Module java.base
Package java.lang

Interface StringTemplate

`public interface` **`StringTemplate`**

`StringTemplate` is a preview API of the Java platform.
Programs can only use `StringTemplate` when preview features are enabled.
Preview features may be removed in a future release, or upgraded to permanent features of the Java platform.

`StringTemplate`PREVIEW is the run-time representation of a string template or text block template in a template expression.

In the source code of a Java program, a string template or text block template contains an interleaved succession of *fragment literals* and *embedded expressions*. The `fragments()` method returns the fragment literals, and the `values()` method returns the results of evaluating the embedded expressions. `StringTemplate`PREVIEW does not provide access to the source code of the embedded expressions themselves; it is not a compile-time representation of a string template or text block template.

`StringTemplate`PREVIEW is primarily used in conjunction with a template processor to produce a string or other meaningful value. Evaluation of a template expression first produces an instance of `StringTemplate`PREVIEW, representing the right hand side of the template expression, and then passes the instance to the template processor given by the template expression.

For example, the following code contains a template expression that uses the template processor `RAW`, which simply yields the `StringTemplate`PREVIEW passed to it:

```
int x = 10;
int y = 20;
StringTemplate st = RAW."\{x} + \{y} = \{x + y}";
List<String> fragments = st.fragments();
List<Object> values = st.values();
```

`fragments` will be equivalent to `List.of("", " + ", " = ", "")`, which includes the empty first and last fragments. `values` will be the equivalent of `List.of(10, 20, 30)`.

The following code contains a template expression with the same template but with a different template processor, STR:

```
int x = 10;
int y = 20;
String s = STR."\{x} + \{y} = \{x + y}";
```

When the template expression is evaluated, an instance of StringTemplatePREVIEW is produced that returns the same lists from fragments() and values() as shown above. The STR template processor uses these lists to yield an interpolated string. The value of s will be equivalent to "10 + 20 = 30".

The interpolate() method provides a direct way to perform string interpolation of a StringTemplatePREVIEW. Template processors can use the following code pattern:

```
List<String> fragments = st.fragments();
List<Object> values    = st.values();
... check or manipulate the fragments and/or values ...
String result = StringTemplate.interpolate(fragments, values);
```

The process(Processor) method, in conjunction with the RAW processor, may be used to defer processing of a StringTemplatePREVIEW.

```
StringTemplate st = RAW."\{x} + \{y} = \{x + y}";
...other steps...
String result = st.process(STR);
```

The factory methods of(String) and of(List, List) can be used to construct a StringTemplatePREVIEW.

Implementation Note:

Implementations of StringTemplatePREVIEW must minimally implement the methods fragments() and values(). Instances of StringTemplatePREVIEW are considered immutable. To preserve the semantics of string templates and text block templates, the list returned by fragments() must be one element larger than the list returned by values().

See *Java Language Specification*:

15.8.6 Process Template Expressions

Since:

21

See Also:

StringTemplate.Processor PREVIEW,
FormatProcessor PREVIEW

Nested Class Summary

Nested Classes

Modifier and Type	Interface	Description
static interface	StringTemplate.Processor PREVIEW<R,E extends Throwable>	**Preview.** This interface describes the methods provided by a generalized string template processor.

Field Summary

Fields

Modifier and Type	Field	Description
static final StringTemplate.Processor PREVIEW <StringTemplate PREVIEW,RuntimeException>	RAW	This StringTemplate.Processor PREVIEW instance is conventionally used to indicate that the processing of the StringTemplate PREVIEW is to be deferred to a later time.
static final StringTemplate.Processor PREVIEW <String,RuntimeException>	STR	This StringTemplate.Processor PREVIEW instance is conventionally used for the string interpolation of a supplied StringTemplate PREVIEW.

Method Summary

All Methods Static Methods Instance Methods Abstract Methods Default Methods

Modifier and Type	Method	Description

static StringTemplate^PREVIEW	combine (StringTemplate^PREVIEW... stringTemplates)	Combine zero or more StringTemplates^PREVIEW into a single StringTemplate^PREVIEW.
static StringTemplate^PREVIEW	combine (List<StringTemplate^PREVIEW> stringTemplates	Combine a list of StringTemplates^PREVIEW into a single StringTemplate^PREVIEW.
List<String>	fragments()	Returns a list of fragment literals for this StringTemplate^PREVIEW.
default String	interpolate()	Returns the string interpolation of the fragments and values for this StringTemplate^PREVIEW.
static String	interpolate(List<String> fragments, List<?> values)	Creates a string that interleaves the elements of values between the elements of fragments.
static StringTemplate^PREVIEW	of(String string)	Returns a StringTemplate^PREVIEW as if constructed by invoking `StringTemplate.of(List.of(string), List.of())`.
static StringTemplate^PREVIEW	of(List<String> fragments, List<?> values)	Returns a StringTemplate with the given fragments and values.
default <R,E extends Throwable> R	process(StringTemplate.Processor^PREVIEW<? extends R,? extends E> processor)	Returns the result of applying the specified processor to this StringTemplate^PREVIEW.
static String	toString (StringTemplate^PREVIEW stringTemplate)	Produces a diagnostic string that describes the fragments and values of the supplied StringTemplate^PREVIEW.
List<Object>	values()	Returns a list of embedded expression results for this StringTemplate^PREVIEW.

Field Details

STR

`static final StringTemplate.Processor`PREVIEW`<String,RuntimeException> STR`

This `StringTemplate.Processor`PREVIEW instance is conventionally used for the string interpolation of a supplied `StringTemplate`PREVIEW.

For better visibility and when practical, it is recommended that users use the `STR` processor instead of invoking the `interpolate()` method. Example:

```
int x = 10;
int y = 20;
String result = STR."\{x} + \{y} = \{x + y}";
```

In the above example, the value of `result` will be `"10 + 20 = 30"`. This is produced by the interleaving concatenation of fragments and values from the supplied `StringTemplate`PREVIEW. To accommodate concatenation, values are converted to strings as if invoking `String.valueOf(Object)`.

API Note:

`STR` is statically imported implicitly into every Java compilation unit.

RAW

`static final StringTemplate.Processor`PREVIEW`<StringTemplate`PREVIEW`,RuntimeException> RAW`

This `StringTemplate.Processor`PREVIEW instance is conventionally used to indicate that the processing of the `StringTemplate`PREVIEW is to be deferred to a later time. Deferred processing can be resumed by invoking the `process(Processor)` or `StringTemplate.Processor.process(StringTemplate)`PREVIEW methods.

```
import static java.lang.StringTemplate.RAW;
...
StringTemplate st = RAW."\{x} + \{y} = \{x + y}";
...other steps...
String result = STR.process(st);
```

Implementation Note:

Unlike STR, RAW must be statically imported explicitly.

Method Details

fragments

```
List<String> fragments()
```

Returns a list of fragment literals for this `StringTemplate`PREVIEW. The fragment literals are the character sequences preceding each of the embedded expressions in source code, plus the character sequence following the last embedded expression. Such character sequences may be zero-length if an embedded expression appears at the beginning or end of a template, or if two embedded expressions are directly adjacent in a template. In the example:

```
String student = "Mary";
String teacher = "Johnson";
StringTemplate st = RAW."The student \{student} is in \{teacher}'s classroom.";
List<String> fragments = st.fragments();
```

`fragments` will be equivalent to `List.of("The student ", " is in ", "'s classroom.")`

Implementation Requirements:

the list returned is immutable

Returns:

list of string fragments

values

```
List<Object> values()
```

Returns a list of embedded expression results for this `StringTemplate`PREVIEW. In the example:

```
String student = "Mary";
String teacher = "Johnson";
StringTemplate st = RAW."The student \{student} is in \{teacher}'s classroom.";
List<Object> values = st.values();
```

`values` will be equivalent to `List.of(student, teacher)`

Implementation Requirements:

the list returned is immutable

Returns:

list of expression values

interpolate

`default String interpolate()`

Returns the string interpolation of the fragments and values for this `StringTemplate`PREVIEW.

API Note:

For better visibility and when practical, it is recommended to use the STR processor instead of invoking the `interpolate()` method.

```
String student = "Mary";
String teacher = "Johnson";
StringTemplate st = RAW."The student \{student} is in \{teacher}'s classroom.";
String result = st.interpolate();
```

In the above example, the value of `result` will be `"The student Mary is in Johnson's classroom."`. This is produced by the interleaving concatenation of fragments and values from the supplied `StringTemplate`PREVIEW. To accommodate concatenation, values are converted to strings as if invoking `String.valueOf(Object)`.

Implementation Requirements:

The default implementation returns the result of invoking `StringTemplate.interpolate(this.fragments(), this.values())`.

Returns:

interpolation of this `StringTemplate`PREVIEW

process

```
default <R,E extends Throwable> R process(StringTemplate.ProcessorPREVIEW<? extends R,? extends E> processor)
                                   throws E
```

Returns the result of applying the specified processor to this `StringTemplate`PREVIEW. This method can be used as an alternative to string template expressions. For example,

```
String student = "Mary";
String teacher = "Johnson";
String result1 = STR."The student \{student} is in \{teacher}'s classroom.";
String result2 = RAW."The student \{student} is in \{teacher}'s classroom.".process(STR);
```

Produces an equivalent result for both `result1` and `result2`.

Implementation Requirements:

The default implementation returns the result of invoking `processor.process(this)`. If the invocation throws an exception that exception is forwarded to the caller.

Type Parameters:

`R` - Processor's process result type.

`E` - Exception thrown type.

Parameters:

`processor` - the `StringTemplate.Processor`PREVIEW instance to process

Returns:

constructed object of type `R`

Throws:

`E` - exception thrown by the template processor when validation fails

`NullPointerException` - if processor is null

toString

```
static String toString(StringTemplatePREVIEW stringTemplate)
```

Produces a diagnostic string that describes the fragments and values of the supplied `StringTemplate`PREVIEW.

Parameters:

`stringTemplate` - the `StringTemplate`PREVIEW to represent

Returns:

diagnostic string representing the supplied string template

Throws:

`NullPointerException` - if stringTemplate is null

of

```
static StringTemplatePREVIEW of(String string)
```

Returns a `StringTemplate`PREVIEW as if constructed by invoking `StringTemplate.of(List.of(string), List.of())`. That is, a `StringTemplate`PREVIEW with one fragment and no values.

Parameters:

`string` - single string fragment

Returns:

StringTemplate composed from string

Throws:

`NullPointerException` - if string is null

of

```
static StringTemplatePREVIEW of(List<String> fragments,
                                List<?> values)
```

Returns a StringTemplate with the given fragments and values.

Implementation Requirements:

The `fragments` list size must be one more that the `values` list size.

Implementation Note:

Contents of both lists are copied to construct immutable lists.

Parameters:

`fragments` - list of string fragments

`values` - list of expression values

Returns:

StringTemplate composed from string

Throws:

`IllegalArgumentException` - if fragments list size is not one more than values list size

`NullPointerException` - if fragments is null or values is null or if any fragment is null.

interpolate

```
static String interpolate(List<String> fragments,
                          List<?> values)
```

Creates a string that interleaves the elements of values between the elements of fragments. To accommodate interpolation, values are converted to strings as if invoking `String.valueOf(Object)`.

Parameters:

`fragments` - list of String fragments

`values` - list of expression values

Returns:

String interpolation of fragments and values

Throws:

`IllegalArgumentException` - if fragments list size is not one more than values list size

NullPointerException - fragments or values is null or if any of the fragments is null

combine

```
static StringTemplatePREVIEW combine(StringTemplatePREVIEW... stringTemplates)
```

Combine zero or more StringTemplatesPREVIEW into a single StringTemplatePREVIEW.

```
StringTemplate st = StringTemplate.combine(RAW."\{a}", RAW."\{b}", RAW."\{c}");
assert st.interpolate().equals(STR."\{a}\{b}\{c}");
```

Fragment lists from the StringTemplatesPREVIEW are combined end to end with the last fragment from each StringTemplatePREVIEW concatenated with the first fragment of the next. To demonstrate, if we were to take two strings and we combined them as follows:

```
String s1 = "abc";
String s2 = "xyz";
String sc = s1 + s2;
assert Objects.equals(sc, "abcxyz");
```

the last character "c" from the first string is juxtaposed with the first character "x" of the second string. The same would be true of combining StringTemplatesPREVIEW.

```
StringTemplate st1 = RAW."a\{}b\{}c";
StringTemplate st2 = RAW."x\{}y\{}z";
StringTemplate st3 = RAW."a\{}b\{}cx\{}y\{}z";
StringTemplate stc = StringTemplate.combine(st1, st2);

assert Objects.equals(st1.fragments(), List.of("a", "b", "c"));
assert Objects.equals(st2.fragments(), List.of("x", "y", "z"));
assert Objects.equals(st3.fragments(), List.of("a", "b", "cx", "y", "z"));
assert Objects.equals(stc.fragments(), List.of("a", "b", "cx", "y", "z"));
```

Values lists are simply concatenated to produce a single values list. The result is a well-formed StringTemplatePREVIEW with n+1 fragments and n values, where n is the total of number of values across all the supplied StringTemplatesPREVIEW.

Implementation Note:

If zero `StringTemplate`PREVIEW arguments are provided then a `StringTemplate`PREVIEW with an empty fragment and no values is returned, as if invoking `StringTemplate.of("")` . If only one `StringTemplate`PREVIEW argument is provided then it is returned unchanged.

Parameters:

`stringTemplates` - zero or more `StringTemplate`PREVIEW

Returns:

combined `StringTemplate`PREVIEW

Throws:

`NullPointerException` - if stringTemplates is null or if any of the `stringTemplates` are null

combine

`static StringTemplate`PREVIEW `combine(List<StringTemplate`PREVIEW`> stringTemplates)`

Combine a list of `StringTemplates`PREVIEW into a single `StringTemplate`PREVIEW.

```
StringTemplate st = StringTemplate.combine(List.of(RAW."\{a}", RAW."\{b}", RAW."\{c}"));
assert st.interpolate().equals(STR."\{a}\{b}\{c}");
```

Fragment lists from the `StringTemplates`PREVIEW are combined end to end with the last fragment from each `StringTemplate`PREVIEW concatenated with the first fragment of the next. To demonstrate, if we were to take two strings and we combined them as follows:

```
String s1 = "abc";
String s2 = "xyz";
String sc = s1 + s2;
assert Objects.equals(sc, "abcxyz");
```

the last character `"c"` from the first string is juxtaposed with the first character `"x"` of the second string. The same would be true of combining `StringTemplates`PREVIEW.

```
StringTemplate st1 = RAW."a\{}b\{}c";
StringTemplate st2 = RAW."x\{}y\{}z";
StringTemplate st3 = RAW."a\{}b\{}cx\{}y\{}z";
StringTemplate stc = StringTemplate.combine(List.of(st1, st2));

assert Objects.equals(st1.fragments(), List.of("a", "b", "c"));
assert Objects.equals(st2.fragments(), List.of("x", "y", "z"));
assert Objects.equals(st3.fragments(), List.of("a", "b", "cx", "y", "z"));
assert Objects.equals(stc.fragments(), List.of("a", "b", "cx", "y", "z"));
```

Values lists are simply concatenated to produce a single values list. The result is a well-formed `StringTemplate`PREVIEW with n+1 fragments and n values, where n is the total of number of values across all the supplied `StringTemplates`PREVIEW.

Implementation Note:

If `stringTemplates.size() == 0` then a `StringTemplate`PREVIEW with an empty fragment and no values is returned, as if invoking `StringTemplate.of("")`. If `stringTemplates.size() == 1` then the first element of the list is returned unchanged.

Parameters:

`stringTemplates` - list of `StringTemplate`PREVIEW

Returns:

combined `StringTemplate`PREVIEW

Throws:

`NullPointerException` - if stringTemplates is null or if any of the its elements are null