

The Java BiFunction Functional Interface

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt



Professor of Computer Science

**Institute for Software
Integrated Systems**

**Vanderbilt University
Nashville, Tennessee, USA**



Learning Objectives in this Part of the Lesson

- Understand the BiFunction functional interface in Java & recognize how it can be used in conjunction with lambda expressions & method references

Interface BiFunction<T,U,R>

Type Parameters:

T - the type of the first argument to the function

U - the type of the second argument to the function

R - the type of the result of the function

All Known Subinterfaces:

BinaryOperator<T>

Functional Interface:

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

Learning Objectives in this Part of the Lesson

- Understand the BiFunction functional interface in Java & recognize how it can be used in conjunction with lambda expressions & method references
- Know how to apply Java BiFunction in a concise example

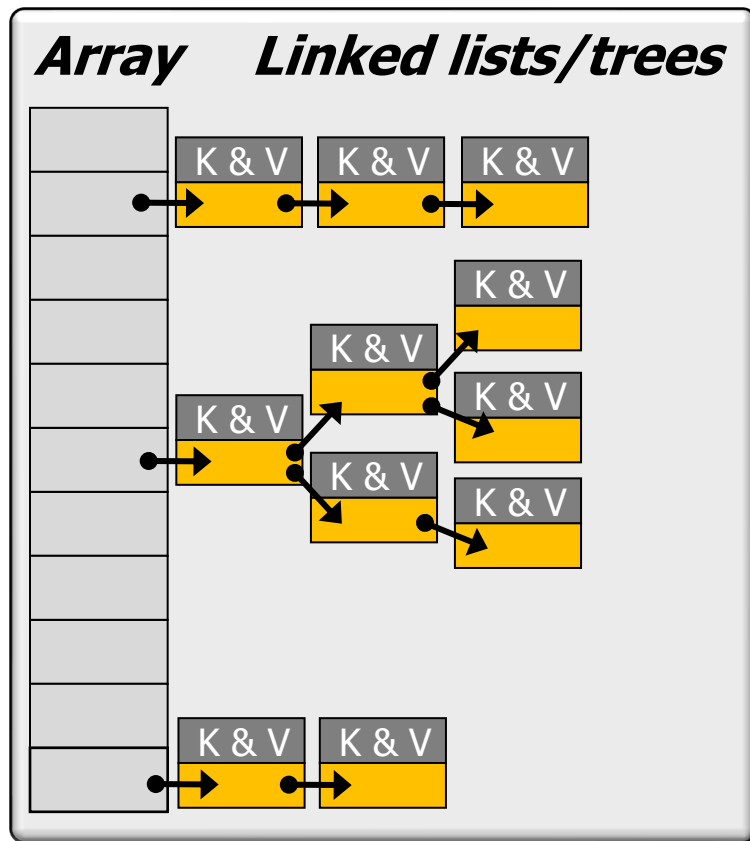


See github.com/douglasraigschmidt/ModernJava/tree/main/FP/ex11

Learning Objectives in this Part of the Lesson

- Understand the BiFunction functional interface in Java & recognize how it can be used in conjunction with lambda expressions & method references
- Know how to apply Java BiFunction in a concise example
 - This example showcases the Java collections framework's ConcurrentHashMap class & Set interface

ConcurrentHashMap



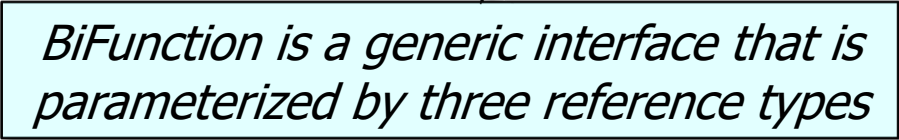
Overview of the BiFunction Functional Interface

Overview of the BiFunction Functional Interface

- A *BiFunction* applies a computation on 2 params & returns a result
 - `public interface BiFunction<T, U, R> { R apply(T t, U u); }`

Overview of the BiFunction Functional Interface

- A *BiFunction* applies a computation on 2 params & returns a result
- `public interface BiFunction<T, U, R> { R apply(T t, U u); }`



BiFunction is a generic interface that is parameterized by three reference types

Overview of the BiFunction Functional Interface

- A *BiFunction* applies a computation on 2 params & returns a result
- `public interface BiFunction<T, U, R> { R apply(T t, U u); }`

Its abstract method is passed two parameters of type T & U & returns a value of type R

Applying the BiFunction Functional Interface

Applying the BiFunction Functional Interface

- This example shows how a modern Java BiFunction lambda can be used to replace all the values of all keys in a ConcurrentHashMap

```
Map<String, Integer> stooges =  
    new ConcurrentHashMap<String, Integer>() {  
        { put("Larry", 100); put("Curly", 90); put("Moe", 110); }  
    };
```

```
for (Map.Entry<String, Integer> entry : stooges.entrySet())  
    entry.setValue(entry.getValue() - 30);
```

VS.

```
stooges.replaceAll((k, v) -> v - 30);
```

Applying the BiFunction Functional Interface

- This example shows how a modern Java BiFunction lambda can be used to replace all the values of all keys in a ConcurrentHashMap

```
Map<String, Integer> stooges =  
    new ConcurrentHashMap<String, Integer>() {  
        { put("Larry", 100); put("Curly", 90); put("Moe", 110); }  
    };
```

Create a map of "stooges" & their IQs!

```
for (Map.Entry<String, Integer> entry : stooges.entrySet())  
    entry.setValue(entry.getValue() - 30);
```

VS.

```
stooges.replaceAll((k, v) -> v - 30);
```



See en.wikipedia.org/wiki/The_Three_Stooges

Applying the BiFunction Functional Interface

- This example shows how a modern Java BiFunction lambda can be used to replace all the values of all keys in a ConcurrentHashMap

```
Map<String, Integer> stooges =  
    new ConcurrentHashMap<String, Integer>() {  
        { put("Larry", 100); put("Curly", 90); put("Moe", 110); }  
    };
```

```
for (Map.Entry<String, Integer> entry : stooges.entrySet())  
    entry.setValue(entry.getValue() - 30);
```

vs.

Conventional way of subtracting 30 IQ points from each stooge in map

```
stooges.replaceAll((k, v) -> v - 30);
```

Applying the BiFunction Functional Interface

- This example shows how a modern Java BiFunction lambda can be used to replace all the values of all keys in a ConcurrentHashMap

```
Map<String, Integer> stooges =  
    new ConcurrentHashMap<String, Integer>() {  
        { put("Larry", 100); put("Curly", 90); put("Moe", 110); }  
    };
```

```
for (Map.Entry<String, Integer> entry : stooges.entrySet())  
    entry.setValue(entry.getValue() - 30);
```

vs.

BiFunction lambda subtracts 30 IQ points from each stooge in map

```
stooges.replaceAll((k, v) -> v - 30);
```

Applying the BiFunction Functional Interface

- This example shows how a modern Java BiFunction lambda can be used to replace all the values of all keys in a ConcurrentHashMap

```
Map<String, Integer> stooges =  
    new ConcurrentHashMap<String, Integer>() {  
        { put("Larry", 100); put("Curly", 90); put("Moe", 110); }  
    };
```

```
for (Map.Entry<String, Integer> entry : stooges.entrySet())  
    entry.setValue(entry.getValue() - 30);
```

vs.

Unlike Entry operations, replaceAll() operates in a thread-safe manner!

```
stooges.replaceAll((k, v) -> v - 30);
```

How ConcurrentHashMap Uses the BiFunction Interface

How ConcurrentHashMap Uses the BiFunction Interface


- The ConcurrentHashMap `replaceAll()` method uses the BiFunction passed to it in a thread-safe manner

```
class ConcurrentHashMap<K,V> {
    ...
    public void replaceAll
        (BiFunction<? super K, ? super V, ? extends V> function) {
        ...
        for (Node<K,V> p; (p = it.advance()) != null; ) {
            V oldValue = p.val;
            for (K key = p.key;;) {
                V newValue = function.apply(key, oldValue);
                ...
                replaceNode(key, newValue, oldValue)
            }
            ...
        }
    }
}
```


How ConcurrentHashMap Uses the BiFunction Interface

- The ConcurrentHashMap replaceAll() method uses the BiFunction passed to it in a thread-safe manner

```
class ConcurrentHashMap<K,V> {  
    ...  
    public void replaceAll  
        (BiFunction<? super K, ? super V, ? extends V> function) {  
        ...  
        for (Node<K,V> p; (p = it.advance()) != null; ) {  
            V oldValue = p.val;  
            for (K key = p.key;;) {  
                V newValue = function.apply(key, oldValue);  
                ...  
                replaceNode(key, newValue, oldValue)  
                ...  
            }  
        }  
    }  
}
```



The bifunction parameter is bound to the lambda expression `v - 30`

How ConcurrentHashMap Uses the BiFunction Interface

- The ConcurrentHashMap replaceAll() method uses the BiFunction passed to it in a thread-safe manner

```
class ConcurrentHashMap<K,V> {  
    ...  
    public void replaceAll  
        (BiFunction<? super K, ? super V, ? extends V> function) {  
        ...  
        for (Node<K,V> p; (p = it.advance()) != null; ) {  
            V oldValue = p.val;  
            for (K key = p.key;;) {  
                V newValue = function.apply(key, oldValue);  
                ...  
                replaceNode(key, newValue, oldValue)  
                ...  
            }  
        }  
    }  
}
```

V newValue =
oldValue - 30

The apply() method is replaced by the v - 30 bifunction lambda

End of the Java BiFunction Functional Interface