

Lambda Expressions and Method References



Jesper de Jong

Software Architect

@jesperdj | www.jesperdj.com



Syntax of Lambda Expressions

public

Access specifier

Syntax of Lambda Expressions

```
public int
```

Access specifier

Return type



Syntax of Lambda Expressions

```
public int compare
```

Access specifier

Return type

Method name



Syntax of Lambda Expressions

```
public int compare(String first, String second)
```

Access specifier

Return type

Method name

Parameter list



Syntax of Lambda Expressions

```
public int compare(String first, String second) {  
    return Integer.compare(first.length(), second.length());  
}
```

Access specifier

Return type

Method name

Parameter list

Body



Syntax of Lambda Expressions

```
public int compare(String first, String second) {  
    return Integer.compare(first.length(), second.length());  
}
```

Access specifier	Return type	Method name	Parameter list	Body
------------------	-------------	-------------	----------------	------

(String first, String second)				
-------------------------------	--	--	--	--

Parameter list



Syntax of Lambda Expressions

```
public int compare(String first, String second) {  
    return Integer.compare(first.length(), second.length());  
}
```

Access specifier	Return type	Method name	Parameter list	Body
------------------	-------------	-------------	----------------	------

```
(String first, String second) ->
```

Parameter list

Arrow



Syntax of Lambda Expressions

```
public int compare(String first, String second) {  
    return Integer.compare(first.length(), second.length());  
}
```

Access specifier	Return type	Method name	Parameter list	Body
------------------	-------------	-------------	----------------	------

```
(String first, String second) -> {  
    return Integer.compare(first.length(), second.length());  
}
```

Parameter list	Arrow	Body
----------------	-------	------



Syntax of Lambda Expressions

```
public int compare(String first, String second) {  
    return Integer.compare(first.length(), second.length());  
}
```

Access specifier	Return type	Method name	Parameter list	Body
------------------	-------------	-------------	----------------	------

```
(first, second) -> {  
    return Integer.compare(first.length(), second.length());  
}
```

Parameter list	Arrow	Body
----------------	-------	------



Syntax of Lambda Expressions

```
public int compare(String first, String second) {  
    return Integer.compare(first.length(), second.length());  
}
```

Access specifier	Return type	Method name	Parameter list	Body
------------------	-------------	-------------	----------------	------

```
(var first, var second) -> {  
    return Integer.compare(first.length(), second.length());  
}
```

Parameter list	Arrow	Body
----------------	-------	------



Syntax of Lambda Expressions

```
(() -> {  
    System.out.println("Hello from Pluralsight");  
})
```

Lambda without parameters

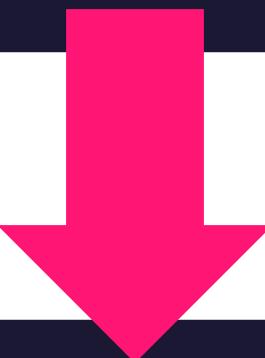
```
name -> {  
    System.out.println("Hello, " + name);  
}
```

Lambda with a single parameter

Syntax of Lambda Expressions

```
(first, second) -> {  
    return Integer.compare(first.length(), second.length());  
}
```

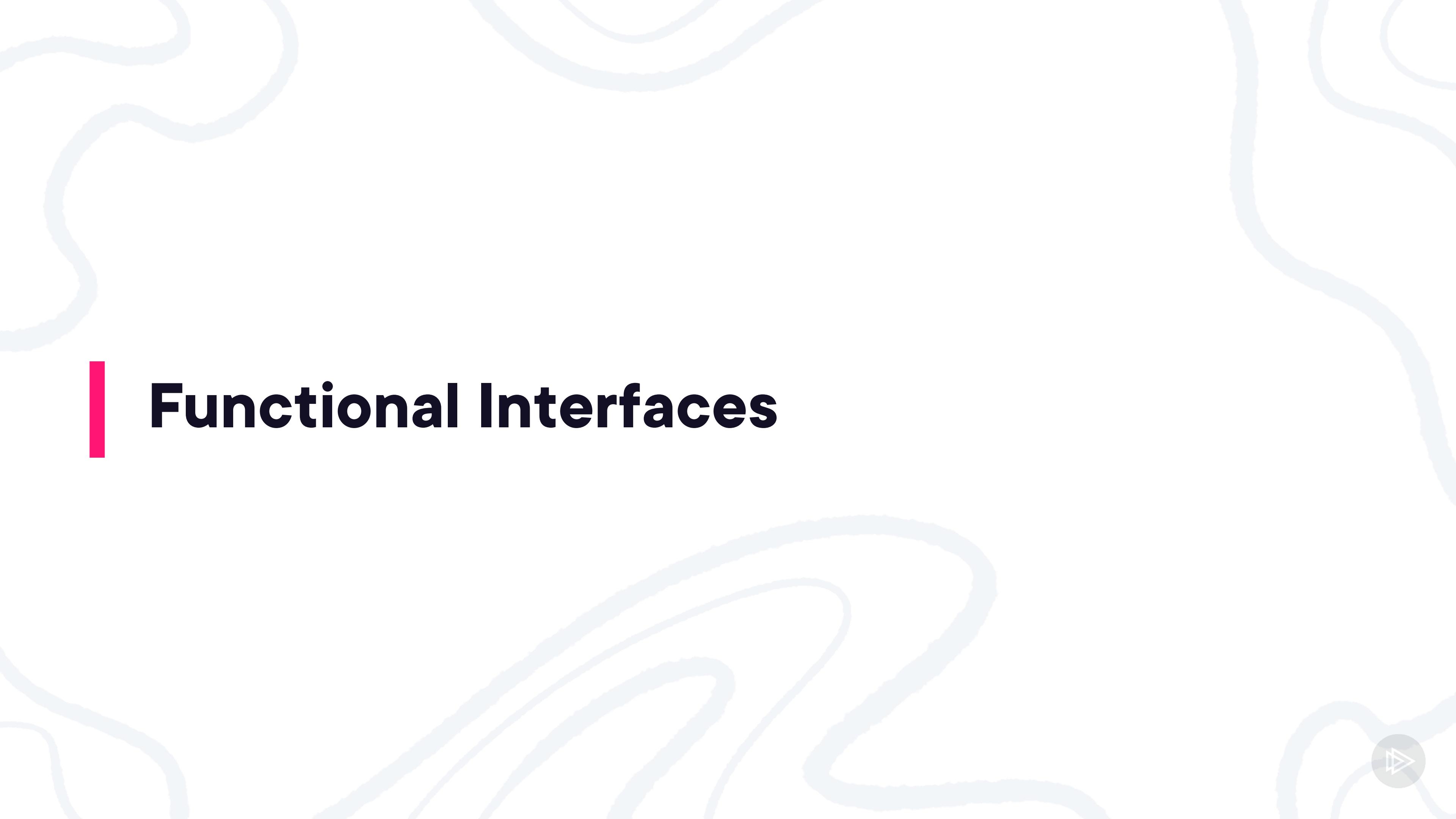
Body as a block



```
(first, second) -> Integer.compare(first.length(), second.length());
```

Body as a single expression





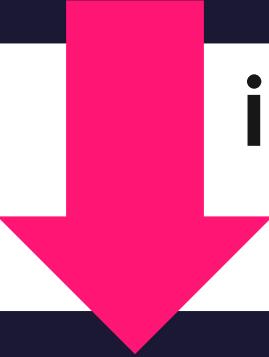
Functional Interfaces



Lambda Expressions and Functional Interfaces

A lambda expression **implements**
a functional interface

```
(first, second) -> Integer.compare(first.length(), second.length());
```



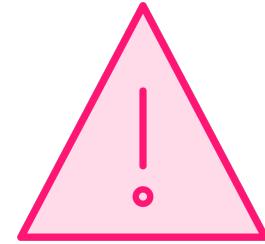
implements

```
public interface Comparator<T> {  
    int compare(T o1, T o2);  
}
```



Lambda Expressions and Functional Interfaces

You cannot assign a lambda expression
to a variable declared with **var**



```
// Error: cannot infer type for local variable comparator
//         (lambda expression needs an explicit target-type)
var comparator = (String first, String second) ->
    Integer.compare(first.length(), second.length());
```

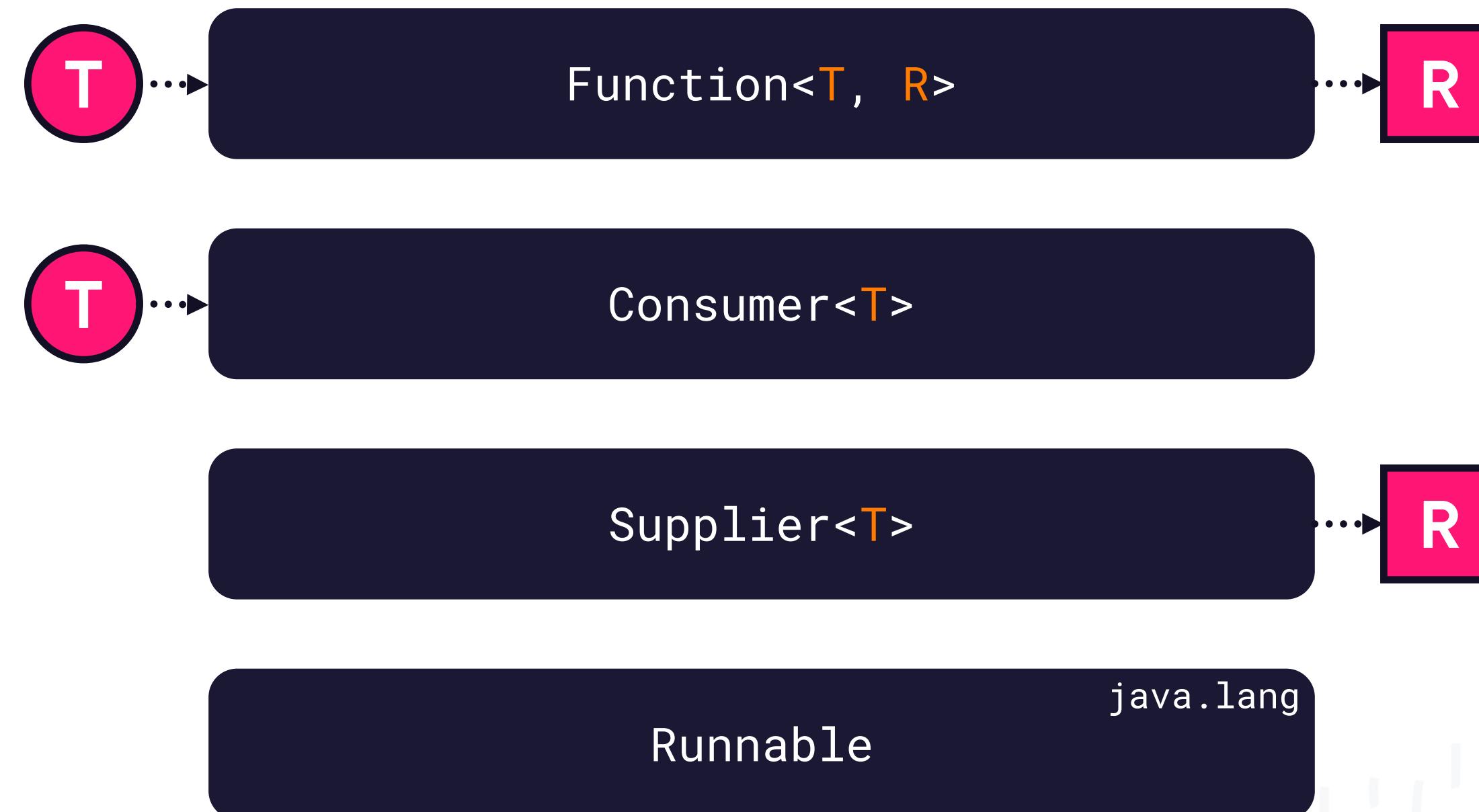


Standard Functional Interfaces



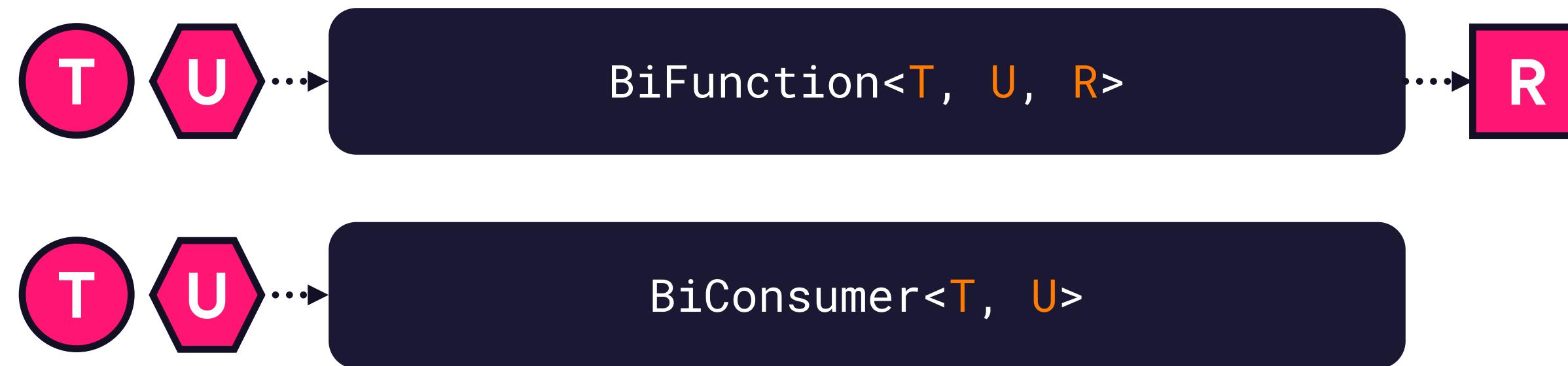
Lambda Expressions and Functional Interfaces

Package `java.util.function`



Lambda Expressions and Functional Interfaces

Package `java.util.function`



Lambda Expressions and Functional Interfaces

Package `java.util.function`



Method References



Method References

```
var names = List.of("Joe Smith", "Susan Miller", "Will Johnson");
```



Method References

```
var names = List.of("Joe Smith", "Susan Miller", "Will Johnson");

// Lambda expression
names.forEach(name -> System.out.println(name));
```



Method References

```
var names = List.of("Joe Smith", "Susan Miller", "Will Johnson");

// Lambda expression
names.forEach(name -> System.out.println(name));

// Method reference
names.forEach(System.out::println);
```



Method References

A method reference **implements**
a functional interface

Lambda expression

Defines an anonymous method
on the spot

Method reference

Points to an existing method



Syntax of Method References

```
TypeName :: staticMethodName
```

Reference to a static method

```
objectRef :: instanceMethodName
```

**Reference to an instance method
of a specific object**

```
TypeName :: instanceMethodName
```

**Reference to an instance method
context determines the object**

```
TypeName :: new
```

Reference to a constructor



Summary



Lambda expressions

- Anonymous method
- Syntax of lambda expressions

Functional interfaces

- Single abstract method
- `@FunctionalInterface` annotation

Standard functional interfaces

- Package `java.util.function`



Summary



Capturing local variables

- Effectively final

Principles of functional programming

- No side effects

Working with checked exceptions

Method references

- Four kinds of method references





More Information

Working with Streams and Lambda Expressions in Java

Jesper de Jong



Up Next:

Annotations

