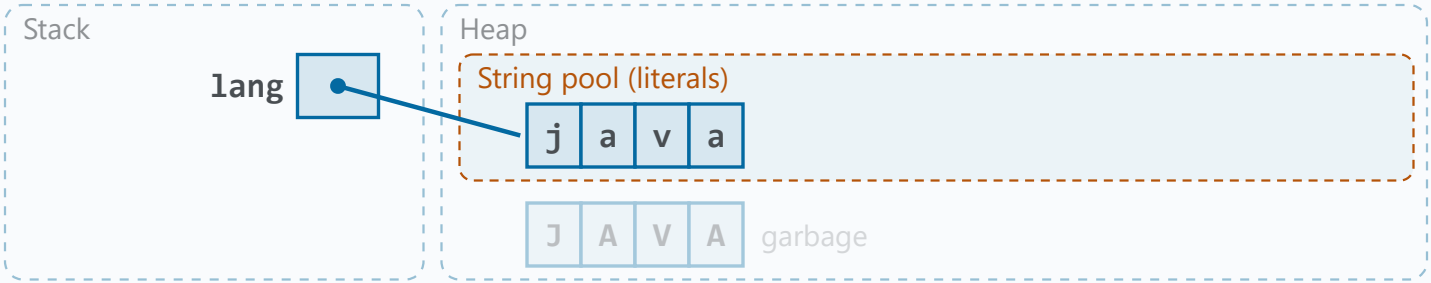
```
1 String lang = "java";
▶ lang.toUpperCase();
3 System.out.println(lang); // java
4 lang = lang.toUpperCase();
5 System.out.println(lang); // JAVA
```

VARIABLES

`lang = "java"`

OUTPUT

Immutability: `toUpperCase()`



The old object is unchanged, so it prints **java**.

Step 3 / 5

```
1 String lang = "java";
2 lang.toUpperCase();
▶ System.out.println(lang); // java
4 lang = lang.toUpperCase();
5 System.out.println(lang); // JAVA
```

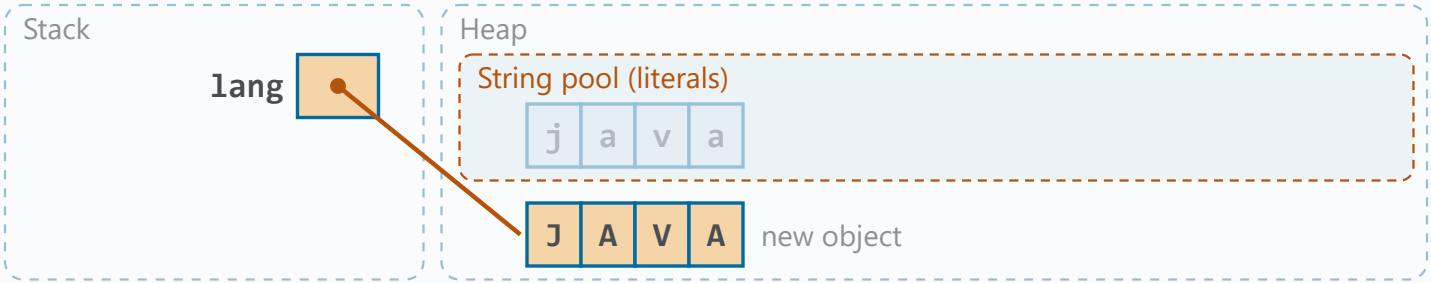
VARIABLES

`lang = "java"`

OUTPUT

```
java
```

Immutability: `toUpperCase()`



This time the returned object is **assigned** to `lang`: the arrow moves to it. The `"java"` object is still intact.

Step 4 / 5

```
1 String lang = "java";
2 lang.toUpperCase();
3 System.out.println(lang); // java
▶ lang = lang.toUpperCase();
5 System.out.println(lang); // JAVA
```

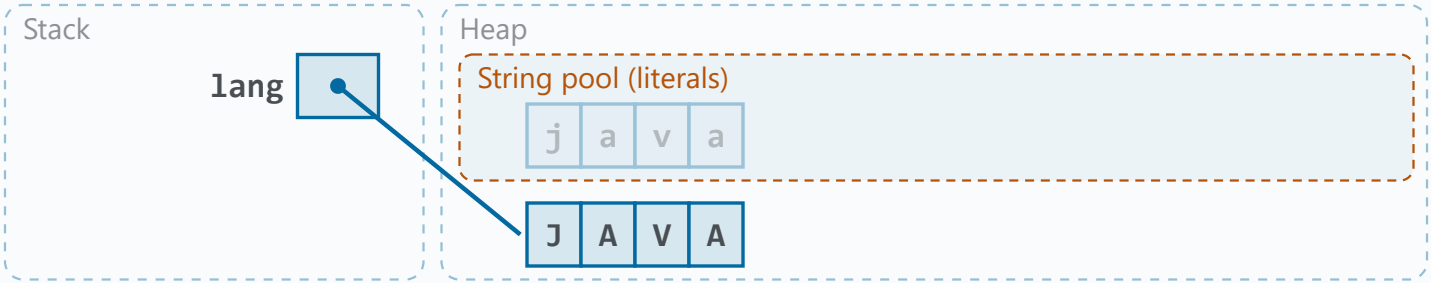
VARIABLES

`lang = "JAVA"`

OUTPUT

```
java
```

Immutability: `toUpperCase()`



It prints **JAVA**. What changed is not the object but where the variable points.

Step 5 / 5

```
1 String lang = "java";
2 lang.toUpperCase();
3 System.out.println(lang); // java
4 lang = lang.toUpperCase();
▶ System.out.println(lang); // JAVA
```

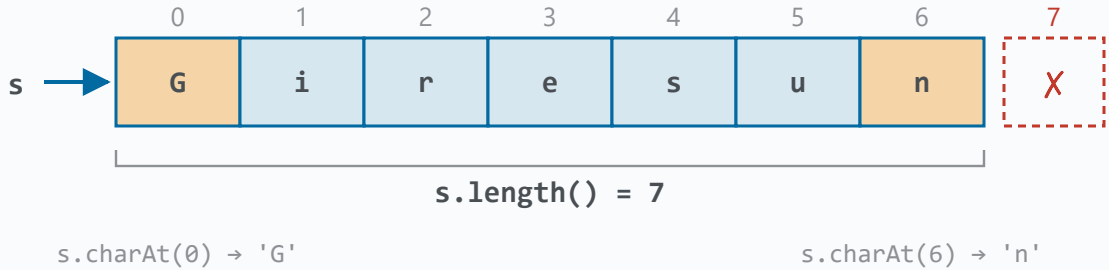
VARIABLES

`lang = "JAVA"`

OUTPUT

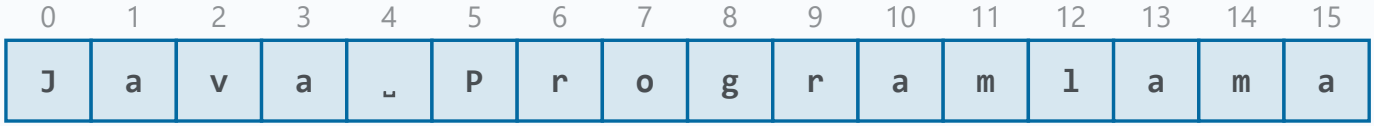
```
java
JAVA
```

Indexing



- The first character is at index **0**, the last at **length() − 1**; `length()` is a method (unlike the `length` field of arrays).
- `charAt(i)` returns the `char` at that position; `charAt(7)` → `StringIndexOutOfBoundsException`.

Basic Methods



A 16-character string (the space is a character too: index 4).

Step 1 / 6

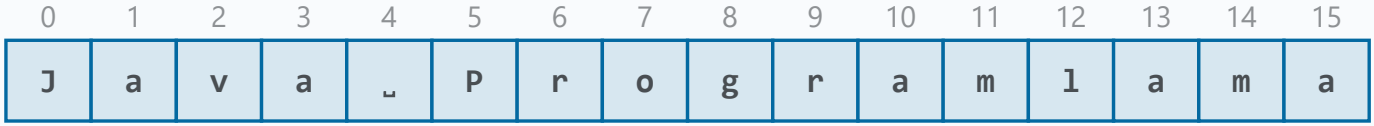
```
▶ String s = "Java Programlama";
2 int len = s.length();
3 int pos = s.indexOf("Programlama");
4 String sub = s.substring(5, 16);
5 boolean has = s.contains("Java");
6 System.out.println(sub + " " + pos);
```

VARIABLES

`s = "Java Programlama"`

OUTPUT

Basic Methods



`length()` → **16**; indices 0 … 15.

Step 2 / 6

```
1 String s = "Java Programlama";
▶ int len = s.length();
3 int pos = s.indexOf("Programlama");
4 String sub = s.substring(5, 16);
5 boolean has = s.contains("Java");
6 System.out.println(sub + " " + pos);
```

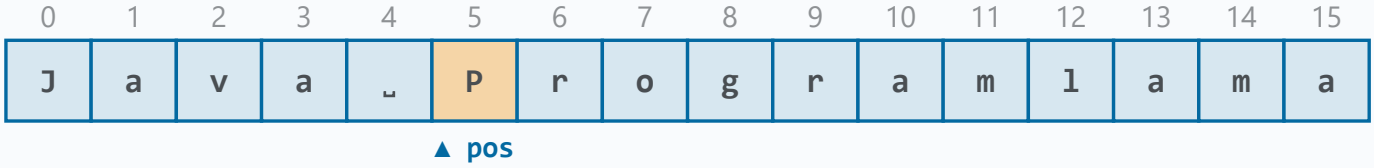
VARIABLES

`s = "Java Programlama"`

`len = 16`

OUTPUT

Basic Methods



`indexOf` gives the index of the **first character** of the match → **5** (−1 if absent).

Step 3 / 6

```
1 String s = "Java Programlama";
2 int len = s.length();
▶ int pos = s.indexOf("Programlama");
4 String sub = s.substring(5, 16);
5 boolean has = s.contains("Java");
6 System.out.println(sub + " " + pos);
```

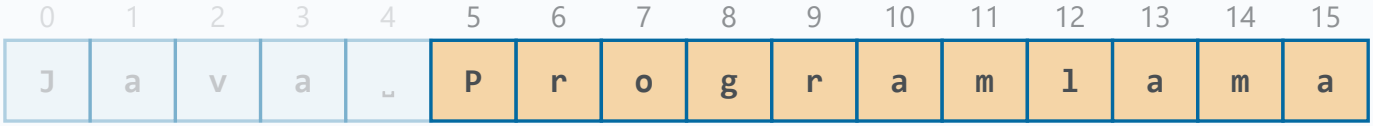
VARIABLES

`s = "Java Programlama"`

`len = 16` `pos = 5`

OUTPUT

Basic Methods



`substring(5, 16)` : 5 **included**, 16 **excluded** → 11 characters: "Programlama".

Step 4 / 6

```
1 String s = "Java Programlama";
2 int len = s.length();
3 int pos = s.indexOf("Programlama");
▶ String sub = s.substring(5, 16);
5 boolean has = s.contains("Java");
6 System.out.println(sub + " " + pos);
```

VARIABLES

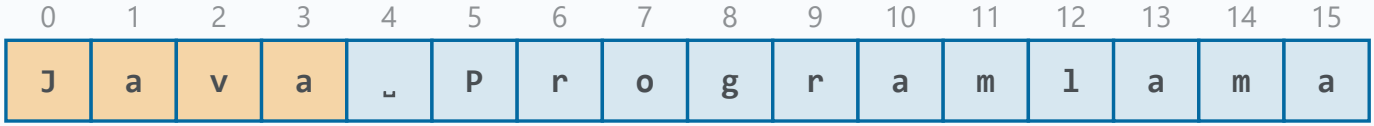
`s = "Java Programlama"`

`len = 16` `pos = 5`

`sub = "Programlama"`

OUTPUT

Basic Methods



`contains("Java")` → **true**. No method changed `s`.

Step 5 / 6

```
1 String s = "Java Programlama";
2 int len = s.length();
3 int pos = s.indexOf("Programlama");
4 String sub = s.substring(5, 16);
▶ boolean has = s.contains("Java");
6 System.out.println(sub + " " + pos);
```

VARIABLES

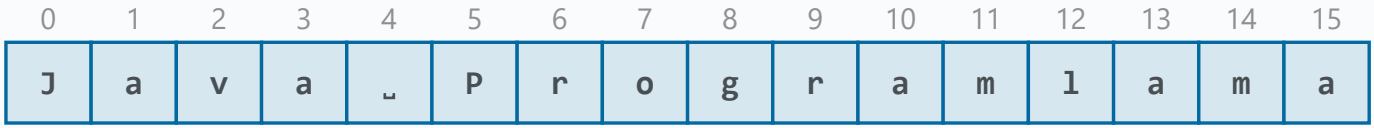
`s = "Java Programlama"`

`len = 16` `pos = 5`

`sub = "Programlama"` `has = true`

OUTPUT

Basic Methods



The results are joined and printed.

Step 6 / 6

```
1 String s = "Java Programlama";
2 int len = s.length();
3 int pos = s.indexOf("Programlama");
4 String sub = s.substring(5, 16);
5 boolean has = s.contains("Java");
▶ System.out.println(sub + " " + pos);
```

VARIABLES

`s = "Java Programlama"`

`len = 16` `pos = 5`

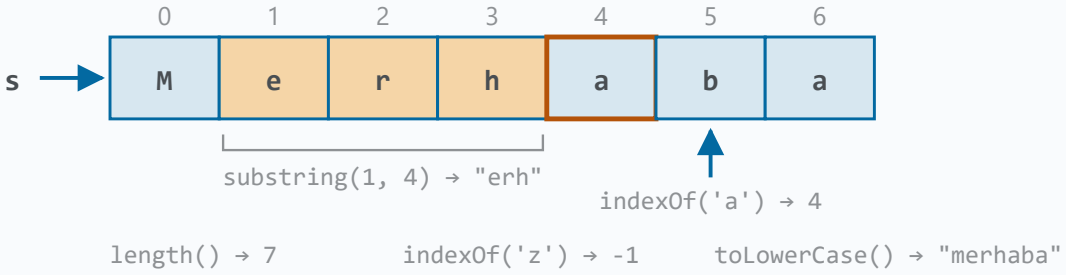
`sub = "Programlama"` `has = true`

OUTPUT

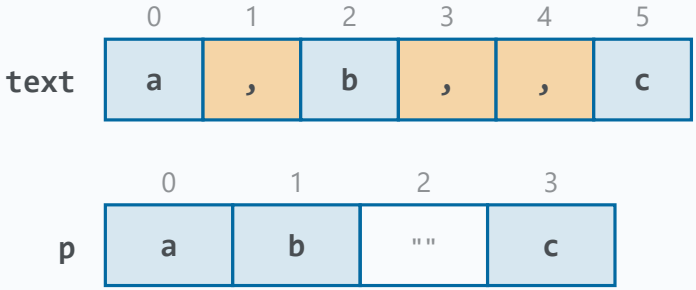
```
Programlama 5
```

Common String Methods

Method	Result
length()	Number of chars (code units)
charAt(i)	The char at position i
substring(a,b)	Substring from a (incl.) to b (excl.)
indexOf(x)	First position or -1
toLowerCase()	New lowercase String
trim() / strip()	New String without leading/trailing spaces



`split` and `replace` Step by Step



4 parts at the commas

`split(",")` cuts the text at the commas: **"a", "b", "", "c"**. Between two adjacent commas is an **empty string**, and it stays in the array.

Step 1 / 6

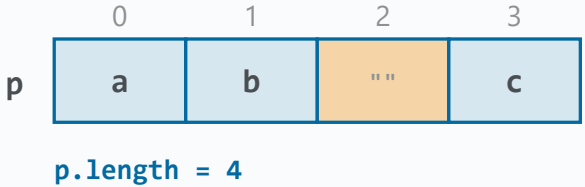
```
▶ String[] p = "a,b,,c".split(",");
2 System.out.println(Arrays.toString(p) + " " + p.length);
  // [a, b, , c] 4
3 String s = "ali veli";
4 String t = s.replace('l', 'L');
5 System.out.println(t); // aLi veLi
6 System.out.println(s); // ali veli
```

VARIABLES

p = [a, b, , c]

OUTPUT

`split` and `replace` Step by Step



`Arrays.toString(p)` → **[a, b, , c]**, `p.length` → **4**. (Empty parts at the *end* are dropped: `"a,b,,".split(",")` gives only 2 parts.)

Step 2 / 6

```
1 String[] p = "a,b,,c".split(",");
▶ System.out.println(Arrays.toString(p) + " " + p.length);
  // [a, b, , c] 4
3 String s = "ali veli";
4 String t = s.replace('l', 'L');
5 System.out.println(t); // aLi veLi
6 System.out.println(s); // ali veli
```

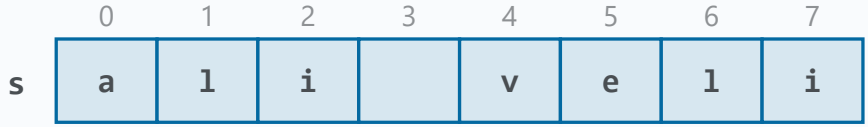
VARIABLES

p = **[a, b, , c]**

OUTPUT

```
[a, b, , c] 4
```

`split` and `replace` Step by Step



`s` = "ali veli": 8 characters, a space at index 3.

Step 3 / 6

```
1 String[] p = "a,b,,c".split(",");
2 System.out.println(Arrays.toString(p) + " " + p.length);
  // [a, b, , c] 4
▶ String s = "ali veli";
4 String t = s.replace('l', 'L');
5 System.out.println(t); // aLi veLi
6 System.out.println(s); // ali veli
```

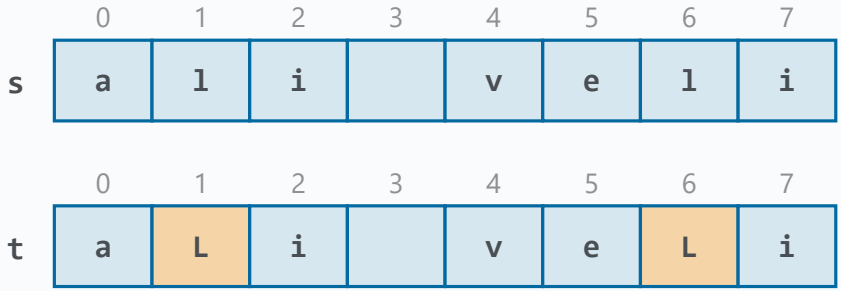
VARIABLES

`p = [a, b, , c]` `s = "ali veli"`

OUTPUT

```
[a, b, , c] 4
```

`split` and `replace` Step by Step



`replace('l','L')`: a new object

`replace('l', 'L')` turns every **'l'** into **'L'** (indices 1 and 6) and returns a **new** String; `s` is unchanged on this line.

Step 4 / 6

```
1 String[] p = "a,b,,c".split(",");
2 System.out.println(Arrays.toString(p) + " " + p.length);
  // [a, b, , c] 4
3 String s = "ali veli";
▶ String t = s.replace('l', 'L');
5 System.out.println(t); // aLi veLi
6 System.out.println(s); // ali veli
```

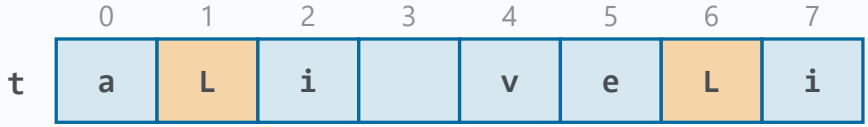
VARIABLES

`p = [a, b, , c]` `s = "ali veli"` `t = "aLi veLi"`

OUTPUT

```
[a, b, , c] 4
```

`split` and `replace` Step by Step



`t` is printed: **aLi veLi**.

Step 5 / 6

```
1 String[] p = "a,b,,c".split(",");
2 System.out.println(Arrays.toString(p) + " " + p.length);
  // [a, b, , c] 4
3 String s = "ali veli";
4 String t = s.replace('l', 'L');
▶ System.out.println(t); // aLi veLi
6 System.out.println(s); // ali veli
```

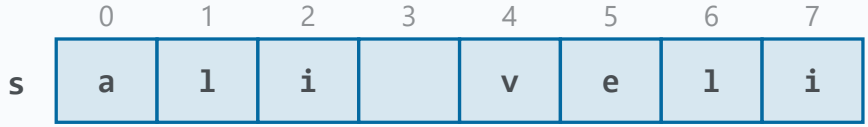
VARIABLES

`p = [a, b, , c]` `s = "ali veli"` `t = "aLi veLi"`

OUTPUT

```
[a, b, , c] 4
aLi veLi
```

`split` and `replace` Step by Step



`s` is still **ali veli**: Strings are immutable, so the result exists only in the returned value.

Step 6 / 6

```
1 String[] p = "a,b,,c".split(",");
2 System.out.println(Arrays.toString(p) + " " + p.length);
  // [a, b, , c] 4
3 String s = "ali veli";
4 String t = s.replace('l', 'L');
5 System.out.println(t); // aLi veLi
▶ System.out.println(s); // ali veli
```

VARIABLES

`p = [a, b, , c]` `s = "ali veli"` `t = "aLi veLi"`

OUTPUT

```
[a, b, , c] 4
aLi veLi
ali veli
```

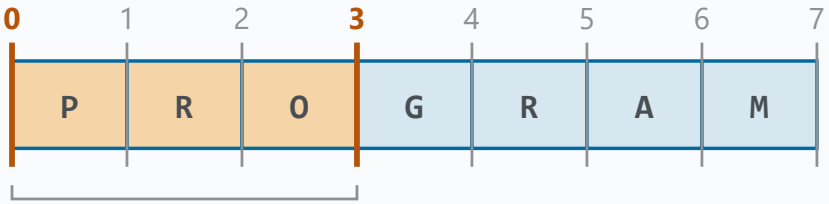
METHODS

replace, replaceAll, join, valueOf, format

Call	Result
`"banana".replace('a', 'o')`	`bonono`
`"banana".replace("an", "AN")`	`bANANa`
`"a1b22c333".replaceAll("[0-9]+", "#")`	`a#b#c#`
`"a1b22c333".replaceFirst("[0-9]+", "#")`	`a#b22c333`
`"Ada Can Ece".replaceAll(" +", " ")`	`Ada Can Ece`
`String.join(", ", "Ada", "Can", "Ece")`	`Ada, Can, Ece`
`String.valueOf(3.5)` / `String.valueOf(new char[]{'h','i'})`	`3.5` / `hi`
`String.format(Locale.US, "%.2f", 3.14159)`	`3.14`
`String.format(Locale.US, "%5d|%-5d|%05d", 42, 42, 42)`	`  42|42 |00042`
`String.format(Locale.US, "%,d", 1234567)`	`1,234,567`

`replace` searches for plain text, `replaceAll` for a regular expression; all return a **new String**. Without a locale, `%.2f` prints `3,14` on a computer set to Turkish.

Substring Bounds: `s = "PROGRAM"`



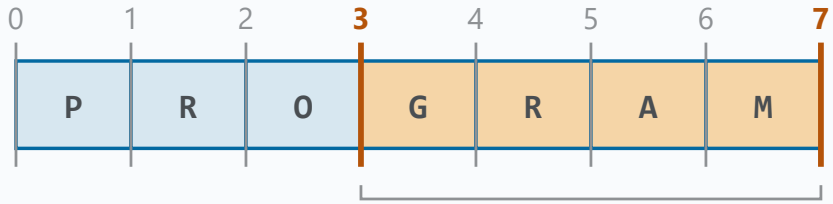
`s.substring(0, 3) → "PRO"`

Think of indices as **the lines between** characters: between 0 and 3 is "PRO". Length = 3 − 0.

Step 1 / 5

- `substring(a, b)` : **a included, b excluded**; the result has length **b − a**.
- Valid range: 0 ≤ a ≤ b ≤ length().

Substring Bounds: `s = "PROGRAM"`



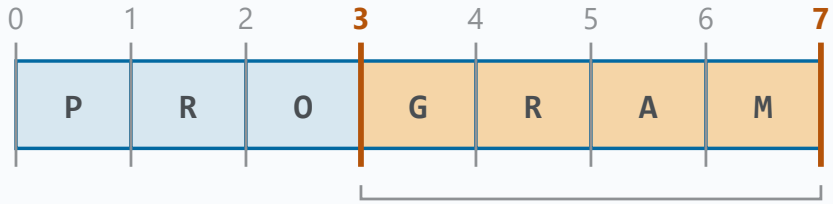
`s.substring(3, 7)` → `"GRAM"`

The end may be `length()` = 7: the last character is included.

Step 2 / 5

- `substring(a, b)` : **a included, b excluded**; the result has length **b − a**.
- Valid range: 0 ≤ a ≤ b ≤ length().

Substring Bounds: `s = "PROGRAM"`

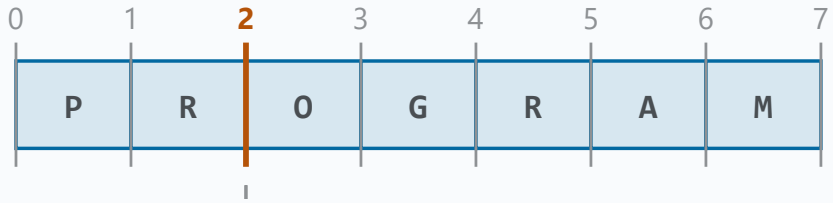


`s.substring(3)` → `"GRAM"`

One-parameter version: from the start **to the end**.

- `substring(a, b)` : **a included, b excluded**; the result has length **b − a**.
- Valid range: 0 ≤ a ≤ b ≤ length().

Substring Bounds: `s = "PROGRAM"`

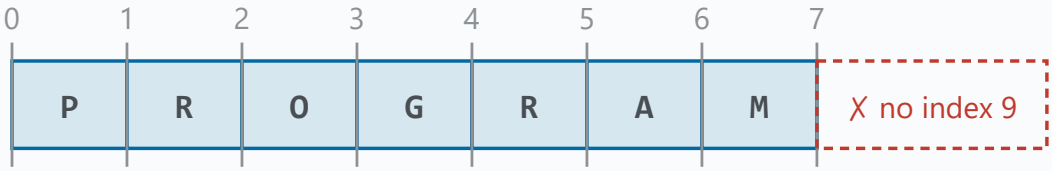


`s.substring(2, 2)` → `""`

Start = end → the empty string `""` (not an error).

- `substring(a, b)` : **a included, b excluded**; the result has length **b − a**.
- Valid range: 0 ≤ a ≤ b ≤ length().

Substring Bounds: `s = "PROGRAM"`



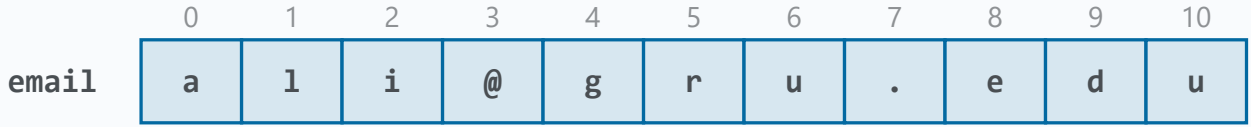
`s.substring(5, 9)` → `StringIndexOutOfBoundsException`

End > length() or start > end → run-time error.

Step 5 / 5

- `substring(a, b)` : **a included, b excluded**; the result has length **b − a**.
- Valid range: 0 ≤ a ≤ b ≤ length().

Application: Splitting an Email Address



An 11-character email address; we will separate the user name and the domain.

Step 1 / 7

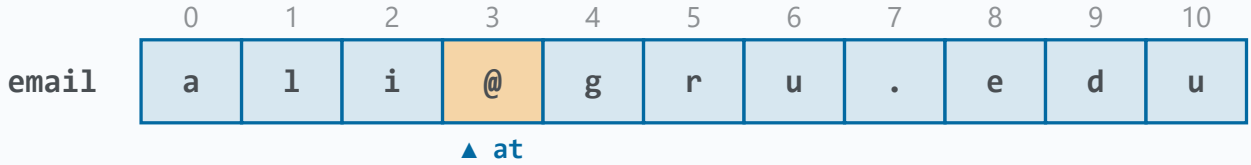
```
1 String email = "ali@gru.edu";
2 int at = email.indexOf('@');
3 String user = email.substring(0, at);
4 String domain = email.substring(at + 1);
5 char first = email.charAt(0);
6 System.out.println(user + " / " + domain);
7 System.out.println(Character.toUpperCase(first));
```

VARIABLES

email = "ali@gru.edu"

OUTPUT

Application: Splitting an Email Address



`indexOf('@')` → **3**: the separator's position.

Step 2 / 7

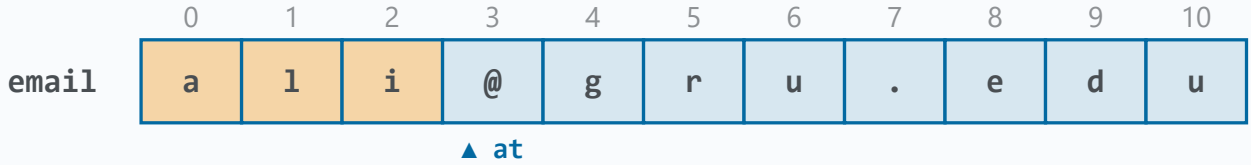
```
1 String email = "ali@gru.edu";
▶ int at = email.indexOf('@');
3 String user = email.substring(0, at);
4 String domain = email.substring(at + 1);
5 char first = email.charAt(0);
6 System.out.println(user + " / " + domain);
7 System.out.println(Character.toUpperCase(first));
```

VARIABLES

`email = "ali@gru.edu"` `at = 3`

OUTPUT

Application: Splitting an Email Address



`substring(0, 3)` : 0 included, 3 (@) excluded → "ali".

Step 3 / 7

```
1 String email = "ali@gru.edu";
2 int at = email.indexOf('@');
▶ String user = email.substring(0, at);
4 String domain = email.substring(at + 1);
5 char first = email.charAt(0);
6 System.out.println(user + " / " + domain);
7 System.out.println(Character.toUpperCase(first));
```

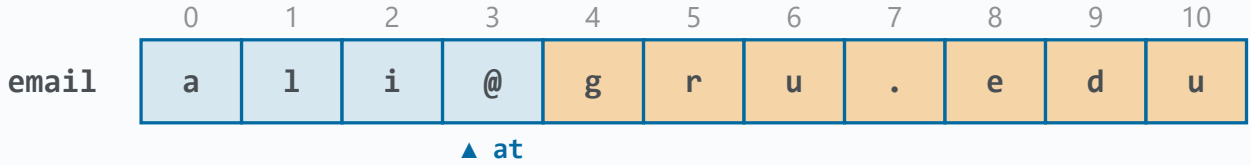
VARIABLES

email = "ali@gru.edu" at = 3

user = "ali"

OUTPUT

Application: Splitting an Email Address



`substring(4)`: from the character after @ to the end → "gru.edu".

Step 4 / 7

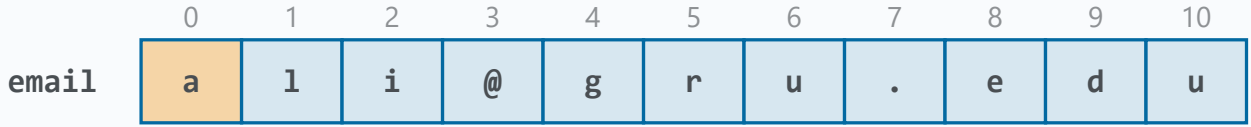
```
1 String email = "ali@gru.edu";
2 int at = email.indexOf('@');
3 String user = email.substring(0, at);
▶ String domain = email.substring(at + 1);
5 char first = email.charAt(0);
6 System.out.println(user + " / " + domain);
7 System.out.println(Character.toUpperCase(first));
```

VARIABLES

`email = "ali@gru.edu"` `at = 3` `user = "ali"` `domain = "gru.edu"`

OUTPUT

Application: Splitting an Email Address



`charAt(0)` returns a `char`: **'a'** (single quotes).

Step 5 / 7

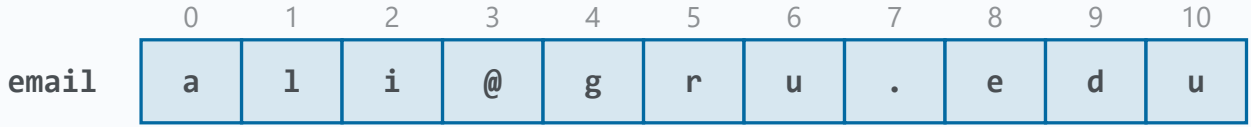
```
1 String email = "ali@gru.edu";
2 int at = email.indexOf('@');
3 String user = email.substring(0, at);
4 String domain = email.substring(at + 1);
▶ char first = email.charAt(0);
6 System.out.println(user + " / " + domain);
7 System.out.println(Character.toUpperCase(first));
```

VARIABLES

`email = "ali@gru.edu"` `at = 3` `user = "ali"` `domain = "gru.edu"` `first = 'a'`

OUTPUT

Application: Splitting an Email Address



The parts are joined and printed.

Step 6 / 7

```
1 String email = "ali@gru.edu";
2 int at = email.indexOf('@');
3 String user = email.substring(0, at);
4 String domain = email.substring(at + 1);
5 char first = email.charAt(0);
▶ System.out.println(user + " / " + domain);
7 System.out.println(Character.toUpperCase(first));
```

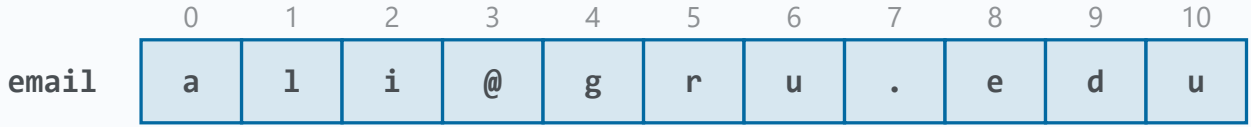
VARIABLES

email = "ali@gru.edu" | at = 3 | user = "ali" | domain = "gru.edu" | first = 'a'

OUTPUT

```
ali / gru.edu
```

Application: Splitting an Email Address



`Character.toUpperCase` converts a char to uppercase → **A**. `email` never changed.

Step 7 / 7

```
1 String email = "ali@gru.edu";
2 int at = email.indexOf('@');
3 String user = email.substring(0, at);
4 String domain = email.substring(at + 1);
5 char first = email.charAt(0);
6 System.out.println(user + " / " + domain);
▶ System.out.println(Character.toUpperCase(first));
```

VARIABLES

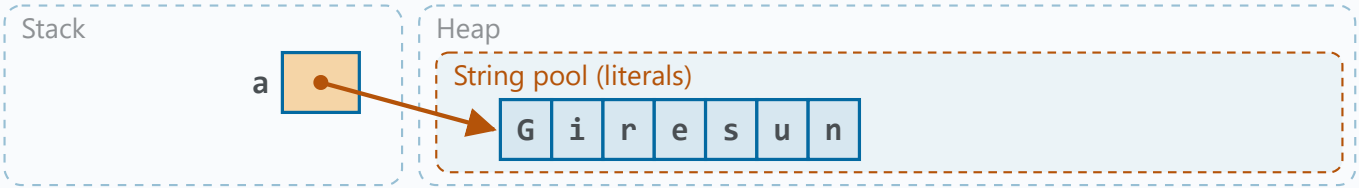
`email = "ali@gru.edu"` `at = 3` `user = "ali"` `domain = "gru.edu"` `first = 'a'`

OUTPUT

```
ali / gru.edu
A
```

COMPARISON · STEP THROUGH THE CODE

== vs. equals



The literal `"Giresun"` goes into the string pool; `a` refers to it.

Step 1 / 6

```
▶ String a = "Giresun";
2 String b = new String("Giresun");
3 String c = "Giresun";
4 System.out.println(a == b);      // false
5 System.out.println(a == c);      // true
6 System.out.println(a.equals(b)); // true
```

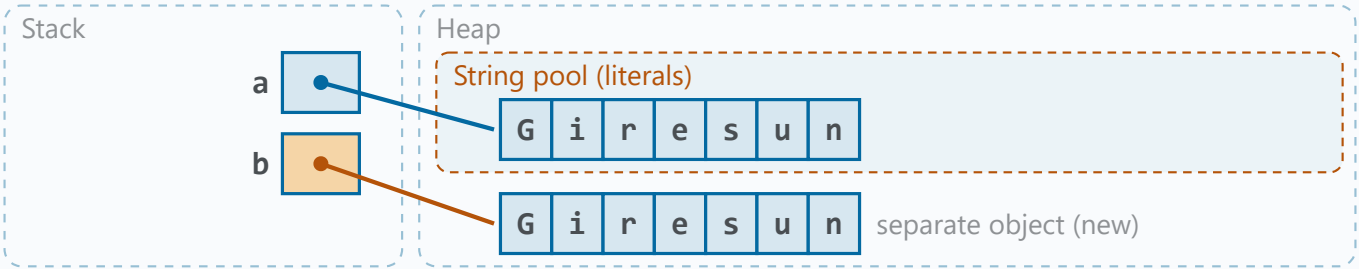
VARIABLES

a = → pool

OUTPUT

COMPARISON · STEP THROUGH THE CODE

== vs. equals



`new String(...)` creates a **separate** object on the heap, even with the same contents.

Step 2 / 6

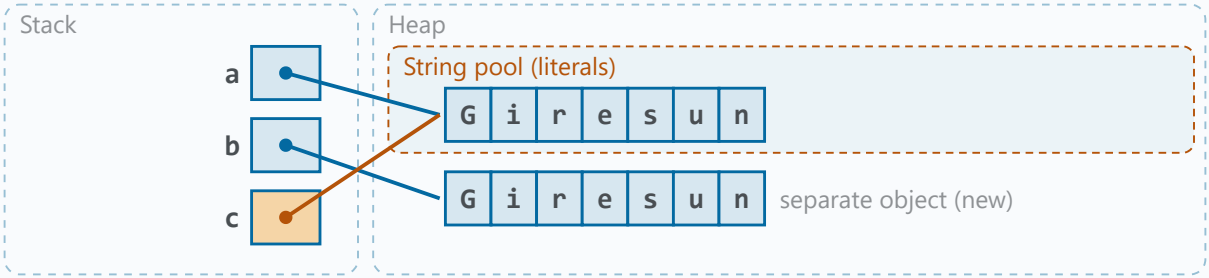
```
1 String a = "Giresun";
▶ String b = new String("Giresun");
3 String c = "Giresun";
4 System.out.println(a == b);      // false
5 System.out.println(a == c);      // true
6 System.out.println(a.equals(b)); // true
```

VARIABLES

`a = → pool` `b = → new object`

OUTPUT

== vs. equals



The same literal again: the **existing** object in the pool is shared → c carries the same arrow as a.

Step 3 / 6

```
1 String a = "Giresun";
2 String b = new String("Giresun");
▶ String c = "Giresun";
4 System.out.println(a == b);      // false
5 System.out.println(a == c);      // true
6 System.out.println(a.equals(b)); // true
```

VARIABLES

a = → pool

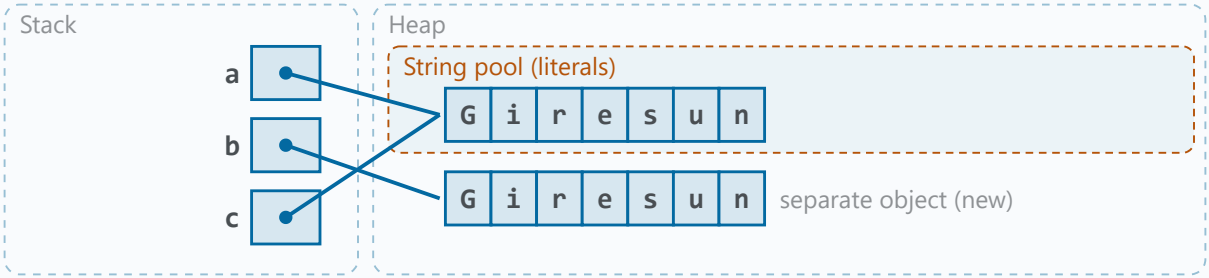
b = → new object

c = → pool

OUTPUT

COMPARISON · STEP THROUGH THE CODE

== vs. equals



`a == b` : the two arrows point to **different** objects → **false** (contents are ignored).

Step 4 / 6

```
1 String a = "Giresun";
2 String b = new String("Giresun");
3 String c = "Giresun";
▶ System.out.println(a == b);       // false
5 System.out.println(a == c);       // true
6 System.out.println(a.equals(b)); // true
```

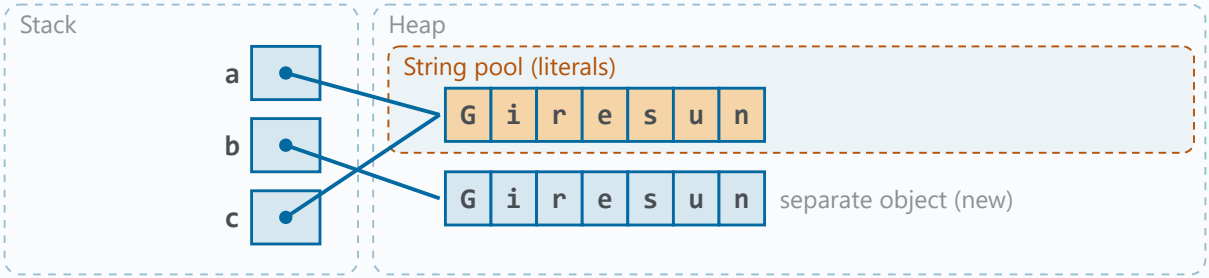
VARIABLES

`a = → pool` `b = → new object` `c = → pool`

OUTPUT

```
false
```

== vs. equals



`a == c` : the two arrows point to **the same** object → **true**.

Step 5 / 6

```
1 String a = "Giresun";
2 String b = new String("Giresun");
3 String c = "Giresun";
4 System.out.println(a == b);       // false
▶ System.out.println(a == c);       // true
6 System.out.println(a.equals(b)); // true
```

VARIABLES

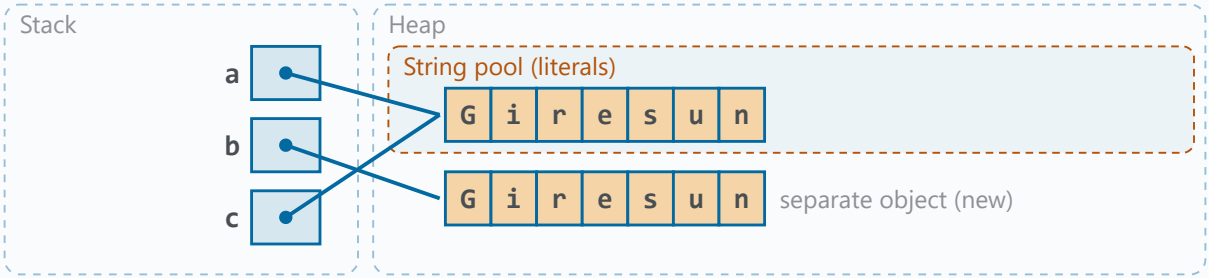
`a = → pool` `b = → new object` `c = → pool`

OUTPUT

```
false
true
```

COMPARISON · STEP THROUGH THE CODE

== vs. equals



`equals` compares character by character: G=G, i=i, … → **true**. Always use `equals` for contents.

Step 6 / 6

```
1 String a = "Giresun";
2 String b = new String("Giresun");
3 String c = "Giresun";
4 System.out.println(a == b);      // false
5 System.out.println(a == c);      // true
▶ System.out.println(a.equals(b)); // true
```

VARIABLES

`a = → pool` `b = → new object` `c = → pool`

OUTPUT

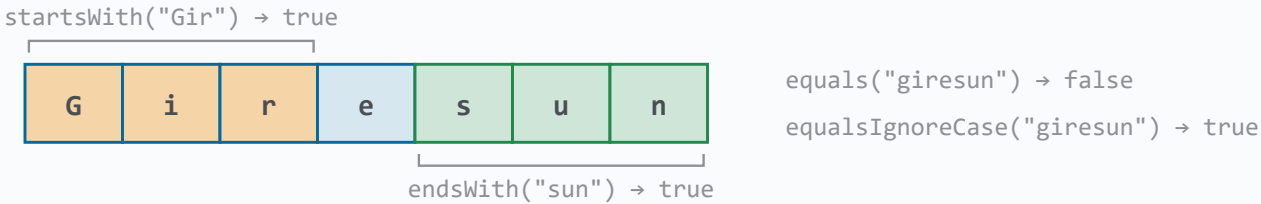
```
false
true
true
```

COMPARISON

Comparison Methods

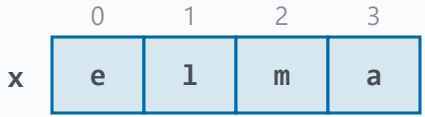
Method	Purpose	Example
`equals`	Content equality	`"abc".equals("abc") → true`
`equalsIgnoreCase`	Equality ignoring case	`"Java".equalsIgnoreCase("JAVA") → true`
`startsWith`	Starts with given text	`"Giresun".startsWith("Gir") → true`
`endsWith`	Ends with given text	`"Giresun".endsWith("sun") → true`
`compareTo`	Lexicographic order: <0, 0, >0	`"elma".compareTo("armut") → positive`

compareTo works on Unicode values: uppercase letters come before lowercase, and Turkish letters (ç, ğ, ı, ö, ş, ü) may not follow alphabetical order.



COMPARISON · STEP THROUGH THE CODE

`compareTo`: Character and Length Differences



First `x` = "elma" (4 characters). The comparison uses the Unicode codes of these characters.

Step 1 / 6

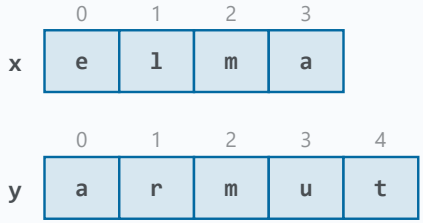
```
▶ String x = "elma";
2 String y = "armut";
3 System.out.println(x.compareTo(y)); // 4
4 String k = "kedi";
5 System.out.println(k.compareTo("kediler")); // -3
```

VARIABLES

`x = "elma"`

OUTPUT

`compareTo`: Character and Length Differences



`y` = "armut" (5 characters). The next line computes `x.compareTo(y)`.

Step 2 / 6

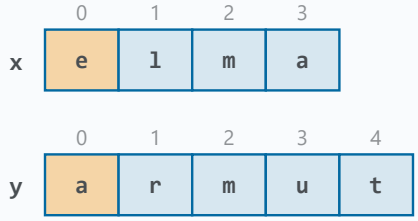
```
1 String x = "elma";
▶ String y = "armut";
3 System.out.println(x.compareTo(y)); // 4
4 String k = "kedi";
5 System.out.println(k.compareTo("kediler")); // -3
```

VARIABLES

`x = "elma"` `y = "armut"`

OUTPUT

`compareTo`: Character and Length Differences



The **first difference** is at position 0: **'e' - 'a' = 101 - 97 = 4**. Positive: "elma" comes after "armut" in the dictionary.

Step 3 / 6

```
1 String x = "elma";
2 String y = "armut";
▶ System.out.println(x.compareTo(y)); // 4
4 String k = "kedi";
5 System.out.println(k.compareTo("kediler")); // -3
```

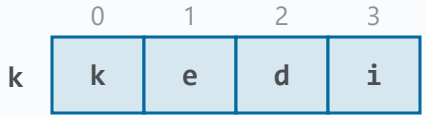
VARIABLES

`x = "elma"` `y = "armut"`

OUTPUT

```
4
```

`compareTo`: Character and Length Differences



`k` = "kedi" (4 characters). Now let's see what happens when one string is a prefix of the other.

Step 4 / 6

```
1 String x = "elma";
2 String y = "armut";
3 System.out.println(x.compareTo(y)); // 4
▶ String k = "kedi";
5 System.out.println(k.compareTo("kediler")); // -3
```

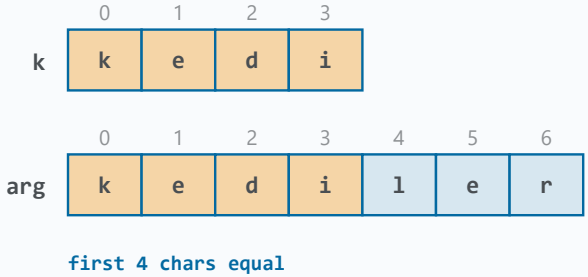
VARIABLES

`x = "elma"` `y = "armut"` `k = "kedi"`

OUTPUT

```
4
```

`compareTo`: Character and Length Differences



`k.compareTo("kediler")` : the first 4 characters are **the same** and "kedi" ends; the result is the **length difference**.

Step 5 / 6

```
1 String x = "elma";
2 String y = "armut";
3 System.out.println(x.compareTo(y)); // 4
4 String k = "kedi";
▶ System.out.println(k.compareTo("kediler")); // -3
```

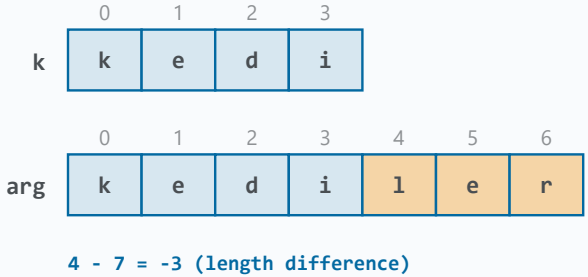
VARIABLES

`x = "elma"` `y = "armut"` `k = "kedi"`

OUTPUT

```
4
```

`compareTo`: Character and Length Differences



4 - 7 = -3: the shorter "kedi" comes first. Rule: the character difference if one exists, otherwise the length difference.

Step 6 / 6

```
1 String x = "elma";
2 String y = "armut";
3 System.out.println(x.compareTo(y)); // 4
4 String k = "kedi";
▶ System.out.println(k.compareTo("kediler")); // -3
```

VARIABLES

`x = "elma"` `y = "armut"` `k = "kedi"`

OUTPUT

```
4
-3
```

COMPARISON

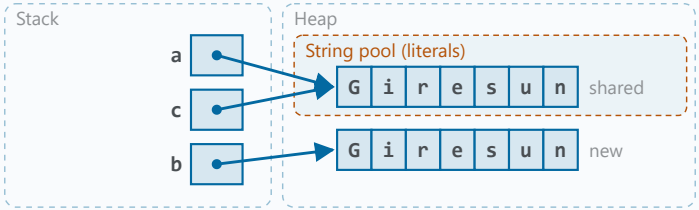
Turkish Characters and Locale

Call (tr = Locale.forLanguageTag("tr"))	Result
`"TITLE".toLowerCase(Locale.ENGLISH)`	`title`
`"TITLE".toLowerCase(Locale.forLanguageTag("tr"))`	`tıtle`
`"i".toUpperCase(tr)`	`İ`
`"i".toUpperCase(Locale.ENGLISH)`	`I`
`"I".toLowerCase(tr)`	`ı`
`"ISPARTA".toLowerCase(tr)`	`ısparta`
`"ısparta".toUpperCase(tr)`	`ISPARTA`

- A plain `toLowerCase()` depends on the machine's **default locale**: the same code may give a different result on another computer.
- Pass `Locale.ENGLISH` for technical text such as keywords or file extensions, and `Locale.forLanguageTag("tr")` for Turkish text.
- `"TITLE".toLowerCase(tr).equals("title")` → `false`; `"title".equalsIgnoreCase("TITLE")` → `true`.

The String Pool

- Identical string literals share **a single** object in the pool (saves memory; safe thanks to immutability).
- `new String(...)` always creates a separate object.
- You can't always tell from the code whether a String came from the pool, so compare contents with **`equals`**, not `==`.



MEMORY

Constant Folding and intern

```
String a = "a", b = "b";
String r = a + b;
final String f = "a";
System.out.println(("a" + "b") == "ab");  // true
System.out.println((f + "b") == "ab");    // true
System.out.println(r == "ab");            // false
System.out.println(r.equals("ab"));       // true
System.out.println(r.intern() == "ab");   // true
```

Why?

The compiler joins constant expressions (`"a" + "b"` , a `final` variable); the result is the pooled `"ab"` . `a + b` creates a **new object** at run time; `intern()` returns the pooled object.

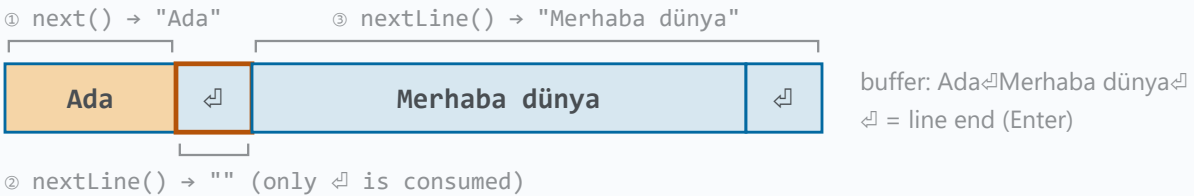
Null-safe comparison

With `String x = null;` , `"text".equals(x)` → `false` , but `x.equals("text")` → **NullPointerException**. So put the literal on the left.

Reading Text with Scanner

```
import java.util.Scanner;
// ...
Scanner in = new Scanner(System.in);
String word = in.next();
in.nextLine();
String line = in.nextLine();
```

- `next()` reads one word: it skips leading spaces and stops at the first space or line end.
- `nextLine()` reads everything up to the line end (⏎) and consumes the ⏎.
- After `next()` or `nextInt()`, the ⏎ left in the buffer must be consumed by a separate `nextLine()`.



INPUT · STEP THROUGH THE CODE

Code Trace: `next()` and `nextLine()`

Keyboard buffer (System.in)



▲ read position

The Scanner attaches to the keyboard stream; the buffer is empty for now.

Step 1 / 7

```
▶ Scanner in = new Scanner(System.in);
2 String word = in.next();
3 in.nextLine();
4 String line = in.nextLine();
5 System.out.println(word + " | " + line);
```

VARIABLES

—

OUTPUT

Without line 3, the `nextLine()` on line 4 would read the leftover ⏎ and return an **empty** line at once.

INPUT · STEP THROUGH THE CODE

Code Trace: `next()` and `nextLine()`

Keyboard buffer (System.in)



▲ read position

`next()` finds nothing to read and **waits**. The user types **Ada** and presses Enter → "Ada⏎" enters the buffer.

Step 2 / 7

```
1 Scanner in = new Scanner(System.in);
▶ String word = in.next();
3 in.nextLine();
4 String line = in.nextLine();
5 System.out.println(word + " | " + line);
```

VARIABLES

—

OUTPUT

Without line 3, the `nextLine()` on line 4 would read the leftover ⏎ and return an **empty** line at once.

INPUT · STEP THROUGH THE CODE

Code Trace: `next()` and `nextLine()`

Keyboard buffer (System.in)



▲ read position

`next()` reads up to a space or line end → "Ada". **The ⏎ stays in the buffer.**

Step 3 / 7

```
1 Scanner in = new Scanner(System.in);
▶ String word = in.next();
3 in.nextLine();
4 String line = in.nextLine();
5 System.out.println(word + " | " + line);
```

VARIABLES

`word = "Ada"`

OUTPUT

Without line 3, the `nextLine()` on line 4 would read the leftover ⏎ and return an **empty** line at once.

INPUT · STEP THROUGH THE CODE

Code Trace: `next()` and `nextLine()`

Keyboard buffer (System.in)



`nextLine()` reads up to the ⏎ → returns "" (empty) and consumes the ⏎. The buffer is clean now.

Step 4 / 7

```
1 Scanner in = new Scanner(System.in);
2 String word = in.next();
▶ in.nextLine();
4 String line = in.nextLine();
5 System.out.println(word + " | " + line);
```

VARIABLES

`word = "Ada"`

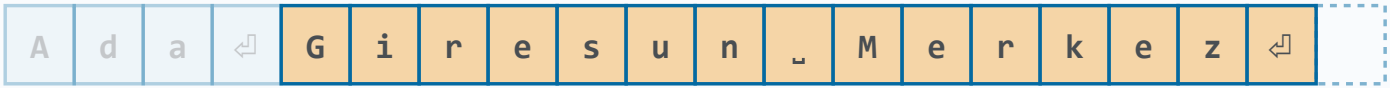
OUTPUT

Without line 3, the `nextLine()` on line 4 would read the leftover ⏎ and return an **empty** line at once.

INPUT · STEP THROUGH THE CODE

Code Trace: `next()` and `nextLine()`

Keyboard buffer (System.in)



The buffer is empty → `nextLine()` waits. The user types **Giresun Merkez** and presses Enter.

Step 5 / 7

```
1 Scanner in = new Scanner(System.in);
2 String word = in.next();
3 in.nextLine();
▶ String line = in.nextLine();
5 System.out.println(word + " | " + line);
```

VARIABLES

`word = "Ada"`

OUTPUT

Without line 3, the `nextLine()` on line 4 would read the leftover ⏎ and return an **empty** line at once.

INPUT · STEP THROUGH THE CODE

Code Trace: `next()` and `nextLine()`

Keyboard buffer (System.in)



The whole line (spaces included) is read → "Giresun Merkez"; the ⏎ is consumed but not added to the result.

Step 6 / 7

```
1 Scanner in = new Scanner(System.in);
2 String word = in.next();
3 in.nextLine();
▶ String line = in.nextLine();
5 System.out.println(word + " | " + line);
```

VARIABLES

`word = "Ada"`

`line = "Giresun Merkez"`

OUTPUT

Without line 3, the `nextLine()` on line 4 would read the leftover ⏎ and return an **empty** line at once.

Code Trace: `next()` and `nextLine()`

Keyboard buffer (System.in)



Both values are printed.

Step 7 / 7

```
1 Scanner in = new Scanner(System.in);
2 String word = in.next();
3 in.nextLine();
4 String line = in.nextLine();
▶ System.out.println(word + " | " + line);
```

VARIABLES

`word = "Ada"`

`line = "Giresun Merkez"`

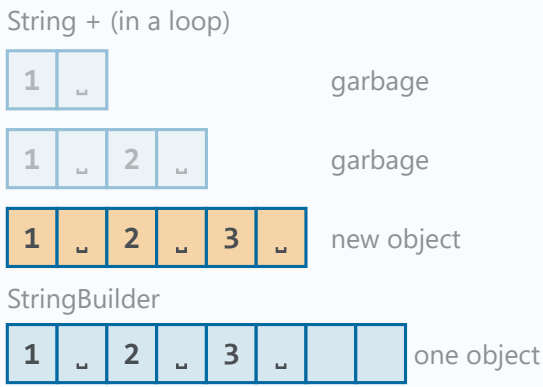
OUTPUT

```
Ada | Giresun Merkez
```

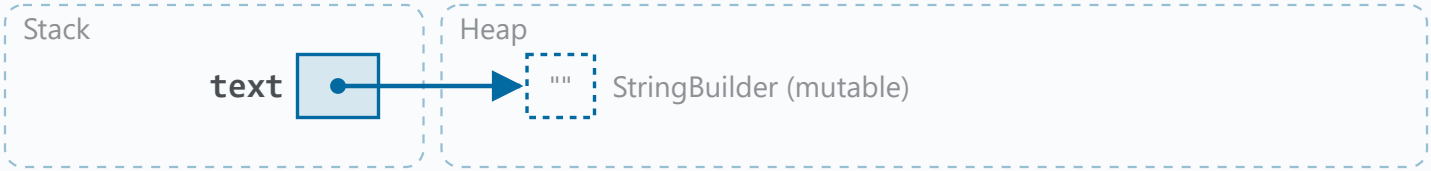
Without line 3, the `nextLine()` on line 4 would read the leftover ⏎ and return an **empty** line at once.

Concatenation and StringBuilder

- `+` reads well for joining a few strings.
- In a loop, `s = s + x` creates a **new** String on every pass and copies the old one.
- `StringBuilder` is a mutable buffer: `append` adds to the same object.
- `toString()` builds the final String from the buffer's current contents.



Code Trace: Joining with `StringBuilder`



An empty, mutable buffer is created.

Step 1 / 10

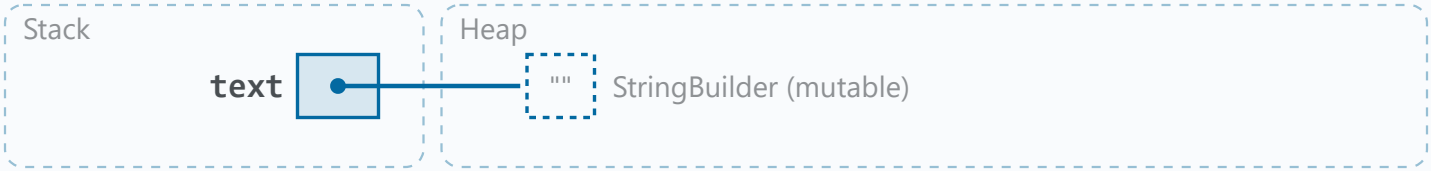
```
▶ StringBuilder text = new StringBuilder();
2 for (int i = 1; i <= 3; i++) {
3     text.append(i).append(' ');
4 }
5 String result = text.toString();
6 System.out.println(result + "|");
```

VARIABLES

`text = ""`

OUTPUT

Code Trace: Joining with `StringBuilder`



1 <= 3 → true

Step 2 / 10

```
1 StringBuilder text = new StringBuilder();
▶ for (int i = 1; i <= 3; i++) {
3     text.append(i).append(' ');
4 }
5 String result = text.toString();
6 System.out.println(result + "|");
```

VARIABLES

`text = ""` `i = 1`

OUTPUT

Code Trace: Joining with `StringBuilder`



`append(1)` and `append(' ')` add to **the same object** (no new String) → "1 ".

Step 3 / 10

```
1 StringBuilder text = new StringBuilder();
2 for (int i = 1; i <= 3; i++) {
▶     text.append(i).append(' ');
4 }
5 String result = text.toString();
6 System.out.println(result + "|");
```

VARIABLES

`text = "1 "` `i = 1`

OUTPUT

Code Trace: Joining with `StringBuilder`



`i++` → **2 <= 3 → true**

Step 4 / 10

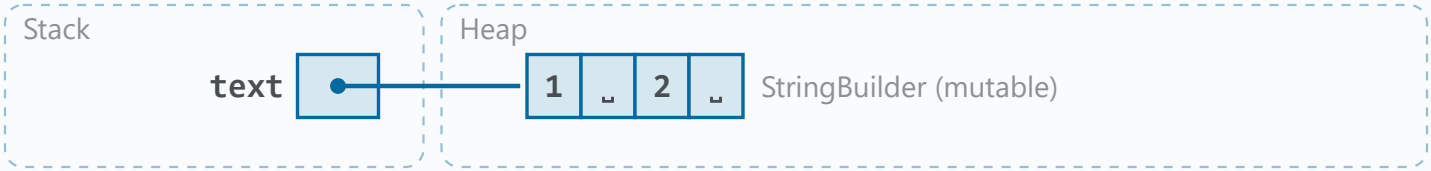
```
1 StringBuilder text = new StringBuilder();
▶ for (int i = 1; i <= 3; i++) {
3     text.append(i).append(' ');
4 }
5 String result = text.toString();
6 System.out.println(result + "|");
```

VARIABLES

`text = "1 "` `i = 2`

OUTPUT

Code Trace: Joining with `StringBuilder`



`append(2)` and `append(' ')` add to **the same object** (no new String) → "1 2 ".

Step 5 / 10

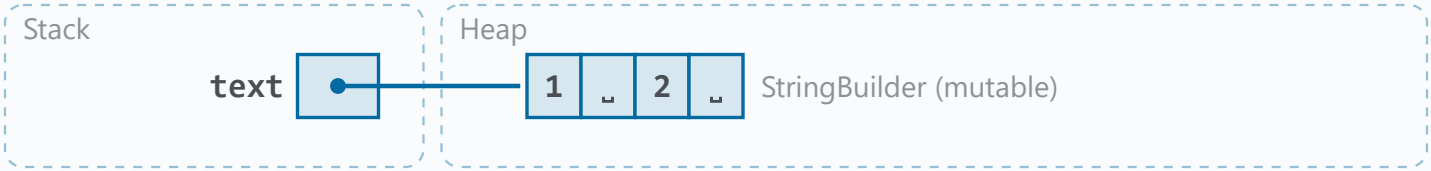
```
1 StringBuilder text = new StringBuilder();
2 for (int i = 1; i <= 3; i++) {
▶     text.append(i).append(' ');
4 }
5 String result = text.toString();
6 System.out.println(result + "|");
```

VARIABLES

`text = "1 2 "` `i = 2`

OUTPUT

Code Trace: Joining with `StringBuilder`



`i++` → 3 <= 3 → true

Step 6 / 10

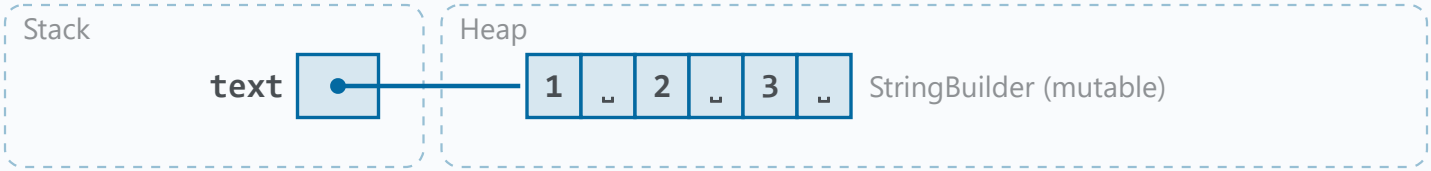
```
1 StringBuilder text = new StringBuilder();
▶ for (int i = 1; i <= 3; i++) {
3     text.append(i).append(' ');
4 }
5 String result = text.toString();
6 System.out.println(result + "|");
```

VARIABLES

`text = "1 2 "` `i = 3`

OUTPUT

Code Trace: Joining with `StringBuilder`



`append(3)` and `append(' ')` add to **the same object** (no new String) → "1 2 3 ".

Step 7 / 10

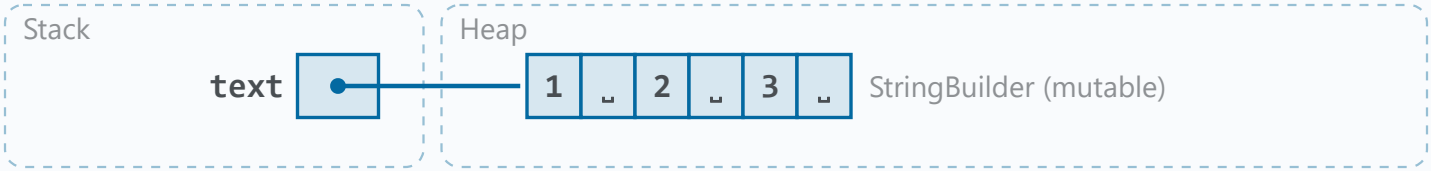
```
1 StringBuilder text = new StringBuilder();
2 for (int i = 1; i <= 3; i++) {
▶     text.append(i).append(' ');
4 }
5 String result = text.toString();
6 System.out.println(result + "|");
```

VARIABLES

`text = "1 2 3 "` `i = 3`

OUTPUT

Code Trace: Joining with `StringBuilder`



`i++` → **4 <= 3** → **false**: the loop ends.

Step 8 / 10

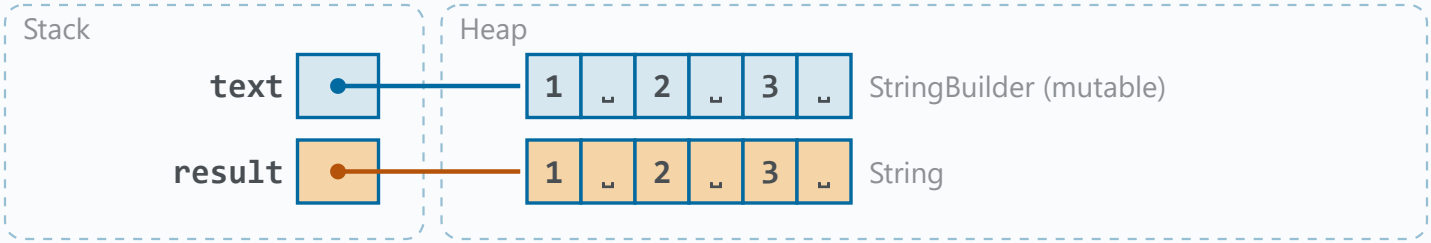
```
1 StringBuilder text = new StringBuilder();
▶ for (int i = 1; i <= 3; i++) {
3     text.append(i).append(' ');
4 }
5 String result = text.toString();
6 System.out.println(result + "|");
```

VARIABLES

`text = "1 2 3 "` `i = 4`

OUTPUT

Code Trace: Joining with `StringBuilder`



`toString()` builds a single String from the buffer.

Step 9 / 10

```
1 StringBuilder text = new StringBuilder();
2 for (int i = 1; i <= 3; i++) {
3     text.append(i).append(' ');
4 }
▶ String result = text.toString();
6 System.out.println(result + "|");
```

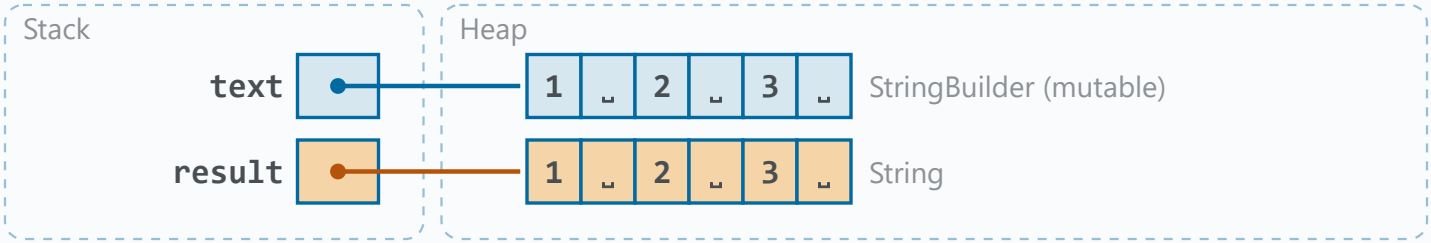
VARIABLES

`text = "1 2 3 "`

`result = "1 2 3 "`

OUTPUT

Code Trace: Joining with `StringBuilder`



A `|` is added so the trailing space shows.

Step 10 / 10

```
1 StringBuilder text = new StringBuilder();
2 for (int i = 1; i <= 3; i++) {
3     text.append(i).append(' ');
4 }
5 String result = text.toString();
▶ System.out.println(result + "|");
```

VARIABLES

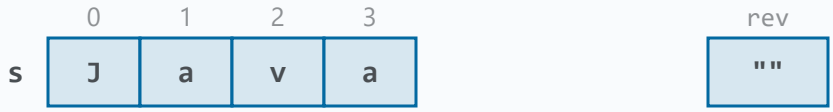
`text = "1 2 3 "`

`result = "1 2 3 "`

OUTPUT

```
1 2 3 |
```

Reversing a String



We read the string from end to start and append to a new string.

Step 1 / 9

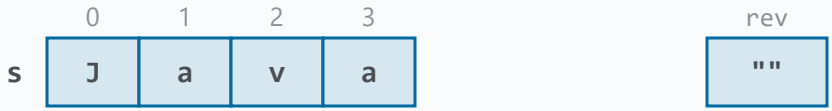
```
▶ String s = "Java";
2 String rev = "";
3 for (int i = s.length() - 1; i >= 0; i--)
4   rev += s.charAt(i);
5 System.out.println(rev);
6 String t2 = new StringBuilder(s).reverse()
7                                 .toString();
8 System.out.println(t2);
```

VARIABLES

`s = "Java"`

OUTPUT

Reversing a String



Start with the empty string.

Step 2 / 9

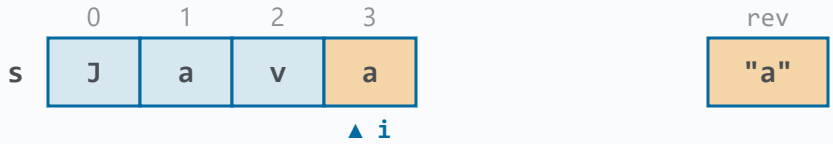
```
1 String s = "Java";
▶ String rev = "";
3 for (int i = s.length() - 1; i >= 0; i--)
4   rev += s.charAt(i);
5 System.out.println(rev);
6 String t2 = new StringBuilder(s).reverse()
7                                 .toString();
8 System.out.println(t2);
```

VARIABLES

`s = "Java"` `rev = ""`

OUTPUT

Reversing a String



i = 3: append `s.charAt(3)` = 'a' → "a" (each += makes a new String).

Step 3 / 9

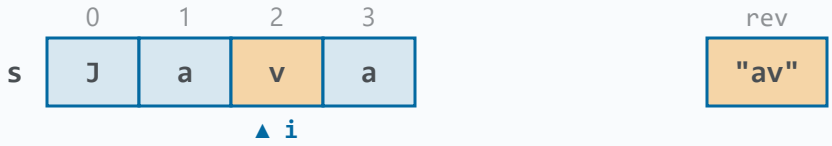
```
1 String s = "Java";
2 String rev = "";
▶ for (int i = s.length() - 1; i >= 0; i--)
▶   rev += s.charAt(i);
5 System.out.println(rev);
6 String t2 = new StringBuilder(s).reverse()
7                                 .toString();
8 System.out.println(t2);
```

VARIABLES

`s = "Java"` `rev = "a"` `i = 3`

OUTPUT

Reversing a String



i = 2: append `s.charAt(2)` = 'v' → "av" (each += makes a new String).

Step 4 / 9

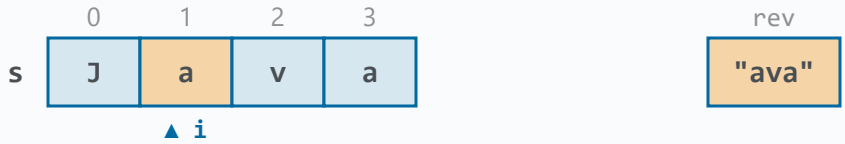
```
1 String s = "Java";
2 String rev = "";
▶ for (int i = s.length() - 1; i >= 0; i--)
▶   rev += s.charAt(i);
5 System.out.println(rev);
6 String t2 = new StringBuilder(s).reverse()
7                                 .toString();
8 System.out.println(t2);
```

VARIABLES

`s = "Java"` `rev = "av"` `i = 2`

OUTPUT

Reversing a String



i = 1: append `s.charAt(1)` = 'a' → "ava" (each += makes a new String).

Step 5 / 9

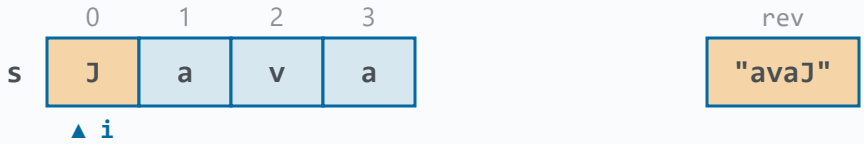
```
1 String s = "Java";
2 String rev = "";
▶ for (int i = s.length() - 1; i >= 0; i--)
▶   rev += s.charAt(i);
5 System.out.println(rev);
6 String t2 = new StringBuilder(s).reverse()
7                                 .toString();
8 System.out.println(t2);
```

VARIABLES

`s = "Java"` `rev = "ava"` `i = 1`

OUTPUT

Reversing a String



i = 0: append `s.charAt(0)` = 'J' → "avaJ" (each += makes a new String).

Step 6 / 9

```
1 String s = "Java";
2 String rev = "";
▶ for (int i = s.length() - 1; i >= 0; i--)
▶   rev += s.charAt(i);
5 System.out.println(rev);
6 String t2 = new StringBuilder(s).reverse()
7                                 .toString();
8 System.out.println(t2);
```

VARIABLES

`s = "Java"` `rev = "avaJ"` `i = 0`

OUTPUT

Reversing a String



i = −1 → the loop ends; the result is printed.

Step 7 / 9

```
1 String s = "Java";
2 String rev = "";
▶ for (int i = s.length() - 1; i >= 0; i--)
4   rev += s.charAt(i);
▶ System.out.println(rev);
6 String t2 = new StringBuilder(s).reverse()
7                                 .toString();
8 System.out.println(t2);
```

VARIABLES

`s = "Java"` `rev = "avaJ"`

OUTPUT

```
avaJ
```

Reversing a String



Shortcut: `StringBuilder`'s built-in `reverse()` gives the same result.

Step 8 / 9

```
1 String s = "Java";
2 String rev = "";
3 for (int i = s.length() - 1; i >= 0; i--)
4   rev += s.charAt(i);
5 System.out.println(rev);
▶ String t2 = new StringBuilder(s).reverse()
▶                                 .toString();
8 System.out.println(t2);
```

VARIABLES

`s = "Java"` `rev = "avaJ"` `t2 = "avaJ"`

OUTPUT

```
avaJ
```

Reversing a String



Both methods produce the same output; `s` never changed.

Step 9 / 9

```
1 String s = "Java";
2 String rev = "";
3 for (int i = s.length() - 1; i >= 0; i--)
4   rev += s.charAt(i);
5 System.out.println(rev);
6 String t2 = new StringBuilder(s).reverse()
7                                 .toString();
▶ System.out.println(t2);
```

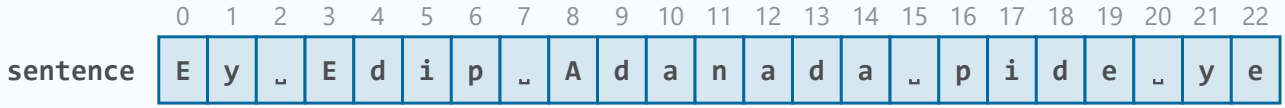
VARIABLES

`s = "Java"` `rev = "avaJ"` `t2 = "avaJ"`

OUTPUT

```
avaJ
avaJ
```

Palindromes: Ignoring Case and Spaces



The sentence has uppercase letters and spaces; compared directly, 'E' ≠ 'e'.

Step 1 / 13

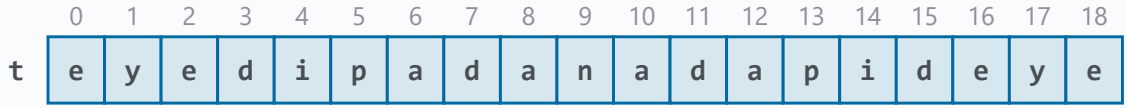
```
▶ String sentence = "Ey Edip Adanada pide ye";
2 Locale tr = Locale.forLanguageTag("tr");
3 String t = sentence.toLowerCase(tr).replace(" ", "");
4 int left = 0, right = t.length() - 1;
5 boolean isPal = true;
6 while (left < right && isPal) {
7   if (t.charAt(left) != t.charAt(right))
8     isPal = false;
9   left++; right--;
10 }
11 System.out.println(isPal);
```

VARIABLES

sentence = "Ey Edip Adanada pide ye"

OUTPUT

Palindromes: Ignoring Case and Spaces



`toLowerCase(tr)` (Turkish rules) and `replace(" ", "")` → "eyedipadanadapideye" (19 characters).

Step 2 / 13

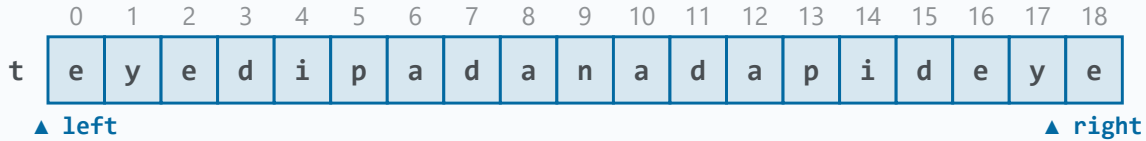
```
1  String sentence = "Ey Edip Adanada pide ye";
▶ Locale tr = Locale.forLanguageTag("tr");
▶ String t = sentence.toLowerCase(tr).replace(" ", "");
4  int left = 0, right = t.length() - 1;
5  boolean isPal = true;
6  while (left < right && isPal) {
7    if (t.charAt(left) != t.charAt(right))
8      isPal = false;
9    left++; right--;
10 }
11 System.out.println(isPal);
```

VARIABLES

`t = "eyedipadanadapideye"`

OUTPUT

Palindromes: Ignoring Case and Spaces



Two end pointers: left = 0, right = 18. Assume isPal = true until proven otherwise.

Step 3 / 13

```
1  String sentence = "Ey Edip Adanada pide ye";
2  Locale tr = Locale.forLanguageTag("tr");
3  String t = sentence.toLowerCase(tr).replace(" ", "");
▶  int left = 0, right = t.length() - 1;
▶  boolean isPal = true;
6  while (left < right && isPal) {
7    if (t.charAt(left) != t.charAt(right))
8      isPal = false;
9    left++; right--;
10 }
11 System.out.println(isPal);
```

VARIABLES

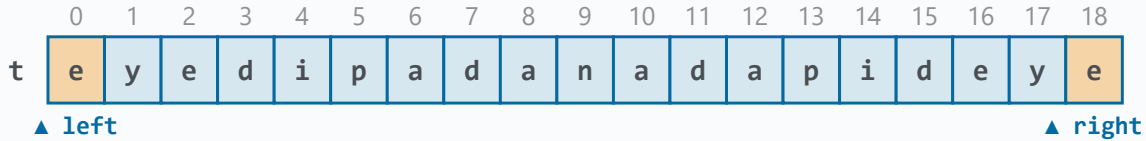
`t = "eyedipadanadapideye"`

`left = 0` `right = 18`

`isPal = true`

OUTPUT

Palindromes: Ignoring Case and Spaces



'e' and 'e' → **equal**; move inward.

Step 4 / 13

```
1  String sentence = "Ey Edip Adanada pide ye";
2  Locale tr = Locale.forLanguageTag("tr");
3  String t = sentence.toLowerCase(tr).replace(" ", "");
4  int left = 0, right = t.length() - 1;
5  boolean isPal = true;
▶  while (left < right && isPal) {
▶    if (t.charAt(left) != t.charAt(right))
8      isPal = false;
9    left++; right--;
10 }
11 System.out.println(isPal);
```

VARIABLES

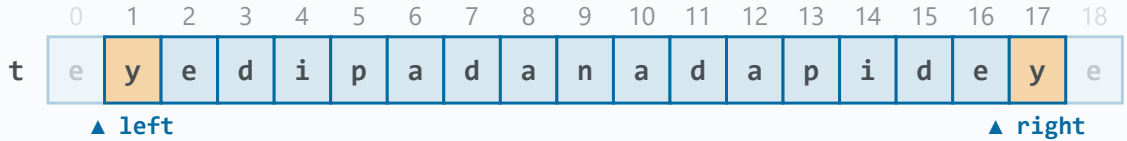
`t = "eyedipadanadapideye"`

`left = 0` `right = 18`

`isPal = true`

OUTPUT

Palindromes: Ignoring Case and Spaces



'y' and 'y' → **equal**; move inward.

Step 5 / 13

```
1  String sentence = "Ey Edip Adanada pide ye";
2  Locale tr = Locale.forLanguageTag("tr");
3  String t = sentence.toLowerCase(tr).replace(" ", "");
4  int left = 0, right = t.length() - 1;
5  boolean isPal = true;
▶  while (left < right && isPal) {
▶    if (t.charAt(left) != t.charAt(right))
8      isPal = false;
9    left++; right--;
10 }
11 System.out.println(isPal);
```

VARIABLES

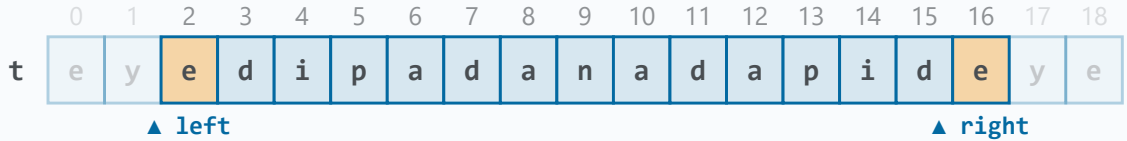
`t = "eyedipadanadapideye"`

`left = 1` `right = 17`

`isPal = true`

OUTPUT

Palindromes: Ignoring Case and Spaces



'e' and 'e' → **equal**; move inward.

Step 6 / 13

```
1  String sentence = "Ey Edip Adanada pide ye";
2  Locale tr = Locale.forLanguageTag("tr");
3  String t = sentence.toLowerCase(tr).replace(" ", "");
4  int left = 0, right = t.length() - 1;
5  boolean isPal = true;
▶  while (left < right && isPal) {
▶    if (t.charAt(left) != t.charAt(right))
8      isPal = false;
9    left++; right--;
10 }
11 System.out.println(isPal);
```

VARIABLES

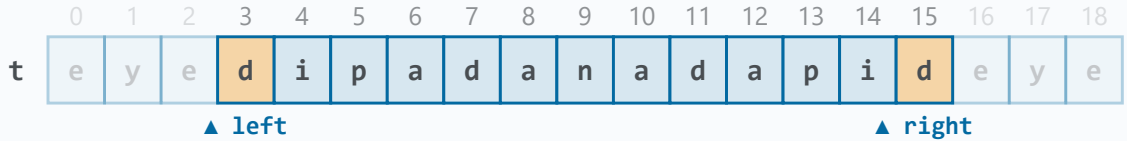
`t = "eyedipadanadapideye"`

`left = 2` `right = 16`

`isPal = true`

OUTPUT

Palindromes: Ignoring Case and Spaces



'd' and 'd' → **equal**; move inward.

Step 7 / 13

```
1  String sentence = "Ey Edip Adanada pide ye";
2  Locale tr = Locale.forLanguageTag("tr");
3  String t = sentence.toLowerCase(tr).replace(" ", "");
4  int left = 0, right = t.length() - 1;
5  boolean isPal = true;
▶  while (left < right && isPal) {
▶    if (t.charAt(left) != t.charAt(right))
8      isPal = false;
9    left++; right--;
10 }
11 System.out.println(isPal);
```

VARIABLES

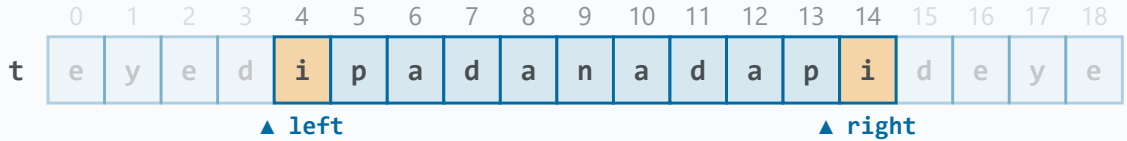
`t = "eyedipadanadapideye"`

`left = 3` `right = 15`

`isPal = true`

OUTPUT

Palindromes: Ignoring Case and Spaces



'i' and 'i' → **equal**; move inward.

Step 8 / 13

```
1  String sentence = "Ey Edip Adanada pide ye";
2  Locale tr = Locale.forLanguageTag("tr");
3  String t = sentence.toLowerCase(tr).replace(" ", "");
4  int left = 0, right = t.length() - 1;
5  boolean isPal = true;
▶  while (left < right && isPal) {
▶    if (t.charAt(left) != t.charAt(right))
8      isPal = false;
9    left++; right--;
10 }
11 System.out.println(isPal);
```

VARIABLES

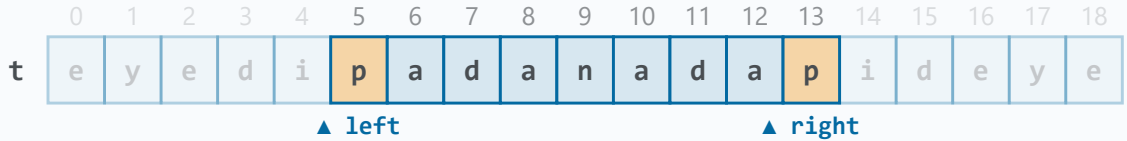
`t = "eyedipadanadapideye"`

`left = 4` `right = 14`

`isPal = true`

OUTPUT

Palindromes: Ignoring Case and Spaces



'p' and 'p' → **equal**; move inward.

Step 9 / 13

```
1  String sentence = "Ey Edip Adanada pide ye";
2  Locale tr = Locale.forLanguageTag("tr");
3  String t = sentence.toLowerCase(tr).replace(" ", "");
4  int left = 0, right = t.length() - 1;
5  boolean isPal = true;
▶  while (left < right && isPal) {
▶    if (t.charAt(left) != t.charAt(right))
8      isPal = false;
9    left++; right--;
10 }
11 System.out.println(isPal);
```

VARIABLES

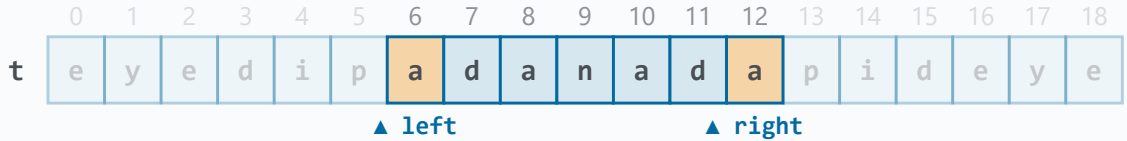
`t = "eyedipadanadapideye"`

`left = 5` `right = 13`

`isPal = true`

OUTPUT

Palindromes: Ignoring Case and Spaces



'a' and 'a' → **equal**; move inward.

Step 10 / 13

```
1  String sentence = "Ey Edip Adanada pide ye";
2  Locale tr = Locale.forLanguageTag("tr");
3  String t = sentence.toLowerCase(tr).replace(" ", "");
4  int left = 0, right = t.length() - 1;
5  boolean isPal = true;
▶  while (left < right && isPal) {
▶    if (t.charAt(left) != t.charAt(right))
8      isPal = false;
9    left++; right--;
10 }
11 System.out.println(isPal);
```

VARIABLES

`t = "eyedipadanadapideye"`

`left = 6` `right = 12`

`isPal = true`

OUTPUT

Palindromes: Ignoring Case and Spaces



'd' and 'd' → **equal**; move inward.

Step 11 / 13

```
1  String sentence = "Ey Edip Adanada pide ye";
2  Locale tr = Locale.forLanguageTag("tr");
3  String t = sentence.toLowerCase(tr).replace(" ", "");
4  int left = 0, right = t.length() - 1;
5  boolean isPal = true;
▶  while (left < right && isPal) {
▶    if (t.charAt(left) != t.charAt(right))
8      isPal = false;
9    left++; right--;
10 }
11 System.out.println(isPal);
```

VARIABLES

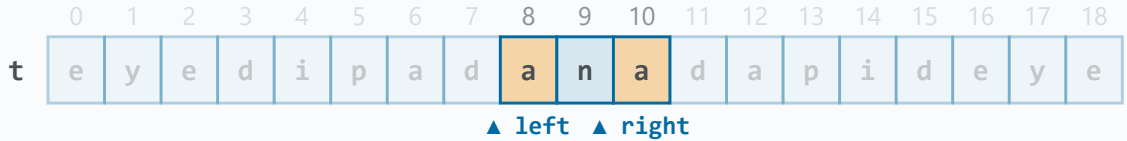
`t = "eyedipadanadapideye"`

`left = 7` `right = 11`

`isPal = true`

OUTPUT

Palindromes: Ignoring Case and Spaces



'a' and 'a' → **equal**; move inward.

Step 12 / 13

```
1  String sentence = "Ey Edip Adanada pide ye";
2  Locale tr = Locale.forLanguageTag("tr");
3  String t = sentence.toLowerCase(tr).replace(" ", "");
4  int left = 0, right = t.length() - 1;
5  boolean isPal = true;
▶  while (left < right && isPal) {
▶    if (t.charAt(left) != t.charAt(right))
8      isPal = false;
9    left++; right--;
10 }
11 System.out.println(isPal);
```

VARIABLES

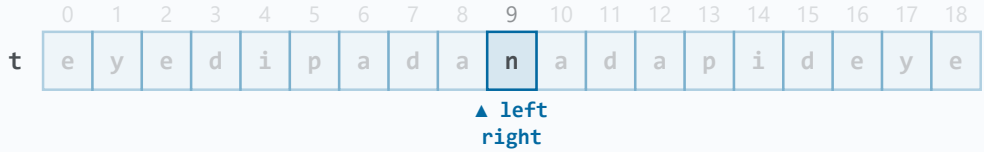
`t = "eyedipadanadapideye"`

`left = 8` `right = 10`

`isPal = true`

OUTPUT

Palindromes: Ignoring Case and Spaces



left = 9, right = 9: the pointers met and every pair matched → **true**.

Step 13 / 13

```
1  String sentence = "Ey Edip Adanada pide ye";
2  Locale tr = Locale.forLanguageTag("tr");
3  String t = sentence.toLowerCase(tr).replace(" ", "");
4  int left = 0, right = t.length() - 1;
5  boolean isPal = true;
6  while (left < right && isPal) {
7    if (t.charAt(left) != t.charAt(right))
8      isPal = false;
9    left++; right--;
10 }
▶  System.out.println(isPal);
```

VARIABLES

`t = "eyedipadanadapideye"`

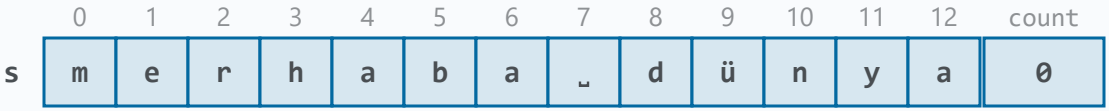
`left = 9` `right = 9`

`isPal = true`

OUTPUT

```
true
```

Counting Vowels



First the text is lowercased with Turkish rules (`tr`); the `vowels` string holds the Turkish vowels.

Step 1 / 15

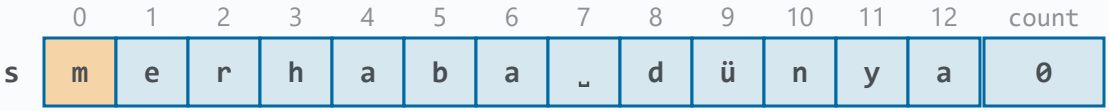
```
Locale tr = Locale.forLanguageTag("tr");
String s = "Merhaba Dünya".toLowerCase(tr);
String vowels = "aeıioöuü";
int count = 0;
for (int i = 0; i < s.length(); i++) {
  char c = s.charAt(i);
  if (vowels.indexOf(c) >= 0) count++;
}
System.out.println(count);
```

VARIABLES

`s = "merhaba dünya"` `count = 0`

OUTPUT

Counting Vowels



'm': `indexOf` → -1 → not a vowel.

Step 2 / 15

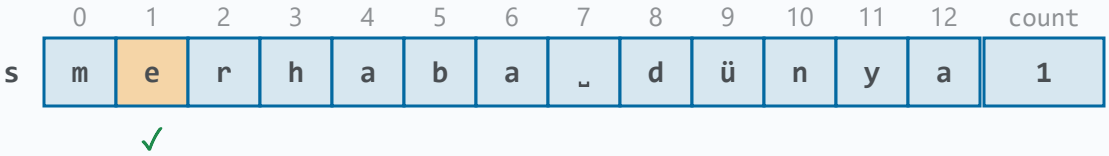
```
1 Locale tr = Locale.forLanguageTag("tr");
2 String s = "Merhaba Dünya".toLowerCase(tr);
3 String vowels = "aeıioöuü";
4 int count = 0;
5 for (int i = 0; i < s.length(); i++) {
▶   char c = s.charAt(i);
▶   if (vowels.indexOf(c) >= 0) count++;
8 }
9 System.out.println(count);
```

VARIABLES

`s = "merhaba dünya"` `count = 0` `i = 0` `c = 'm'`

OUTPUT

Counting Vowels



'e': `indexOf` → 1 ≥ 0 → vowel, count = 1.

Step 3 / 15

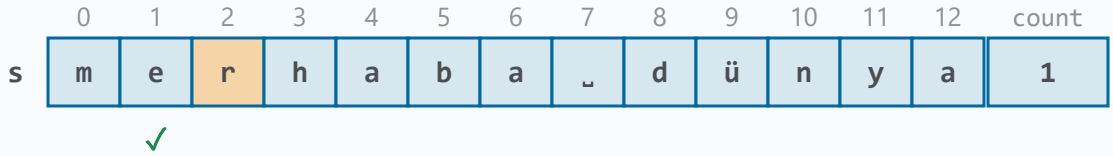
```
1 Locale tr = Locale.forLanguageTag("tr");
2 String s = "Merhaba Dünya".toLowerCase(tr);
3 String vowels = "aeıioöuü";
4 int count = 0;
5 for (int i = 0; i < s.length(); i++) {
▶   char c = s.charAt(i);
▶   if (vowels.indexOf(c) >= 0) count++;
8 }
9 System.out.println(count);
```

VARIABLES

`s = "merhaba dünya"` `count = 1` `i = 1` `c = 'e'`

OUTPUT

Counting Vowels



'r': `indexOf` → -1 → not a vowel.

Step 4 / 15

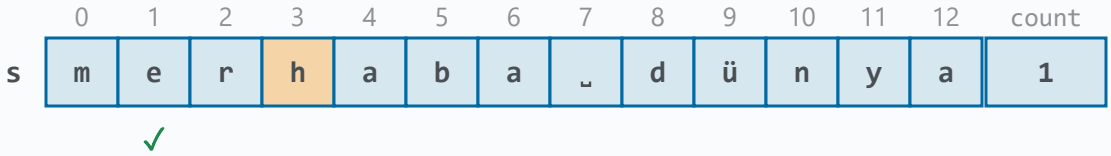
```
1 Locale tr = Locale.forLanguageTag("tr");
2 String s = "Merhaba Dünya".toLowerCase(tr);
3 String vowels = "aeıioöuü";
4 int count = 0;
5 for (int i = 0; i < s.length(); i++) {
▶   char c = s.charAt(i);
▶   if (vowels.indexOf(c) >= 0) count++;
8 }
9 System.out.println(count);
```

VARIABLES

`s = "merhaba dünya"` `count = 1` `i = 2` `c = 'r'`

OUTPUT

Counting Vowels



'h': `indexOf` → -1 → not a vowel.

Step 5 / 15

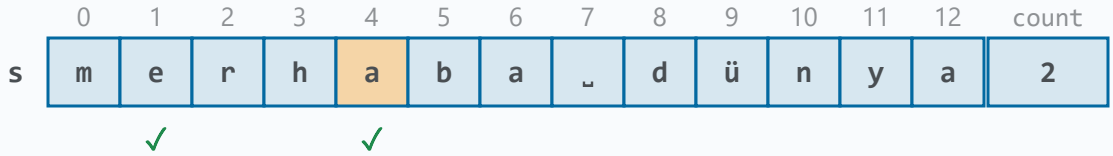
```
1 Locale tr = Locale.forLanguageTag("tr");
2 String s = "Merhaba Dünya".toLowerCase(tr);
3 String vowels = "aeıioöuü";
4 int count = 0;
5 for (int i = 0; i < s.length(); i++) {
▶   char c = s.charAt(i);
▶   if (vowels.indexOf(c) >= 0) count++;
8 }
9 System.out.println(count);
```

VARIABLES

`s = "merhaba dünya"` `count = 1` `i = 3` `c = 'h'`

OUTPUT

Counting Vowels



'a': `indexOf` → 0 ≥ 0 → vowel, count = 2.

Step 6 / 15

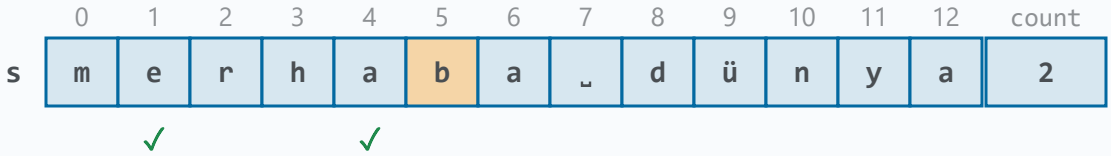
```
1 Locale tr = Locale.forLanguageTag("tr");
2 String s = "Merhaba Dünya".toLowerCase(tr);
3 String vowels = "aeıioöuü";
4 int count = 0;
5 for (int i = 0; i < s.length(); i++) {
▶   char c = s.charAt(i);
▶   if (vowels.indexOf(c) >= 0) count++;
8 }
9 System.out.println(count);
```

VARIABLES

`s = "merhaba dünya"` `count = 2` `i = 4` `c = 'a'`

OUTPUT

Counting Vowels



'b': `indexOf` → -1 → not a vowel.

Step 7 / 15

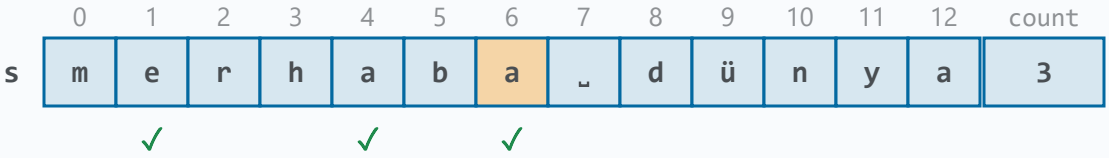
```
1 Locale tr = Locale.forLanguageTag("tr");
2 String s = "Merhaba Dünya".toLowerCase(tr);
3 String vowels = "aeıioöuü";
4 int count = 0;
5 for (int i = 0; i < s.length(); i++) {
▶   char c = s.charAt(i);
▶   if (vowels.indexOf(c) >= 0) count++;
8 }
9 System.out.println(count);
```

VARIABLES

`s = "merhaba dünya"` `count = 2` `i = 5` `c = 'b'`

OUTPUT

Counting Vowels



'a': `indexOf` → 0 ≥ 0 → vowel, count = 3.

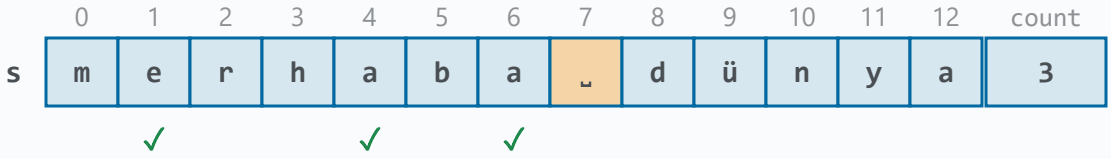

```
1 Locale tr = Locale.forLanguageTag("tr");
2 String s = "Merhaba Dünya".toLowerCase(tr);
3 String vowels = "aeıioöuü";
4 int count = 0;
5 for (int i = 0; i < s.length(); i++) {
▶   char c = s.charAt(i);
▶   if (vowels.indexOf(c) >= 0) count++;
8 }
9 System.out.println(count);
```

VARIABLES

`s = "merhaba dünya"` `count = 3` `i = 6` `c = 'a'`

OUTPUT

Counting Vowels



'␣': `indexOf` → -1 → not a vowel.

Step 9 / 15

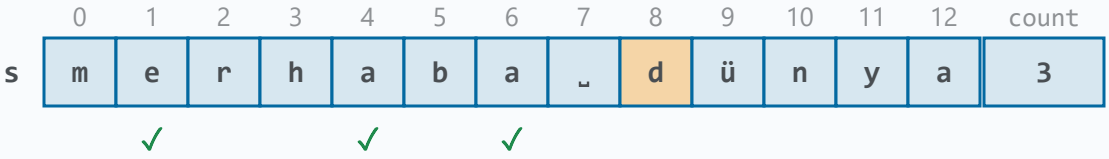
```
1 Locale tr = Locale.forLanguageTag("tr");
2 String s = "Merhaba Dünya".toLowerCase(tr);
3 String vowels = "aeıioöuü";
4 int count = 0;
5 for (int i = 0; i < s.length(); i++) {
▶   char c = s.charAt(i);
▶   if (vowels.indexOf(c) >= 0) count++;
8 }
9 System.out.println(count);
```

VARIABLES

`s = "merhaba dünya"` `count = 3` `i = 7` `c = ' '`

OUTPUT

Counting Vowels



'd': `indexOf` → -1 → not a vowel.

Step 10 / 15

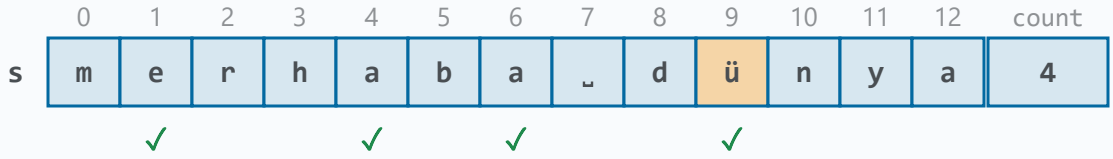
```
1 Locale tr = Locale.forLanguageTag("tr");
2 String s = "Merhaba Dünya".toLowerCase(tr);
3 String vowels = "aeıioöuü";
4 int count = 0;
5 for (int i = 0; i < s.length(); i++) {
▶   char c = s.charAt(i);
▶   if (vowels.indexOf(c) >= 0) count++;
8 }
9 System.out.println(count);
```

VARIABLES

`s = "merhaba dünya"` `count = 3` `i = 8` `c = 'd'`

OUTPUT

Counting Vowels



'ü': `indexOf` → 7 ≥ 0 → vowel, count = 4.

Step 11 / 15

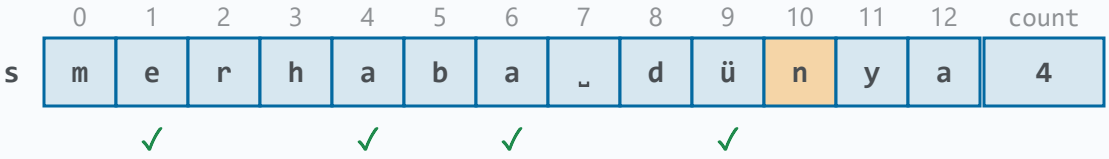
```
1 Locale tr = Locale.forLanguageTag("tr");
2 String s = "Merhaba Dünya".toLowerCase(tr);
3 String vowels = "aeıioöuü";
4 int count = 0;
5 for (int i = 0; i < s.length(); i++) {
▶   char c = s.charAt(i);
▶   if (vowels.indexOf(c) >= 0) count++;
8 }
9 System.out.println(count);
```

VARIABLES

`s = "merhaba dünya"` `count = 4` `i = 9` `c = 'ü'`

OUTPUT

Counting Vowels



'n': `indexOf` → -1 → not a vowel.

Step 12 / 15

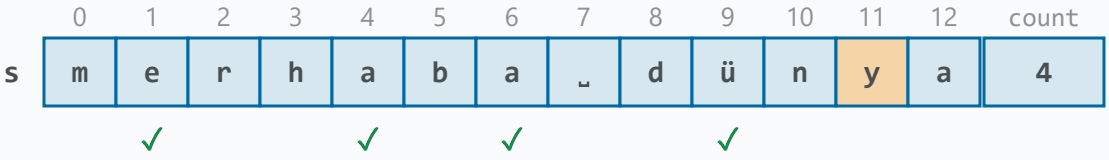
```
1 Locale tr = Locale.forLanguageTag("tr");
2 String s = "Merhaba Dünya".toLowerCase(tr);
3 String vowels = "aeıioöuü";
4 int count = 0;
5 for (int i = 0; i < s.length(); i++) {
▶   char c = s.charAt(i);
▶   if (vowels.indexOf(c) >= 0) count++;
8 }
9 System.out.println(count);
```

VARIABLES

`s = "merhaba dünya"` `count = 4` `i = 10` `c = 'n'`

OUTPUT

Counting Vowels



'y': `indexOf` → -1 → not a vowel.

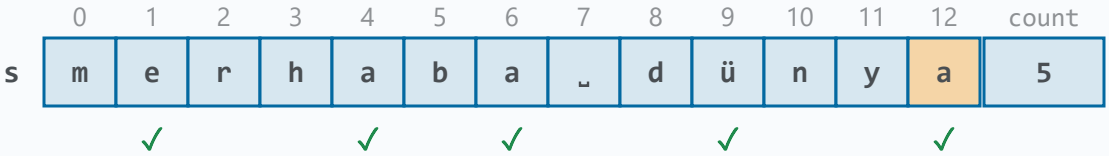

```
1 Locale tr = Locale.forLanguageTag("tr");
2 String s = "Merhaba Dünya".toLowerCase(tr);
3 String vowels = "aeıioöuü";
4 int count = 0;
5 for (int i = 0; i < s.length(); i++) {
▶   char c = s.charAt(i);
▶   if (vowels.indexOf(c) >= 0) count++;
8 }
9 System.out.println(count);
```

VARIABLES

`s = "merhaba dünya"` `count = 4` `i = 11` `c = 'y'`

OUTPUT

Counting Vowels



'a': `indexOf` → 0 ≥ 0 → vowel, count = 5.

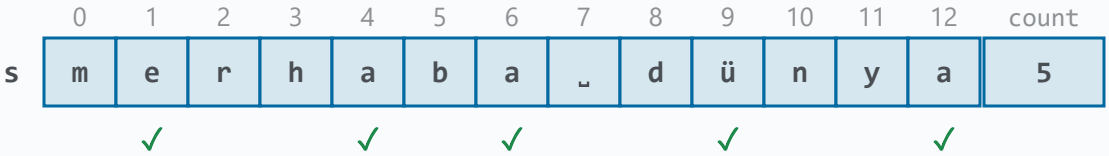

```
1 Locale tr = Locale.forLanguageTag("tr");
2 String s = "Merhaba Dünya".toLowerCase(tr);
3 String vowels = "aeıioöuü";
4 int count = 0;
5 for (int i = 0; i < s.length(); i++) {
▶   char c = s.charAt(i);
▶   if (vowels.indexOf(c) >= 0) count++;
8 }
9 System.out.println(count);
```

VARIABLES

`s = "merhaba dünya"` `count = 5` `i = 12` `c = 'a'`

OUTPUT

Counting Vowels



13 characters scanned: **5** vowels.

Step 15 / 15

```
1  Locale tr = Locale.forLanguageTag("tr");
2  String s = "Merhaba Dünya".toLowerCase(tr);
3  String vowels = "aeıioöuü";
4  int count = 0;
5  for (int i = 0; i < s.length(); i++) {
6    char c = s.charAt(i);
7    if (vowels.indexOf(c) >= 0) count++;
8  }
▶  System.out.println(count);
```

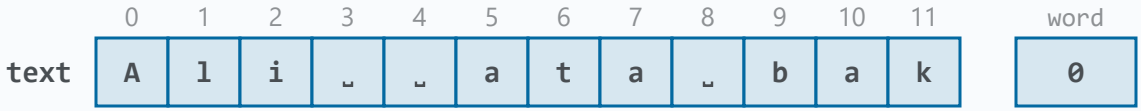
VARIABLES

`s = "merhaba dünya"` `count = 5`

OUTPUT

```
5
```

Counting Words (space-to-letter transitions)



A word is an unbroken run of non-space characters. We count the **transitions from space to letter**.

Step 1 / 14

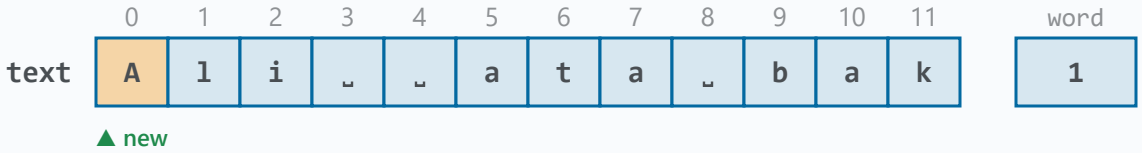
```
String text = "Ali  ata bak";
int word = 0;
boolean inWord = false;
for (int i = 0; i < text.length(); i++) {
  char c = text.charAt(i);
  if (c != ' ' && !inWord) {
    word++;
    inWord = true;
  } else if (c == ' ') inWord = false;
}
System.out.println(word);
```

VARIABLES

`text = "Ali ata bak"` `word = 0` `inWord = false`

OUTPUT

Counting Words (space-to-letter transitions)



'A' is a letter and we were outside a word → **new word**: word = 1.

Step 2 / 14

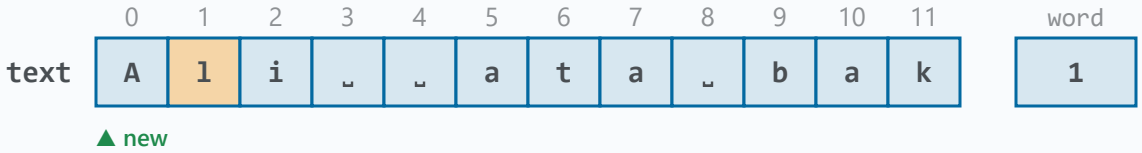
```
1  String text = "Ali  ata bak";
2  int word = 0;
3  boolean inWord = false;
4  for (int i = 0; i < text.length(); i++) {
5    char c = text.charAt(i);
▶    if (c != ' ' && !inWord) {
▶      word++;
▶      inWord = true;
9    } else if (c == ' ') inWord = false;
10 }
11 System.out.println(word);
```

VARIABLES

`text = "Ali ata bak"` `word = 1` `inWord = true` `i = 0`

OUTPUT

Counting Words (space-to-letter transitions)



'l' continues the same word.

Step 3 / 14

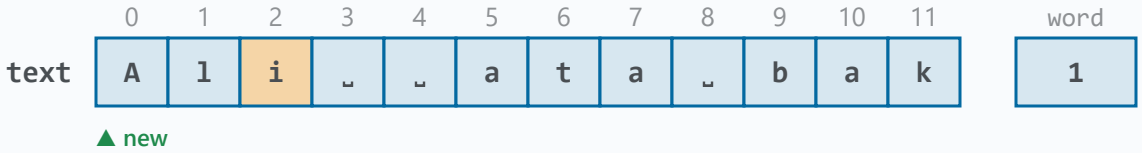
```
1  String text = "Ali  ata bak";
2  int word = 0;
3  boolean inWord = false;
4  for (int i = 0; i < text.length(); i++) {
5    char c = text.charAt(i);
▶    if (c != ' ' && !inWord) {
7      word++;
8      inWord = true;
9    } else if (c == ' ') inWord = false;
10 }
11 System.out.println(word);
```

VARIABLES

`text = "Ali ata bak"` `word = 1` `inWord = true` `i = 1`

OUTPUT

Counting Words (space-to-letter transitions)



'i' continues the same word.

Step 4 / 14

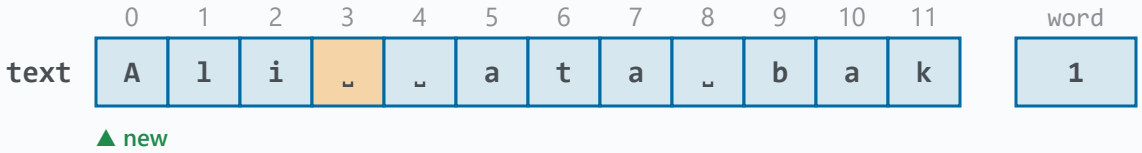
```
1  String text = "Ali  ata bak";
2  int word = 0;
3  boolean inWord = false;
4  for (int i = 0; i < text.length(); i++) {
5    char c = text.charAt(i);
▶    if (c != ' ' && !inWord) {
7      word++;
8      inWord = true;
9    } else if (c == ' ') inWord = false;
10 }
11 System.out.println(word);
```

VARIABLES

`text = "Ali ata bak"` `word = 1` `inWord = true` `i = 2`

OUTPUT

Counting Words (space-to-letter transitions)



Space → we left the word.

Step 5 / 14

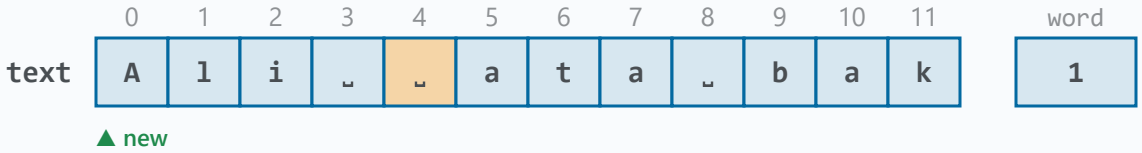
```
1  String text = "Ali  ata bak";
2  int word = 0;
3  boolean inWord = false;
4  for (int i = 0; i < text.length(); i++) {
5    char c = text.charAt(i);
6    if (c != ' ' && !inWord) {
7      word++;
8      inWord = true;
▶    } else if (c == ' ') inWord = false;
10 }
11 System.out.println(word);
```

VARIABLES

`text = "Ali ata bak"` `word = 1` `inWord = false` `i = 3`

OUTPUT

Counting Words (space-to-letter transitions)



Another space (double space) → not counted.

Step 6 / 14

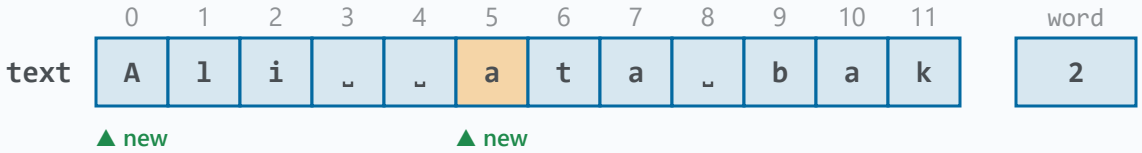
```
1  String text = "Ali  ata bak";
2  int word = 0;
3  boolean inWord = false;
4  for (int i = 0; i < text.length(); i++) {
5    char c = text.charAt(i);
6    if (c != ' ' && !inWord) {
7      word++;
8      inWord = true;
▶    } else if (c == ' ') inWord = false;
10 }
11 System.out.println(word);
```

VARIABLES

`text = "Ali ata bak"` `word = 1` `inWord = false` `i = 4`

OUTPUT

Counting Words (space-to-letter transitions)



'a' is a letter and we were outside a word → **new word**: word = 2.

Step 7 / 14

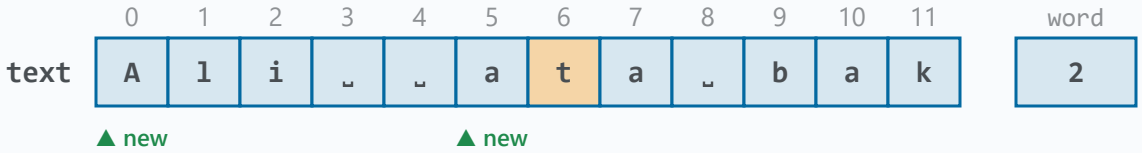
```
1  String text = "Ali  ata bak";
2  int word = 0;
3  boolean inWord = false;
4  for (int i = 0; i < text.length(); i++) {
5    char c = text.charAt(i);
▶    if (c != ' ' && !inWord) {
▶      word++;
▶      inWord = true;
9    } else if (c == ' ') inWord = false;
10 }
11 System.out.println(word);
```

VARIABLES

`text = "Ali ata bak"` `word = 2` `inWord = true` `i = 5`

OUTPUT

Counting Words (space-to-letter transitions)



't' continues the same word.

Step 8 / 14

```
1  String text = "Ali  ata bak";
2  int word = 0;
3  boolean inWord = false;
4  for (int i = 0; i < text.length(); i++) {
5    char c = text.charAt(i);
▶    if (c != ' ' && !inWord) {
7      word++;
8      inWord = true;
9    } else if (c == ' ') inWord = false;
10 }
11 System.out.println(word);
```

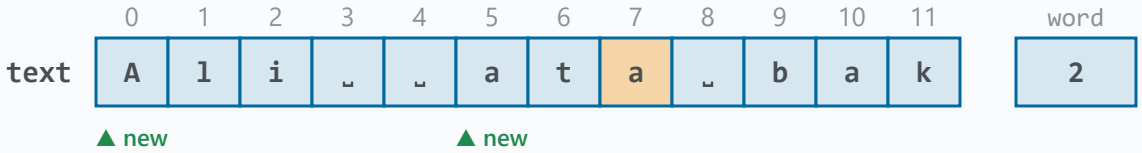
VARIABLES

`text = "Ali ata bak"` `word = 2`

`inWord = true` `i = 6`

OUTPUT

Counting Words (space-to-letter transitions)



'a' continues the same word.

Step 9 / 14

```
1  String text = "Ali  ata bak";
2  int word = 0;
3  boolean inWord = false;
4  for (int i = 0; i < text.length(); i++) {
5    char c = text.charAt(i);
▶    if (c != ' ' && !inWord) {
7      word++;
8      inWord = true;
9    } else if (c == ' ') inWord = false;
10 }
11 System.out.println(word);
```

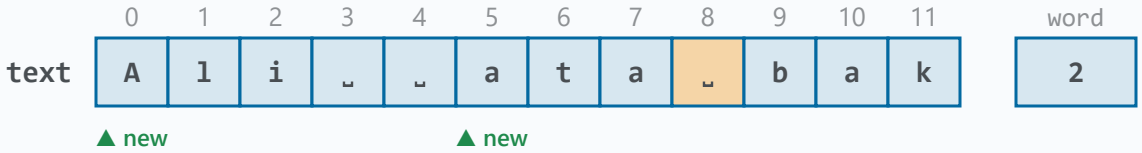
VARIABLES

`text = "Ali ata bak"` `word = 2`

`inWord = true` `i = 7`

OUTPUT

Counting Words (space-to-letter transitions)



Space → we left the word.

Step 10 / 14

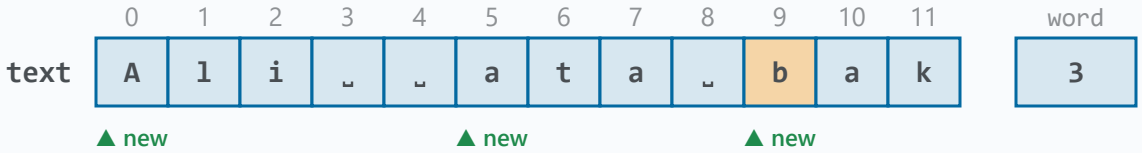
```
1  String text = "Ali  ata bak";
2  int word = 0;
3  boolean inWord = false;
4  for (int i = 0; i < text.length(); i++) {
5    char c = text.charAt(i);
6    if (c != ' ' && !inWord) {
7      word++;
8      inWord = true;
▶    } else if (c == ' ') inWord = false;
10 }
11 System.out.println(word);
```

VARIABLES

`text = "Ali ata bak"` `word = 2` `inWord = false` `i = 8`

OUTPUT

Counting Words (space-to-letter transitions)



'b' is a letter and we were outside a word → **new word**: word = 3.

Step 11 / 14

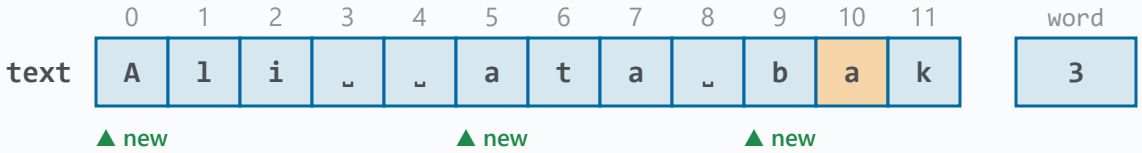
```
1  String text = "Ali  ata bak";
2  int word = 0;
3  boolean inWord = false;
4  for (int i = 0; i < text.length(); i++) {
5    char c = text.charAt(i);
▶    if (c != ' ' && !inWord) {
▶      word++;
▶      inWord = true;
9    } else if (c == ' ') inWord = false;
10 }
11 System.out.println(word);
```

VARIABLES

`text = "Ali ata bak"` `word = 3` `inWord = true` `i = 9`

OUTPUT

Counting Words (space-to-letter transitions)



'a' continues the same word.

Step 12 / 14

```
1  String text = "Ali  ata bak";
2  int word = 0;
3  boolean inWord = false;
4  for (int i = 0; i < text.length(); i++) {
5    char c = text.charAt(i);
▶    if (c != ' ' && !inWord) {
7      word++;
8      inWord = true;
9    } else if (c == ' ') inWord = false;
10 }
11 System.out.println(word);
```

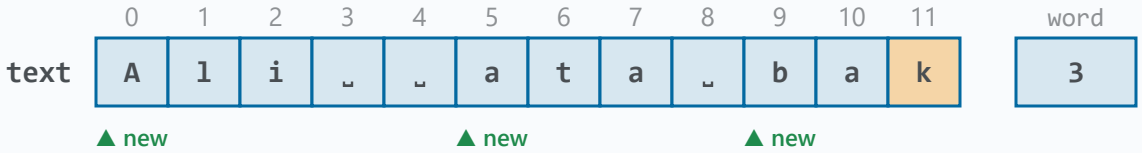
VARIABLES

`text = "Ali ata bak"` `word = 3`

`inWord = true` `i = 10`

OUTPUT

Counting Words (space-to-letter transitions)



'k' continues the same word.

Step 13 / 14

```
1  String text = "Ali  ata bak";
2  int word = 0;
3  boolean inWord = false;
4  for (int i = 0; i < text.length(); i++) {
5    char c = text.charAt(i);
▶    if (c != ' ' && !inWord) {
7      word++;
8      inWord = true;
9    } else if (c == ' ') inWord = false;
10 }
11 System.out.println(word);
```

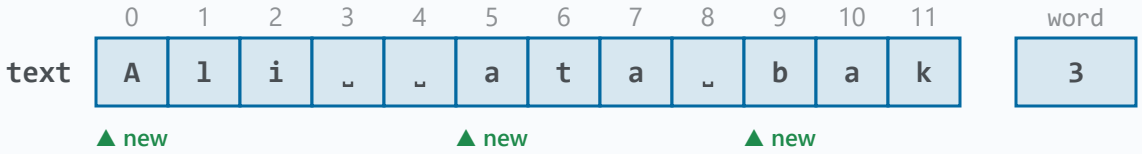
VARIABLES

`text = "Ali ata bak"` `word = 3`

`inWord = true` `i = 11`

OUTPUT

Counting Words (space-to-letter transitions)



The result is **3**. Shortcut: `text.trim().split("\\s+").length` (only for nonblank text; blank text gives 1, not 0, so check `isBlank()` first).

Step 14 / 14

```
1  String text = "Ali  ata bak";
2  int word = 0;
3  boolean inWord = false;
4  for (int i = 0; i < text.length(); i++) {
5    char c = text.charAt(i);
6    if (c != ' ' && !inWord) {
7      word++;
8      inWord = true;
9    } else if (c == ' ') inWord = false;
10 }
▶  System.out.println(word);
```

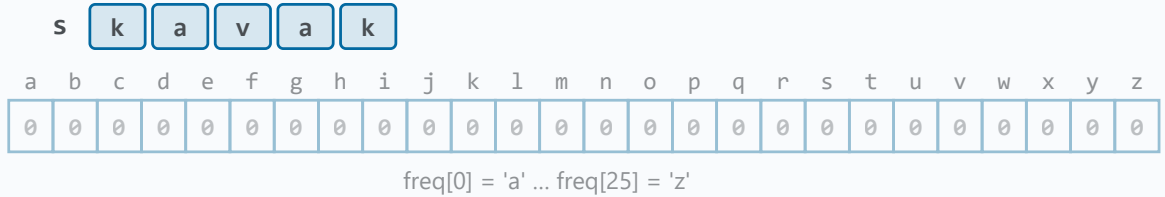
VARIABLES

`text = "Ali ata bak"` `word = 3` `inWord = true`

OUTPUT

```
3
```

Character Frequency (`int[26]`)



A 26-cell counter: **'a' → 0, 'b' → 1, …** A letter's index is `c - 'a'` (chars are numbers). Characters outside a–z are skipped with `continue`.

Step 1 / 9

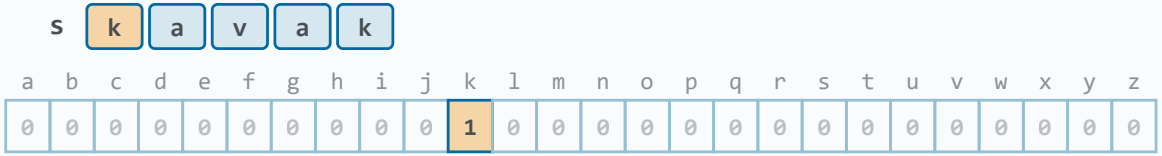
```
▶ String s = "kavak";
▶ int[] freq = new int[26];
3 for (int i = 0; i < s.length(); i++) {
4   char c = s.charAt(i);
5   if (c < 'a' || c > 'z') continue;
6   freq[c - 'a']++;
7 }
8 for (int k = 0; k < 26; k++)
9   if (freq[k] > 0)
10    System.out.println((char) ('a' + k)
11                       + ": " + freq[k]);
```

VARIABLES

`s = "kavak"`

OUTPUT

Character Frequency (`int[26]`)



'k' - 'a' = 107 − 97 = **10** → freq[10] = 1.

Step 2 / 9

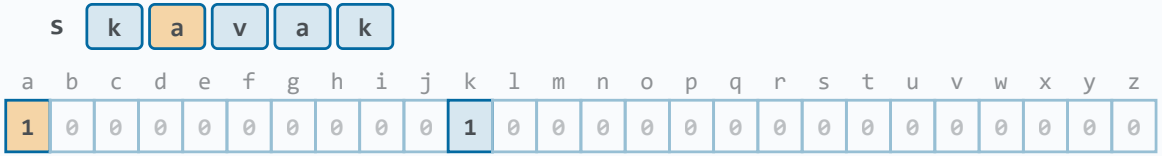
```
1  String s = "kavak";
2  int[] freq = new int[26];
3  for (int i = 0; i < s.length(); i++) {
▶    char c = s.charAt(i);
▶    if (c < 'a' || c > 'z') continue;
▶    freq[c - 'a']++;
7  }
8  for (int k = 0; k < 26; k++)
9    if (freq[k] > 0)
10     System.out.println((char) ('a' + k)
11                        + ": " + freq[k]);
```

VARIABLES

`s = "kavak"` `i = 0` `c = 'k'`

OUTPUT

Character Frequency (`int[26]`)



'a' - 'a' = 97 − 97 = **0** → freq[0] = 1.

Step 3 / 9

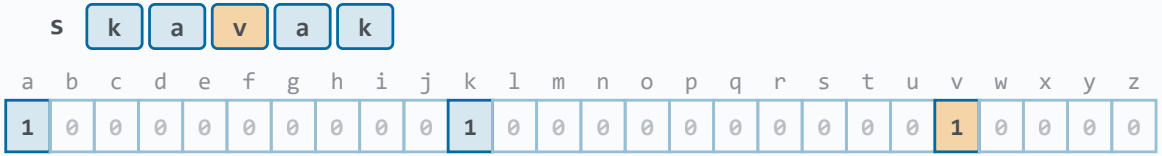
```
1  String s = "kavak";
2  int[] freq = new int[26];
3  for (int i = 0; i < s.length(); i++) {
▶    char c = s.charAt(i);
▶    if (c < 'a' || c > 'z') continue;
▶    freq[c - 'a']++;
7  }
8  for (int k = 0; k < 26; k++)
9    if (freq[k] > 0)
10     System.out.println((char) ('a' + k)
11                        + ": " + freq[k]);
```

VARIABLES

`s = "kavak"` `i = 1` `c = 'a'`

OUTPUT

Character Frequency (`int[26]`)



'v' - 'a' = 118 − 97 = **21** → freq[21] = 1.

Step 4 / 9

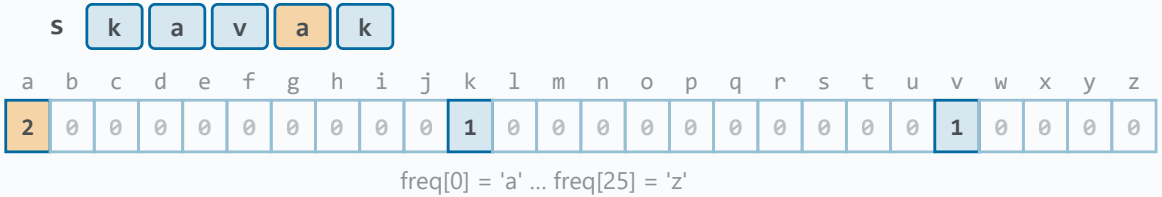
```
1  String s = "kavak";
2  int[] freq = new int[26];
3  for (int i = 0; i < s.length(); i++) {
▶    char c = s.charAt(i);
▶    if (c < 'a' || c > 'z') continue;
▶    freq[c - 'a']++;
7  }
8  for (int k = 0; k < 26; k++)
9    if (freq[k] > 0)
10     System.out.println((char) ('a' + k)
11                        + ": " + freq[k]);
```

VARIABLES

`s = "kavak"` `i = 2` `c = 'v'`

OUTPUT

Character Frequency (`int[26]`)



'a' - 'a' = 97 − 97 = **0** → freq[0] = 2.

Step 5 / 9

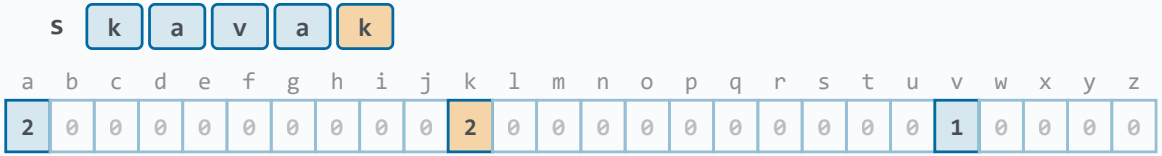
```
1  String s = "kavak";
2  int[] freq = new int[26];
3  for (int i = 0; i < s.length(); i++) {
▶    char c = s.charAt(i);
▶    if (c < 'a' || c > 'z') continue;
▶    freq[c - 'a']++;
7  }
8  for (int k = 0; k < 26; k++)
9    if (freq[k] > 0)
10     System.out.println((char) ('a' + k)
11                        + ": " + freq[k]);
```

VARIABLES

`s = "kavak"` `i = 3` `c = 'a'`

OUTPUT

Character Frequency (`int[26]`)



'k' - 'a' = 107 − 97 = **10** → freq[10] = 2.

Step 6 / 9

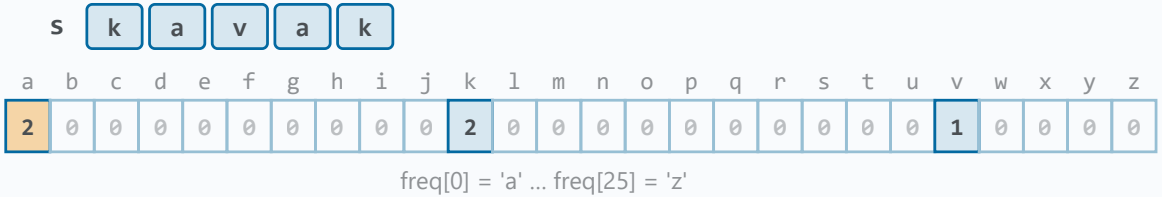
```
1  String s = "kavak";
2  int[] freq = new int[26];
3  for (int i = 0; i < s.length(); i++) {
▶    char c = s.charAt(i);
▶    if (c < 'a' || c > 'z') continue;
▶    freq[c - 'a']++;
7  }
8  for (int k = 0; k < 26; k++)
9    if (freq[k] > 0)
10     System.out.println((char) ('a' + k)
11                        + ": " + freq[k]);
```

VARIABLES

`s = "kavak"` `i = 4` `c = 'k'`

OUTPUT

Character Frequency (`int[26]`)



k = 0: freq > 0 → `(char) ('a' + 0)` = 'a' is printed.

Step 7 / 9

```
1 String s = "kavak";
2 int[] freq = new int[26];
3 for (int i = 0; i < s.length(); i++) {
4   char c = s.charAt(i);
5   if (c < 'a' || c > 'z') continue;
6   freq[c - 'a']++;
7 }
▶ for (int k = 0; k < 26; k++)
▶   if (freq[k] > 0)
▶     System.out.println((char) ('a' + k)
▶                        + ": " + freq[k]);
```

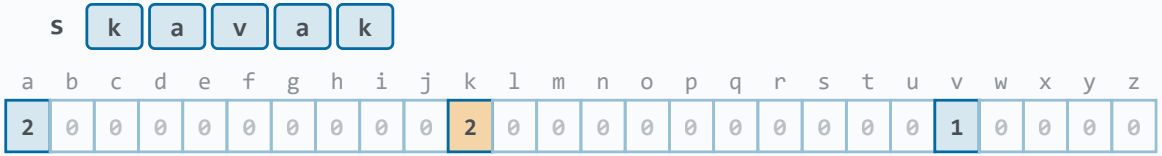
VARIABLES

`s = "kavak"` `k = 0`

OUTPUT

```
a: 2
```

Character Frequency (`int[26]`)



k = 10: freq > 0 → `(char) ('a' + 10)` = 'k' is printed.

Step 8 / 9

```
1 String s = "kavak";
2 int[] freq = new int[26];
3 for (int i = 0; i < s.length(); i++) {
4   char c = s.charAt(i);
5   if (c < 'a' || c > 'z') continue;
6   freq[c - 'a']++;
7 }
▶ for (int k = 0; k < 26; k++)
▶   if (freq[k] > 0)
▶     System.out.println((char) ('a' + k)
▶                        + ": " + freq[k]);
```

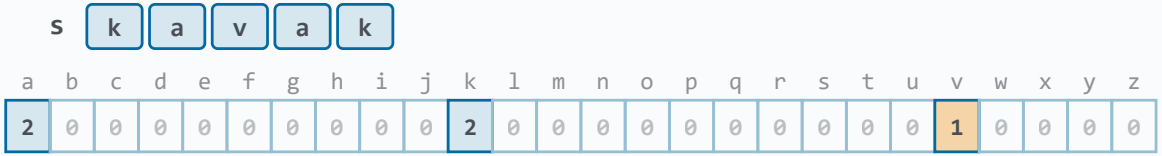
VARIABLES

`s = "kavak"` `k = 10`

OUTPUT

```
a: 2
k: 2
```

Character Frequency (`int[26]`)



k = 21: freq > 0 → `(char) ('a' + 21)` = 'v' is printed. The table covers only a–z; ç, ğ, ı, ö, ş, ü need a separate scheme.

Step 9 / 9

```
1  String s = "kavak";
2  int[] freq = new int[26];
3  for (int i = 0; i < s.length(); i++) {
4    char c = s.charAt(i);
5    if (c < 'a' || c > 'z') continue;
6    freq[c - 'a']++;
7  }
▶ for (int k = 0; k < 26; k++)
▶   if (freq[k] > 0)
▶     System.out.println((char) ('a' + k)
▶                        + ": " + freq[k]);
```

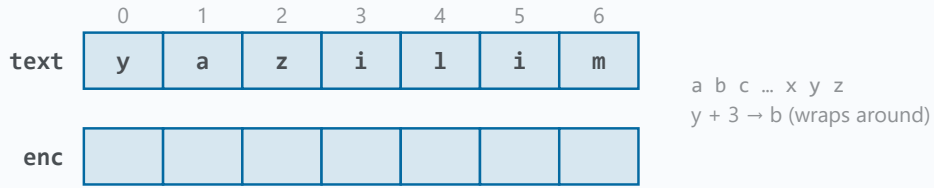
VARIABLES

`s = "kavak"` `k = 21`

OUTPUT

```
a: 2
k: 2
v: 1
```

Caesar Cipher: Encrypting and Decrypting



Each letter shifts **3** places forward in the alphabet. Only lowercase English letters (a–z) are encrypted; other characters are appended unchanged. The result builds up in a StringBuilder.

Step 1 / 9

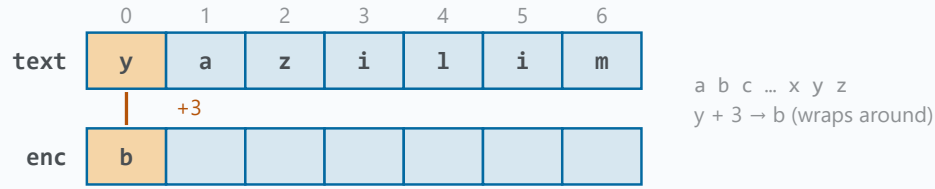
```
▶ String text = "yazilim";
▶ int shift = 3;
▶ StringBuilder enc = new StringBuilder();
4 for (int i = 0; i < text.length(); i++) {
5   char c = text.charAt(i);
6   if (c < 'a' || c > 'z') { enc.append(c); continue; }
7   int pos = (c - 'a' + shift) % 26;
8   char out = (char) ('a' + pos);
9   enc.append(out);
10 }
11 System.out.println(enc);
```

VARIABLES

`text = "yazilim"` `shift = 3` `enc = ""`

OUTPUT

Caesar Cipher: Encrypting and Decrypting



'y': (24 + 3) % 26 = **1** → 'b' (past 26, so % 26 wraps around).

Step 2 / 9

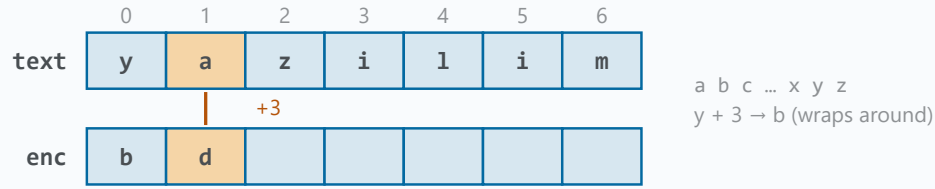
```
1  String text = "yazilim";
2  int shift = 3;
3  StringBuilder enc = new StringBuilder();
4  for (int i = 0; i < text.length(); i++) {
▶    char c = text.charAt(i);
▶    if (c < 'a' || c > 'z') { enc.append(c); continue; }
▶    int pos = (c - 'a' + shift) % 26;
▶    char out = (char) ('a' + pos);
▶    enc.append(out);
10 }
11 System.out.println(enc);
```

VARIABLES

`text = "yazilim"` `shift = 3` `enc = "b"` `c = 'y'` `pos = 1`

OUTPUT

Caesar Cipher: Encrypting and Decrypting



'a': (0 + 3) % 26 = **3** → 'd'.

Step 3 / 9

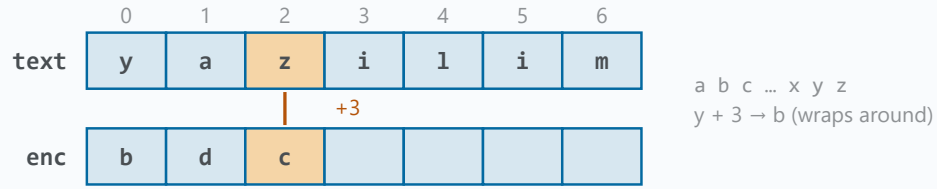
```
1  String text = "yazilim";
2  int shift = 3;
3  StringBuilder enc = new StringBuilder();
4  for (int i = 0; i < text.length(); i++) {
▶    char c = text.charAt(i);
▶    if (c < 'a' || c > 'z') { enc.append(c); continue; }
▶    int pos = (c - 'a' + shift) % 26;
▶    char out = (char) ('a' + pos);
▶    enc.append(out);
10 }
11 System.out.println(enc);
```

VARIABLES

`text = "yazilim"` `shift = 3`
`enc = "bd"` `c = 'a'` `pos = 3`

OUTPUT

Caesar Cipher: Encrypting and Decrypting



'z': (25 + 3) % 26 = **2** → 'c' (past 26, so % 26 wraps around).

Step 4 / 9

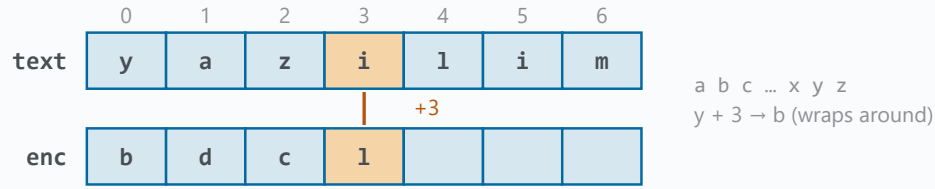
```
1  String text = "yazilim";
2  int shift = 3;
3  StringBuilder enc = new StringBuilder();
4  for (int i = 0; i < text.length(); i++) {
▶    char c = text.charAt(i);
▶    if (c < 'a' || c > 'z') { enc.append(c); continue; }
▶    int pos = (c - 'a' + shift) % 26;
▶    char out = (char) ('a' + pos);
▶    enc.append(out);
10 }
11 System.out.println(enc);
```

VARIABLES

`text = "yazilim"` `shift = 3` `enc = "bdc"` `c = 'z'` `pos = 2`

OUTPUT

Caesar Cipher: Encrypting and Decrypting



a b c … x y z
y + 3 → b (wraps around)

'i': (8 + 3) % 26 = **11** → 'l'.

Step 5 / 9

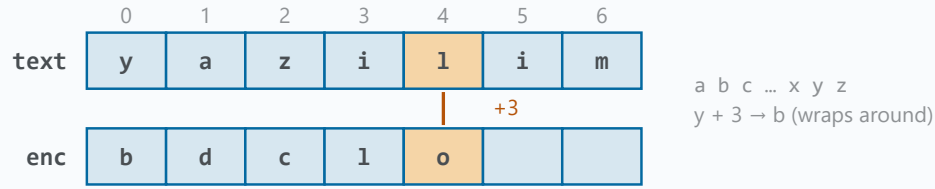
```
1  String text = "yazilim";
2  int shift = 3;
3  StringBuilder enc = new StringBuilder();
4  for (int i = 0; i < text.length(); i++) {
▶    char c = text.charAt(i);
▶    if (c < 'a' || c > 'z') { enc.append(c); continue; }
▶    int pos = (c - 'a' + shift) % 26;
▶    char out = (char) ('a' + pos);
▶    enc.append(out);
10 }
11 System.out.println(enc);
```

VARIABLES

`text = "yazilim"` `shift = 3`
`enc = "bdcl"` `c = 'i'` `pos = 11`

OUTPUT

Caesar Cipher: Encrypting and Decrypting



'l': (11 + 3) % 26 = **14** → 'o'.

Step 6 / 9

```
1 String text = "yazilim";
2 int shift = 3;
3 StringBuilder enc = new StringBuilder();
4 for (int i = 0; i < text.length(); i++) {
▶   char c = text.charAt(i);
▶   if (c < 'a' || c > 'z') { enc.append(c); continue; }
▶   int pos = (c - 'a' + shift) % 26;
▶   char out = (char) ('a' + pos);
▶   enc.append(out);
10 }
11 System.out.println(enc);
```

VARIABLES

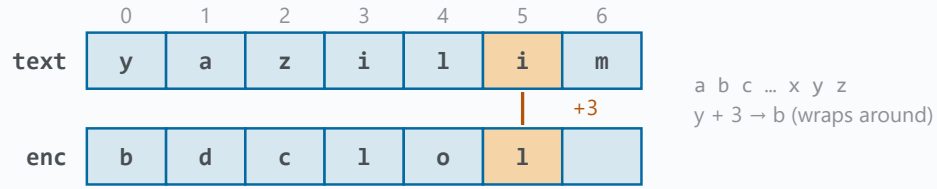
`text = "yazilim"` `shift = 3`

`enc = "bdclo"` `c = 'l'`

`pos = 14`

OUTPUT

Caesar Cipher: Encrypting and Decrypting



'i': (8 + 3) % 26 = **11** → 'l'.

Step 7 / 9

```
1 String text = "yazilim";
2 int shift = 3;
3 StringBuilder enc = new StringBuilder();
4 for (int i = 0; i < text.length(); i++) {
    char c = text.charAt(i);
    if (c < 'a' || c > 'z') { enc.append(c); continue; }
    int pos = (c - 'a' + shift) % 26;
    char out = (char) ('a' + pos);
    enc.append(out);
10 }
11 System.out.println(enc);
```

VARIABLES

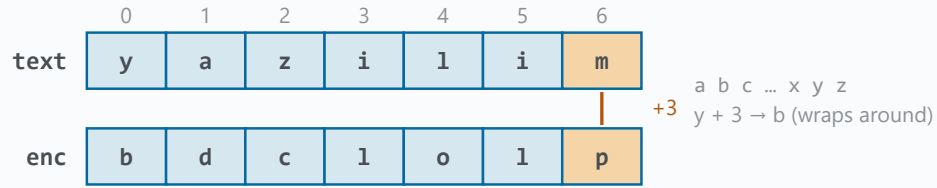
`text = "yazilim"` `shift = 3`

`enc = "bdclol"` `c = 'i'`

`pos = 11`

OUTPUT

Caesar Cipher: Encrypting and Decrypting



'm': (12 + 3) % 26 = **15** → 'p'.

Step 8 / 9

```
1  String text = "yazilim";
2  int shift = 3;
3  StringBuilder enc = new StringBuilder();
4  for (int i = 0; i < text.length(); i++) {
▶    char c = text.charAt(i);
▶    if (c < 'a' || c > 'z') { enc.append(c); continue; }
▶    int pos = (c - 'a' + shift) % 26;
▶    char out = (char) ('a' + pos);
▶    enc.append(out);
10 }
11 System.out.println(enc);
```

VARIABLES

`text = "yazilim"` `shift = 3`
`enc = "bdclolp"` `c = 'm'`
`pos = 15`

OUTPUT

Caesar Cipher: Encrypting and Decrypting



The ciphertext is **bdclolp**. **Decrypting:** the same loop with `26 - shift` instead of `shift` → "yazilim".

Step 9 / 9

```
1  String text = "yazilim";
2  int shift = 3;
3  StringBuilder enc = new StringBuilder();
4  for (int i = 0; i < text.length(); i++) {
5    char c = text.charAt(i);
6    if (c < 'a' || c > 'z') { enc.append(c); continue; }
7    int pos = (c - 'a' + shift) % 26;
8    char out = (char) ('a' + pos);
9    enc.append(out);
10 }
▶ System.out.println(enc);
```

VARIABLES

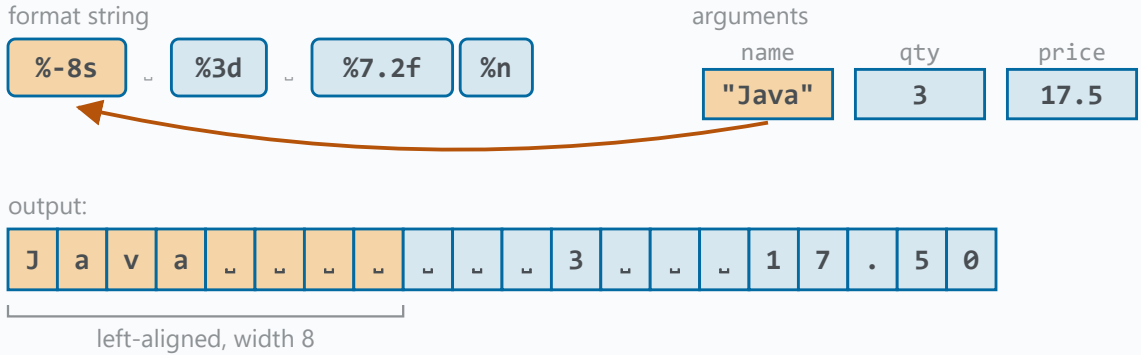
`text = "yazilim"` `shift = 3` `enc = "bdclolp"`

OUTPUT

```
bdclolp
```

Formatting with printf

```
String name = "Java"; int qty = 3; double price = 17.5;
System.out.printf("%-8s %3d %7.2f%n", name, qty, price);
```



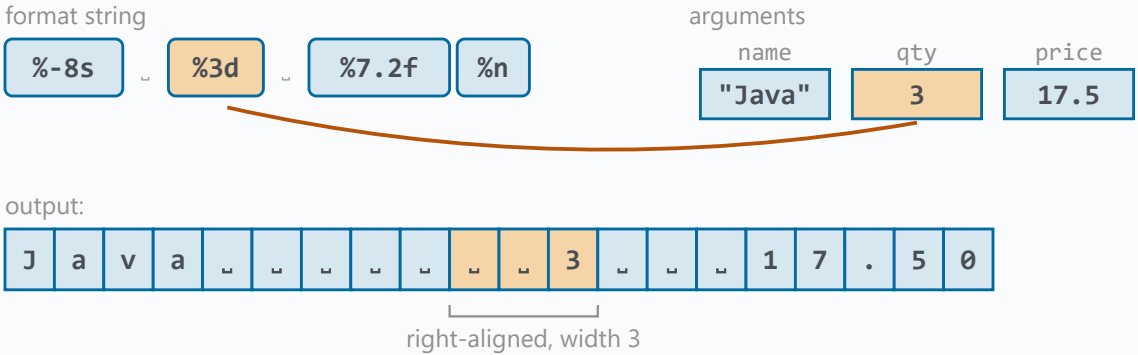
`%-8s` ← `name` : "Java" is **left**-aligned in 8 cells; the other 4 are spaces.

Step 1 / 4

%s text, %d integer, %f floating point. On a computer with a Turkish locale, `%.2f` may print a comma as the decimal separator (17,50).

Formatting with printf

```
String name = "Java"; int qty = 3; double price = 17.5;
System.out.printf("%-8s %3d %7.2f%n", name, qty, price);
```



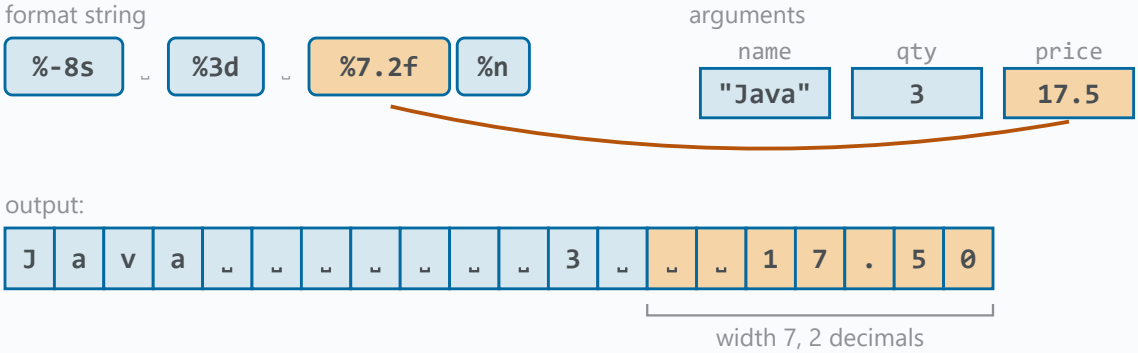
`%3d` ← `qty` : the integer 3 is **right**-aligned in 3 cells.

Step 2 / 4

%s text, %d integer, %f floating point. On a computer with a Turkish locale, `%.2f` may print a comma as the decimal separator (17,50).

Formatting with printf

```
String name = "Java"; int qty = 3; double price = 17.5;
System.out.printf("%-8s %3d %7.2f%n", name, qty, price);
```



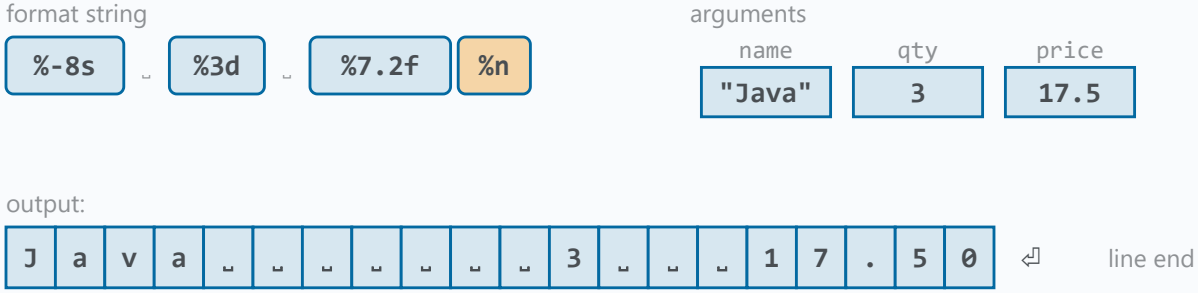
`%7.2f` ← `price` : "17.50" (5 characters, 2 decimals) is right-aligned in 7 cells.

%s text, %d integer, %f floating point. On a computer with a Turkish locale, `%.2f` may print a comma as the decimal separator (17,50).

FORMATTING

Formatting with printf

```
String name = "Java"; int qty = 3; double price = 17.5;
System.out.printf("%-8s %3d %7.2f%n", name, qty, price);
```



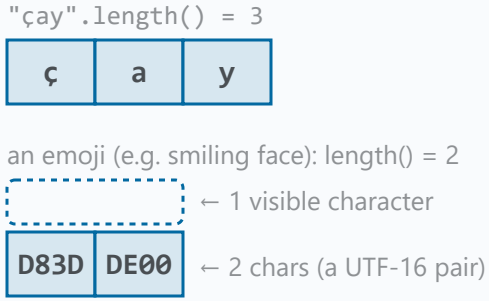
`%n` adds a line end. Format specifiers match the arguments **in order**; a type mismatch is an error.

Step 4 / 4

%s text, %d integer, %f floating point. On a computer with a Turkish locale, `%.2f` may print a comma as the decimal separator (17,50).

Unicode and Text

- A String is made of UTF-16 **code units** (char); Turkish letters fit in a single char.
- Some visible characters (emoji, some symbols) take two chars or several Unicode code points.
- So `length()` is not always the number of characters the user sees.



SUMMARY

Chapter Summary

Immutability

A String object is immutable; `toUpperCase`, `+` and `substring` return new objects.

Index

0 … length() − 1; `substring(a, b)`: a included, b excluded, length b − a.

Comparison

`equals` for contents, `compareTo` for order; `==` compares only references.

Input / output

`next` reads a word, `nextLine` a line; StringBuilder for heavy concatenation, printf for formatting.

SUMMARY

Review Questions

1. Why is String called immutable?
2. How does substring use its start and end bounds?
3. What is the difference between `==` and `equals`?
4. How do next and nextLine differ?
5. Why might StringBuilder be preferred inside a loop?

▶ **Answers**

SUMMARY

Exercises

1. Count the uppercase letters, lowercase letters, digits and spaces in a sentence separately.
2. Check whether two words are anagrams (e.g. "elma" – "alem") using character frequencies.
3. Print a full name entered by the user as "Surname, Name" with the surname in uppercase.
4. Find the longest word in a sentence (with `split`).
5. Extend the Caesar cipher to support uppercase letters (A–Z); leave other characters as they are.
6. Write code with StringBuilder that removes all spaces from a string without using `replace`.

References

- Deitel and Deitel, Java: How to Program, Early Objects, 11th ed., Pearson, 2018.
- Liang, Introduction to Java Programming and Data Structures, 13th ed., Pearson, 2024.
- Oracle, The Java Tutorials: **Strings**, **The StringBuilder Class**.
- Oracle, Java SE 21 API: **String**, **StringBuilder**.
- Sedgewick and Wayne, Computer Science: An Interdisciplinary Approach, Addison-Wesley, 2016: **3.1 Using Data Types**.
- Unicode Consortium, The Unicode Standard.

The End

Chapter 11: Strings

Previous: Chapter 10: Arrays · Next: Chapter 12: Recursion