Module java.base
Package java.util.function

Interface BiConsumer<T,U>

Type Parameters:

`T` - the type of the first argument to the operation

`U` - the type of the second argument to the operation

Functional Interface:

This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

```
@FunctionalInterface
public interface BiConsumer<T,U>
```

Represents an operation that accepts two input arguments and returns no result. This is the two-arity specialization of `Consumer`. Unlike most other functional interfaces, `BiConsumer` is expected to operate via side-effects.

This is a functional interface whose functional method is `accept(Object, Object)`.

Since:

1.8

See Also:

`Consumer`

Method Summary

All Methods | Instance Methods | Abstract Methods | Default Methods

Modifier and Type	Method	Description
`void`	`accept(T t, U u)`	Performs this operation on the given arguments.
`default BiConsumer<T,U>`	`andThen(BiConsumer<? super T,? super U> after)`	Returns a composed `BiConsumer` that performs, in sequence, this operation followed by the `after` operation.

Method Details

accept

```
void accept(T t,
            U u)
```

Performs this operation on the given arguments.

Parameters:

`t` - the first input argument

`u` - the second input argument

andThen

```
default BiConsumer<T,U> andThen(BiConsumer<? super T,? super U> after)
```

Returns a composed `BiConsumer` that performs, in sequence, this operation followed by the `after` operation. If performing either operation throws an exception, it is relayed to the caller of the composed operation. If performing this operation throws an exception, the `after` operation will not be performed.

Parameters:

`after` - the operation to perform after this operation

Returns:

a composed `BiConsumer` that performs in sequence this operation followed by the `after` operation

Throws:

`NullPointerException` - if `after` is null