Module java.base
Package java.util

Class FormatProcessor

java.lang.Object
    java.util.FormatProcessor

All Implemented Interfaces:

StringTemplate.Processor[PREVIEW]<String,RuntimeException>, StringTemplate.Processor.Linkage[PREVIEW]

```
public final class FormatProcessor
extends Object
implements StringTemplate.Processor[PREVIEW]<String,RuntimeException>, StringTemplate.Processor.Linkage[PREVIEW]
```

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

This StringTemplate.Processor[PREVIEW] constructs a String result using Formatter specifications and values found in the StringTemplate[PREVIEW]. Unlike Formatter, FormatProcessor[PREVIEW] uses the value from the embedded expression that immediately follows, without whitespace, the format specifier. For example:

```
FormatProcessor fmt = FormatProcessor.create(Locale.ROOT);
int x = 10;
int y = 20;
String result = fmt."%05d\{x} + %05d\{y} = %05d\{x + y}";
```

In the above example, the value of `result` will be `"00010 + 00020 = 00030"`.

Embedded expressions without a preceeding format specifier, use `%s` by default.

```
FormatProcessor fmt = FormatProcessor.create(Locale.ROOT);
int x = 10;
```

```
int y = 20;
String result1 = fmt."\{x} + \{y} = \{x + y}";
String result2 = fmt."%s\{x} + %s\{y} = %s\{x + y}";
```

In the above example, the value of `result1` and `result2` will both be `"10 + 20 = 30"`.

The `FormatProcessor`[PREVIEW] format specification used and exceptions thrown are the same as those of `Formatter`.

However, there are two significant differences related to the position of arguments. An explict `n$` and relative `<` index will cause an exception due to a missing argument list. Whitespace appearing between the specification and the embedded expression will also cause an exception.

`FormatProcessor`[PREVIEW] allows the use of different locales. For example:

```
Locale locale = Locale.forLanguageTag("th-TH-u-nu-thai");
FormatProcessor thaiFMT = FormatProcessor.create(locale);
int x = 10;
int y = 20;
String result = thaiFMT."%4d\{x} + %4d\{y} = %5d\{x + y}";
```

In the above example, the value of `result` will be `" ๑๐ + ๒๐ = ๓๐"`.

For day to day use, the predefined `FMT` `FormatProcessor`[PREVIEW] is available. `FMT` is defined using the `Locale.ROOT`. Example:

```
int x = 10;
int y = 20;
String result = FMT."0x%04x\{x} + 0x%04x\{y} = 0x%04x\{x + y}";
```

In the above example, the value of `result` will be `"0x000a + 0x0014 = 0x001E"`.

Since:

21

See Also:

`StringTemplate.Processor`[PREVIEW]

Nested Class Summary

Nested classes/interfaces declared in interface java.lang.StringTemplate.Processor[PREVIEW]

StringTemplate.Processor.Linkage[PREVIEW]

Field Summary

Fields

Modifier and Type	Field	Description
static final FormatProcessor[PREVIEW]	FMT	This predefined FormatProcessor[PREVIEW] instance constructs a String result using the Locale.ROOT Locale.

Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
static FormatProcessor[PREVIEW]	create(Locale locale)	Create a new FormatProcessor[PREVIEW] using the specified locale.
MethodHandle	linkage(List<String> fragments, MethodType type)	Constructs a MethodHandle that when supplied with the values from a StringTemplate[PREVIEW] will produce a result equivalent to that provided by process(StringTemplate).
final String	process(StringTemplate[PREVIEW] stringTemplate)	Constructs a String based on the fragments, format specifications found in the fragments and values in the supplied StringTemplate[PREVIEW] object.

Methods declared in class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Details

FMT

```
public static final FormatProcessor PREVIEW FMT
```

This predefined FormatProcessor PREVIEW instance constructs a String result using the Locale.ROOT Locale. See FormatProcessor PREVIEW for more details. Example:

```
int x = 10;
int y = 20;
String result = FMT."0x%04x\{x} + 0x%04x\{y} = 0x%04x\{x + y}";
```

In the above example, the value of result will be "0x000a + 0x0014 = 0x001E".

See Also:

FormatProcessor PREVIEW

Method Details

create

```
public static FormatProcessor PREVIEW create(Locale locale)
```

Create a new FormatProcessor PREVIEW using the specified locale.

Parameters:

`locale` - `Locale` used to format

Returns:

a new instance of `FormatProcessor`[PREVIEW]

Throws:

`NullPointerException` - if locale is null

process

```
public final String process(StringTemplate[PREVIEW] stringTemplate)
```

Constructs a `String` based on the fragments, format specifications found in the fragments and values in the supplied `StringTemplate`[PREVIEW] object. This method constructs a format string from the fragments, gathers up the values and evaluates the expression asif evaulating `new Formatter(locale).format(format, values).toString()`.

If an embedded expression is not immediately preceded by a specifier then a `%s` is inserted in the format.

Specified by:

`process` in interface `StringTemplate.Processor`[PREVIEW]<`String`,`RuntimeException`>

Parameters:

`stringTemplate` - a `StringTemplate`[PREVIEW] instance

Returns:

constructed `String`

Throws:

`IllegalFormatException` - If a format specifier contains an illegal syntax, a format specifier that is incompatible with the given arguments, a specifier not followed immediately by an embedded expression or other illegal conditions. For specification of all possible formatting errors, see the details section of the formatter class specification.

`NullPointerException` - if stringTemplate is null

See Also:

`Formatter`

linkage

```
public MethodHandle linkage(List<String> fragments,
                            MethodType type)
```

Constructs a `MethodHandle` that when supplied with the values from a `StringTemplate`[PREVIEW] will produce a result equivalent to that provided by `process(StringTemplate)`. This `MethodHandle` is used by `FMT` and the ilk to perform a more specialized composition of a result. This specialization is done by prescanning the fragments and value types of a `StringTemplate`[PREVIEW].

Process template expressions can be specialized when the processor is of type `StringTemplate.Processor.Linkage`[PREVIEW] and fetched from a static constant as is `FMT` (`static final FormatProcessor`).

Other `FormatProcessors`[PREVIEW] can be specialized when stored in a static final. For example:

```
FormatProcessor THAI_FMT = FormatProcessor.create(Locale.forLanguageTag("th-TH-u-nu-thai"));
```

`THAI_FMT` will now produce specialized `MethodHandles` by way of `linkage(List, MethodType)`. See `process(StringTemplate)` for more information.

Specified by:

`linkage` in interface `StringTemplate.Processor.Linkage`[PREVIEW]

Parameters:

`fragments` - string template fragments

`type` - method type, includes the StringTemplate receiver as well as the value types

Returns:

`MethodHandle` for the processor applied to template

Throws:

`IllegalFormatException` - If a format specifier contains an illegal syntax, a format specifier that is incompatible with the given arguments, a specifier not followed immediately by an embedded expression or other illegal conditions. For specification of all possible formatting errors, see the details section of the formatter class specification.

`NullPointerException` - if fragments or type is null

See Also:

```
Formatter
```
