

Refactoring to Java 8 Streams and Lambdas 2.0

Dr Heinz M. Kabutz

Last updated 2021-06-30

© 2020 Heinz Kabutz – All Rights Reserved



Javaspecialists.eu
java training

Copyright Notice

- © 2020 Heinz Kabutz, All Rights Reserved
- No part of this course material may be reproduced without the express written permission of the author, including but not limited to: blogs, books, courses, public presentations.
- A license is hereby granted to use the ideas and source code in this course material for your personal and professional software development.
- Contact heinz@javaspecialists.eu if you are in any way uncertain as to your rights and obligations.

Introduction



Warm Welcome - Course Author

- **Dr Heinz Kabutz**

- Born in Cape Town, now lives on Crete
- Created The Java Specialists' Newsletter
 - www.javaspecialists.eu
- One of the first Java Champions
 - www.javachampions.org



Comfort and Learning

- We need
 - oxygen
 - short breaks every 45 minutes
 - physical exercise after class
 - Run, walk, gym, cycle, etc.

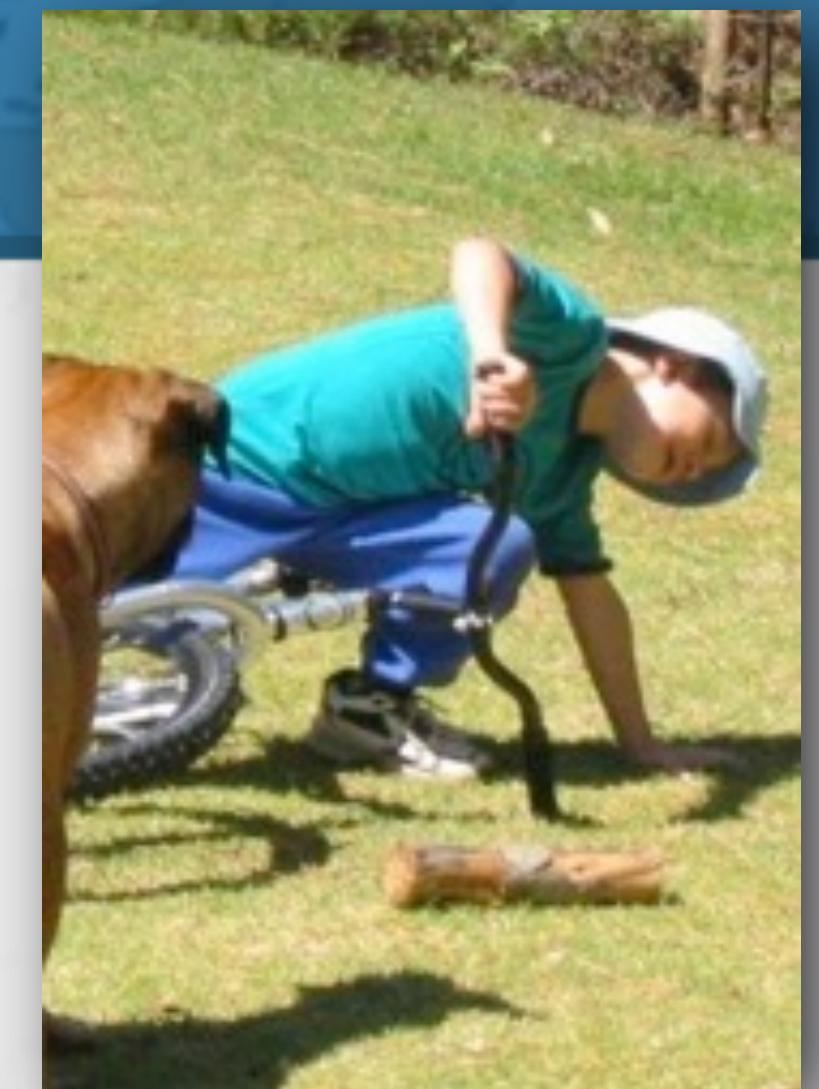


Questions

- Please please please ask questions!
 - For self-study, please leave comment in sections
- Interrupt me at any time
 - Questions that are off-topic might be delayed until later
 - If so, please write down question and we can look at it during exercise time
- There are some stupid questions
 - They are the ones we did not ask
 - Once we have asked them, they are not stupid anymore
- The more we ask, the more everyone learns

Exercises

- We learn cycling by falling
 - Listening to lectures is not enough
- Exercises help us to internalize refactoring
- Please make sure you have at least Java 8
- IntelliJ IDEA 2020.2 or later
- Get project using git
 - Will send you URL and credentials now



Refactoring

How to do it



Refactoring

- Pioneered by Martin Fowler
 - Based on research by William Opdyke
- What it is
 - Improving the design of existing code
 - Without adding new functionality
- Unit testing
 - Bad refactorings often introduce bugs
- IntelliJ
 - Analyze -> Inspect Code great, even in Community Edition

Inspecting Code with IntelliJ IDEA

Quick demo



Javaspecialists.eu
java training

Java Language Changes



Java Language Changes

- **Java 1.1**
 - Inner classes and anonymous inner classes
 - Also called anonymous types
 - No more "private protected"
- **Java 2**
 - `java.util` collections and maps
- **Java 3**
 - Not much
- **Java 4**
 - `assert` keyword added

Java Language Changes

- **Java 5**

- Generics
- **java.util.concurrent** thread-safe classes
- Enums
- Autoboxing
- Enhanced 'for' loop
- **StringBuilder**

- **Java 6**

- Not much

Java Language Changes

- **Java 7**
 - Diamond generics operator <>
 - `Objects.equals()`
 - Multi-catch in try-catch
 - try-with-resource
 - Compare for numbers
 - Switching on String

Java Language Changes

- **Java 8**
 - Default and static interface methods
 - Lambdas
 - Method references
 - Compound Map methods
 - `putIfAbsent()`, `computeIfAbsent()`, `merge()`, etc.
 - `Collection.removeIf`
 - Optional
 - Streams
 - Primitive vs Object streams
 - `allMatch()`, `map()`, `filter()`, `findFirst()`, `collect()`, etc.
 - Spliterators

Java Language Changes

- Java 9
 - Java Platform Module System (JPMS)
 - Diamond operator for anonymous types
 - List.of(), Set.of(), Map.of()
- Java 10
 - Local variable can be omitted (var)
- Java 11
 - Not much
- Java 14
 - Enhanced switch

Java Language Changes

- **Java 14 Preview**

- Records
- Pattern variables
- Text blocks

- **Java 15**

- Text blocks

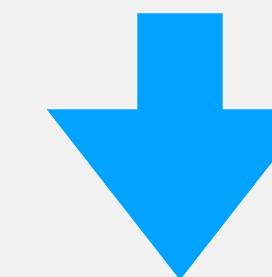
1. Default Methods in Interfaces



1. Default Methods in Interfaces

- **ConcurrentMap vs Map in Java 5**
- **List.sort() vs Collections.sort()**
 - AuthHelper.loadAuthenticators_internal()

```
Collections.sort(authenticators, new AuthenticationComparator());
```



```
authenticators.sort(new AuthenticationComparator());
```

Now it's your turn

- **Hyperlinks in Refactoring Tasks**
 - Look in `task1_defaultMethodsInInterfaces()`

`AuthHelper.loadAuthenticators_internal()`
`BillingAccountWorker.makePartyBillingAccountList()`
`ContentJsonEvents.getContentAssocs()`
`EntityUtil.orderBy()`
`OrderMapList.exec()`
`ProductDisplayWorker.productOrderByMap()`
`ShoppingCart.getLineListOrderedByBasePrice()`
`UtilMisc.sortMaps()`

Static Methods in Interfaces



Static Methods in Interfaces

- ContentJsonEvents.getContentAssocs()

```
nodes.sort(new Comparator<Map<String, Object>>() {
    public int compare(Map<String, Object> node1, Map<String, Object> node2) {
        Map<String, Object> data1 = UtilGenerics.cast(node1.get("data"));
        Map<String, Object> data2 = UtilGenerics.cast(node2.get("data"));
        if (data1 == null || data2 == null) return 0;
        String title1 = (String) data1.get("title");
        String title2 = (String) data2.get("title");
        if (title1 == null || title2 == null) return 0;
        return title1.toLowerCase(Locale.getDefault())
            .compareTo(title2.toLowerCase(Locale.getDefault())));
    }
});
```

Comparator interface static methods

- **Comparator.comparing(Function<T, U>)**
- **Comparator.nullsFirst(Comparator)**

Functional Interfaces

Predicate, Consumer, Function, Supplier



Functional Interfaces

- Interface with a single abstract method
 - e.g. Runnable
- `java.util.function` has 43 "functional interfaces"
- But actually only 4
 - Predicate (5)
 - Consumer (8)
 - Function (25)
 - Supplier (5)
- Interfaces for int, long, double and objects
 - e.g. IntPredicate, LongPredicate, DoublePredicate, Predicate

Static Methods in Interfaces

- Comparator with a key extractor Function

```
nodes.sort(  
    Comparator.comparing(new Function<Map<String, Object>, String>() {  
        public String apply(Map<String, Object> node) {  
            Map<String, Object> data = (Map<String, Object>) node.get("data");  
            if (data == null) return null;  
            String title = (String) data.get("title");  
            if (title == null) return null;  
            return title.toLowerCase(Locale.getDefault());  
        }  
    })  
);
```

Static Methods in Interfaces

- With the `nullsFirst` comparator

```
nodes.sort(  
    Comparator.nullsFirst()  
        Comparator.comparing(new Function<Map<String, Object>, String>() {  
            public String apply(Map<String, Object> node) {  
                Map<String, Object> data = (Map<String, Object>) node.get("data");  
                if (data == null) return null;  
                String title = (String) data.get("title");  
                if (title == null) return null;  
                return title.toLowerCase(Locale.getDefault());  
            }  
        })  
);
```

2. Lambdas



2. Lambdas

- Tedium to type anonymous types

```
nodes.sort(  
    Comparator.nullsFirst(  
        Comparator.comparing(new Function<Map<String, Object>, String>() {  
            public String apply(Map<String, Object> node) {  
                Map<String, Object> data = (Map<String, Object>) node.get("data");  
                if (data == null) return null;  
                String title = (String) data.get("title");  
                if (title == null) return null;  
                return title.toLowerCase(Locale.getDefault());  
            }  
        })  
    ));
```

Lambdas

- The compiler can deduce all this

```
nodes.sort(  
    Comparator.nullsFirst(  
        Comparator.comparing(new Function<Map<String, Object>, String>() {  
            public String apply(Map<String, Object> node) {  
                Map<String, Object> data = (Map<String, Object>) node.get("data");  
                if (data == null) return null;  
                String title = (String) data.get("title");  
                if (title == null) return null;  
                return title.toLowerCase(Locale.getDefault());  
            }  
        })  
    ));
```

Lambdas

- Replaced with lambda using ->

```
nodes.sort(  
    Comparator.nullsFirst(  
        Comparator.comparing((Map<String, Object> node) -> {  
            Map<String, Object> data = (Map<String, Object>) node.get("data");  
            if (data == null) return null;  
            String title = (String) data.get("title");  
            if (title == null) return null;  
            return title.toLowerCase(Locale.getDefault());  
        })  
    )  
)
```

Lambdas

- Method parameter can be omitted

```
nodes.sort(  
    Comparator.nullsFirst(  
        Comparator.comparing((Map<String, Object> node) -> {  
            Map<String, Object> data = (Map<String, Object>) node.get("data");  
            if (data == null) return null;  
            String title = (String) data.get("title");  
            if (title == null) return null;  
            return title.toLowerCase(Locale.getDefault());  
        })  
    )  
) ;
```

Lambdas

- Less clutter

```
nodes.sort(  
    Comparator.nullsFirst(  
        Comparator.comparing((node) -> {  
            Map<String, Object> data = (Map<String, Object>) node.get("data");  
            if (data == null) return null;  
            String title = (String) data.get("title");  
            if (title == null) return null;  
            return title.toLowerCase(Locale.getDefault());  
        })  
    )  
)
```

Lambdas

- Single parameter does not need brackets

```
nodes.sort(  
    Comparator.nullsFirst(  
        Comparator.comparing((node) -> {  
            Map<String, Object> data = (Map<String, Object>) node.get("data");  
            if (data == null) return null;  
            String title = (String) data.get("title");  
            if (title == null) return null;  
            return title.toLowerCase(Locale.getDefault());  
        })  
    )  
)
```

Lambdas

- That's better!

```
nodes.sort(  
    Comparator.nullsFirst(  
        Comparator.comparing(node -> {  
            Map<String, Object> data = (Map<String, Object>) node.get("data");  
            if (data == null) return null;  
            String title = (String) data.get("title");  
            if (title == null) return null;  
            return title.toLowerCase(Locale.getDefault());  
        })  
    )  
)
```

Let's extract the body as a method

- That's better!

```
nodes.sort(  
    Comparator.nullsFirst(  
        Comparator.comparing(node -> { return extractTitle(node); })  
    )  
);  
private static String extractTitle(Map<String, Object> node) {  
    Map<String, Object> data = (Map<String, Object>) node.get("data");  
    if (data == null) return null;  
    String title = (String) data.get("title");  
    if (title == null) return null;  
    return title.toLowerCase(Locale.getDefault());  
}
```

Statement vs Expression Lambda

- Can also get rid of return

```
nodes.sort(  
    Comparator.nullsFirst(  
        Comparator.comparing(node -> { return extractTitle(node); })  
    )  
);
```

Statement vs Expression Lambda

- We can breathe again

```
nodes.sort(  
    Comparator.nullsFirst(  
        Comparator.comparing(node -> extractTitle(node))  
    )  
)
```

Now it's your turn

- **Hyperlinks in Refactoring Tasks**

- **task2_replaceAnonymousTypeWithLambda()**

DataResourceWorker.getDataResourceContentUploadPath()
EntityFunction.LENGTH.FETCHER
EntityFunction.TRIM.FETCHER
EntityFunction.UPPER.FETCHER
EntityFunction.LOWER.FETCHER
ModelWidgetCondition.DefaultConditionFactory.TRUE
ModelWidgetCondition.DefaultConditionFactory.FALSE
SSLUtil.getHostnameVerifier()
AuthHelper.getContextClassLoader()
ContentJsonEvents.getContentAssocs()
DelegatorEcaHandler.setDelegator()

3. Method References



3. Method References

- Lambdas often follow the same pattern

```
x -> x.f()  
x -> f(x)  
x -> new A(x)  
x -> B.f(x)
```

- Each of these can be changed to a *method reference*

x -> x.f()	=>	A::f
x -> f(x)	=>	this::f
x -> new A(x)	=>	A::new
x -> B.f(x)	=>	B::f

Examples from ofbiz

```
command -> command.getProperties()  
=> StartupCommand::getProperties
```

```
virtualHost -> host.addAlias(virtualHost)  
=> host::addAlias
```

```
connectorProp -> prepareConnector(connectorProp)  
=> this::prepareConnector
```

```
() -> createEntityEcaHandler()  
=> this::createEntityEcaHandler
```

```
() -> new HashMap<>()  
=> HashMap::new
```

```
() -> new LinkedHashSet<>()  
=> LinkedHashSet::new
```

```
set -> Collections.unmodifiableSet(set)  
=> Collections::unmodifiableSet
```

Now it's your turn

- **Hyperlinks in Refactoring Tasks**

- **task3_replaceLambdaWithMethodReference()**

CatalinaContainer.init()

CatalinaContainer.prepareVirtualHost()

CatalinaContainer.prepareTomcatConnectors()

ContainerLoader.filterContainersHavingMatchingLoaders()

EntityDataLoadContainer.init()

EntityUtil.filterByCondition()

GenericDelegator.initEntityEcaHandler()

GenericDelegator.initDistributedCacheClear()

MapContext.entrySet()

MultivaluedMapContextAdapter.entrySet()

ShippingEvents.getGeoIdFromPostalContactMech()

TestRunContainer.init()

UtilMisc.toMap()

4. Iterable and Map forEach()



4. Iterable and Map forEach()

- Both Iterable and Map have a forEach()
 - `Iterable<E>.forEach(Consumer<E>)`
 - `Map<K, V>.forEach(BiConsumer<K, V>)`

Iterable.forEach()

- Apply consumer to all entries
 - Should not be used for creating a new collection

```
for (GenericServiceCallback gsc : dispatcher.getCallbacks(model.name)) {  
    gsc.receiveEvent(context);  
}
```



```
dispatcher.getCallbacks(model.name)  
    .forEach(gsc -> gsc.receiveEvent(context));
```

Map.forEach()

- Instead of iterating over entrySet

```
for (Entry<? extends K, ? extends V> entry : m.entrySet()) {  
    adaptee.putSingle(entry.getKey(), entry.getValue());  
}
```



```
m.forEach((k, v) -> adaptee.putSingle(k, v));
```



```
m.forEach(adaptee::putSingle);
```

Now it's your turn

- **Hyperlinks in Refactoring Tasks**

- [task4_replaceLoopWithForEach\(\)](#)

Replace with Map.forEach()

MultivaluedMapContextAdapter.putAll()

CatalinaContainer.prepareContext()

Replace with Iterable.forEach()

AbstractEngine.sendCallbacks() x 3

- **Bonus: extract common code into separate method, passing in a Consumer of GenericServiceCallback**

5. removeIf()



5. removelf()

- What is wrong with this code?

```
List<Integer> evens = new ArrayList<>();
for (int i = 0; i < 1_000_000_000; i++) {
    evens.add(i);
}

// oh, we only wanted even numbers?
for (Iterator<Integer> it = evens.iterator(); it.hasNext(); ) {
    Integer i = it.next();
    if (i % 2 == 1) it.remove();
}
```

Quadratic performance

- It will take about a month to finish
 - Each time we remove an item, the items shift left

```
List<Integer> evens = new ArrayList<>();  
for (int i = 0; i < 1_000_000_000; i++) {  
    evens.add(i);  
}
```

// Linear performance, completing in seconds
evens.removeIf(i -> i % 2 == 1);

- Quadratic performance happens with array based lists
 - **ArrayList**, **Vector**, **CopyOnWriteArrayList**

Now it's your turn

- **Hyperlinks in Refactoring Tasks**
 - [task5_replaceLoopWithRemoveIf\(\)](#)

[ShoppingCart.clearPaymentMethodsById\(\)](#)

[ShoppingCart.cleanUpShipGroups\(\)](#)

[ShoppingCart.removeFreeShippingProductPromoAction\(\)](#)

[ShoppingCart.clearAllPromotionAdjustments\(\)](#)

[ShoppingCartItem.removeFeatureAdjustment\(\)](#)

6. Map Compound Methods



6. Map Compound Methods

- **getOrDefault(key, defaultValue)**
 - Returns a default value if the key is not in the map
- **putIfAbsent(key, value)**
 - Returns null if we were the first to put with that key; otherwise the old value
- **merge(key, value, remappingFunction)**
 - BiFunction<V, V, V> - merges two values into one
- **computeIfAbsent(key, mappingFunction)**
 - Function<K, V> return a new value for the key
 - Great for maps with values that are collections

Map.getOrDefault()

- Common coding pattern

```
if (positions.containsKey(name)) {  
    return positions.get(name);  
} else {  
    return -1;  
}
```

- Can get replaced with

```
return positions.getOrDefault(name, -1);
```

Map.putIfAbsent()

- Common coding pattern

```
if (returnInvoices.get(invoice.getString("invoiceId")) == null) {  
    returnInvoices.put(invoice.getString("invoiceId"), invoice);  
}
```

- Can get replaced with

```
returnInvoices.putIfAbsent(invoice.getString("invoiceId"), invoice);
```

Map.computeIfAbsent

- How many hash lookups are we doing?

```
UtilTimer timer = staticTimers.get(timerName);
if (timer == null) {
    timer = new UtilTimer(timerName, false);
    timer.setLog(log);
    staticTimers.putIfAbsent(timerName, timer);
    timer = staticTimers.get(timerName);
}
return timer;
```



```
return staticTimers.computeIfAbsent(timerName, key -> {
    UtilTimer timer = new UtilTimer(key, false);
    timer.setLog(log);
    return timer;
});
```

Map Compound Method Caveats

- Functions should not change map structure

- In some versions of Java, live lock can happen
 - In others, this will cause an exception

- Fibonacci "works" in Java 8, exception in Java 9

```
public class Fibonacci {  
    private final Map<Integer, BigInteger> cache = new HashMap<>();  
  
    cache.put(0, BigInteger.ZERO);  
    cache.put(1, BigInteger.ONE);  
  
    public BigInteger f(int n) {  
        if (n < 0) throw new IllegalArgumentException();  
        return cache.computeIfAbsent(n, key -> f(key - 1).add(f(key - 2)));  
    }  
}
```

Now it's your turn

- **Hyperlinks in Refactoring Tasks**

- **task6_replaceWithCompoundMapMethods()**

Map.getOrDefault()

AIMRespPositions.getPosition()

CPRespPositions.getPosition()

RequestHandler.renderView()

TaxAuthorityServices.rateProductTaxCalc()

Map.putIfAbsent()

OrderReturnServices.createPaymentApplicationsFromReturnItemResponse()

Converters.getConverter()

Map.merge()

ShoppingCartItem.resetPromoRuleUse()

ShoppingCartItem.confirmPromoRuleUse()

OrderReadHelper.getOrderNonReturnedTaxAndShipping()

Map Compound Methods (continued)

Map.computeIfAbsent()

UtilTimer.getTimer()

UtilCache.getNextDefaultIndex()

DelegatorFactory.getDelegatorFuture()

GenericDAO.getGenericDAO()

ContentManagementWorker.getStaticValue()

DatabaseUtil.getColumnInfo()

EntityEcaUtil.readConfig()

FindServices.prepareField()

ModelReader.buildEntity()

ModelReader.rebuildResourceHandlerEntities()

ModelReader.getEntitiesByPackage()

ParametricSearch.makeCategoryFeatureLists()

ShoppingCartServices.loadCartFromQuote()

Streams

Object and primitive streams, lazy evaluation, debugging



Streams

- We can create streams from any Iterable
 - `list.stream()`
 - `set.stream()`
 - `queue.stream()`
 - `map.entrySet().stream()`
- Or from arrays
 - `Stream.of(new String[] {"John", "Anton", "Heinz"})`
 - `IntStream.of(99, 72, 56)` or `IntStream.range(0, 100)`
 - `LongStream.of(100, 200, 300)`
 - `DoubleStream.of(65.3, 114.5, 123.8)`

7. Stream.all/any/ noneMatch()



7. Stream.all/any/noneMatch()

- Stream can return boolean
 - Takes a Predicate as a parameter

anyMatch()

- Any element has to match predicate
 - If any matches, we immediately return true

```
for (ModelField mf : getFieldsUnmodifiable()) {  
    if (mf.getEnableAuditLog()) {  
        return true;  
    }  
}  
return false;
```



```
return getFieldsUnmodifiable().stream()  
.anyMatch(ModelField::getEnableAuditLog);
```

allMatch()

- All elements have to match predicate
 - If any does not match, we immediately return false

```
String fullPath = dir.getPath().replace('\\\\', '/');  
boolean hasAllPathStrings = true;  
for (String pathString: stringsToFindInPath) {  
    if (!fullPath.contains(pathString)) {  
        hasAllPathStrings = false;  
        break;  
    }  
}
```



```
String fullPath = dir.getPath().replace('\\\\', '/');  
boolean hasAllPathStrings = stringsToFindInPath.stream()  
    .allMatch(fullPath::contains);
```

noneMatch()

- No elements may match predicate
 - If any does match, we immediately return false

```
for (String element : validOut) {  
    if (name.equals(element)) {  
        return false;  
    }  
}  
return true;
```



```
return Stream.of(validOut).noneMatch(name::equals);
```

Now it's your turn

- **Hyperlinks in Refactoring Tasks**

- `task7_replaceWithAllAnyNoneMatch()`

anyMatch()

`ModelEntity.getHasFieldWithAuditLog()`

`ProductPromoWorker.hasOrderTotalCondition()`

allMatch()

`FileUtil.SearchTextFilesFilter.accept()`

`ModelEntity.areFields()`

`EntityJoinOperator.isEmpty()`

noneMatch()

`LoginWorker.hasApplicationPermission()`

`PcChargeApi.checkIn()`

`PcChargeApi.checkOut()`

8. Stream.map() and collect()



8. Stream.map() and collect()

- **map()** converts from one type to another
 - **mapToInt()** converts object to int for an IntStream
 - **mapToLong()** converts object to long for a LongStream
 - **mapToDouble()** converts object to double for a DoubleStream
 - **mapToObj()** a primitive stream to an object stream
 - **boxed()** converts primitive stream to its wrapper classes
- **collect(Collector)** converts a stream to a collection
 - **Collectors.toSet()** converts stream to HashSet
 - **Collectors.toList()** converts stream to ArrayList

Transforming Loop to map()/collect()

```
List<String> nameList = new ArrayList<>();  
for (ModelField field: modelFields) {  
    nameList.add(field.getName());  
}  
return nameList;
```



```
Stream<ModelField> modelFieldsStream = modelFields.stream();  
Stream<String> nameStream = modelFieldsStream.map(ModelField::getName);  
List<String> nameList = nameStream.collect(Collectors.toList());  
return nameList;
```



```
return modelFields.stream() // Stream<ModelField>  
.map(ModelField::getName) // Stream<String>  
.collect(Collectors.toList()); // List<String>
```

Transforming Loop to map()/collect()

```
List<String> nameList = new ArrayList<>();  
for (ModelField field: modelFields) {  
    nameList.add(field.getName());  
}  
return nameList;
```



```
Stream<ModelField> modelFieldsStream = modelFields.stream();  
Stream<String> nameStream = modelFieldsStream.map(ModelField::getName);  
List<String> nameList = nameStream.collect(Collectors.toList());  
return nameList;
```



```
return modelFields.stream() // Stream<ModelField>  
    .map(ModelField::getName) // Stream<String>  
    .collect(Collectors.toList()); // List<String>
```

Transforming Loop to map()/collect()

```
List<String> nameList = new ArrayList<>();  
for (ModelField field: modelFields) {  
    nameList.add(field.getName());  
}  
return nameList;
```



```
Stream<ModelField> modelFieldsStream = modelFields.stream();  
Stream<String> nameStream = modelFieldsStream.map(ModelField::getName);  
List<String> nameList = nameStream.collect(Collectors.toList());  
return nameList;
```



```
return modelFields.stream() // Stream<ModelField>  
    .map(ModelField::getName) // Stream<String>  
    .collect(Collectors.toList()); // List<String>
```

Transforming Loop to map()/collect()

```
List<String> nameList = new ArrayList<>();  
for (ModelField field: modelFields) {  
    nameList.add(field.getName());  
}  
return nameList;
```



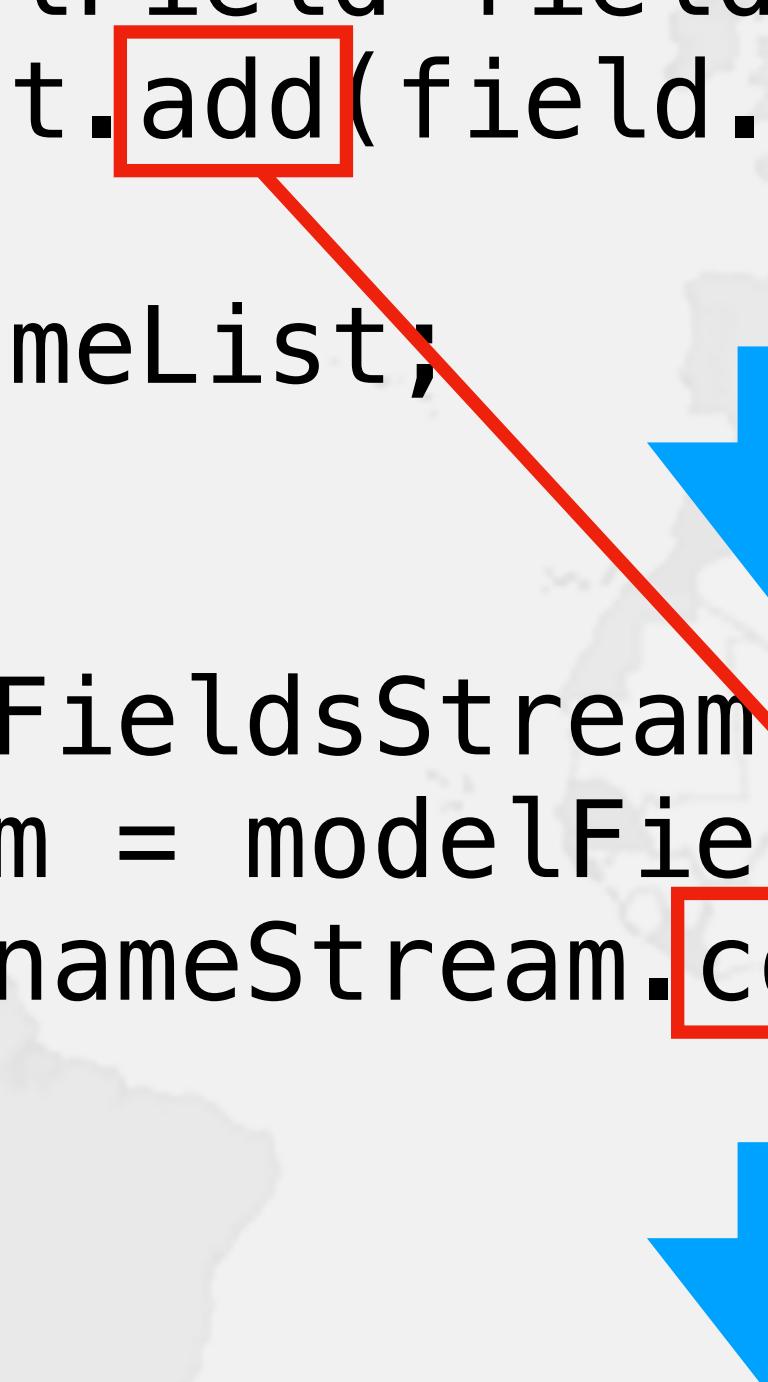
```
Stream<ModelField> modelFieldsStream = modelFields.stream();  
Stream<String> nameStream = modelFieldsStream.map(ModelField::getName);  
List<String> nameList = nameStream.collect(Collectors.toList());  
return nameList;
```



```
return modelFields.stream() // Stream<ModelField>  
    .map(ModelField::getName) // Stream<String>  
    .collect(Collectors.toList()); // List<String>
```

Transforming Loop to map()/collect()

```
List<String> nameList = new ArrayList<>();  
for (ModelField field: modelFields) {  
    nameList.add(field.getName());  
}  
return nameList;
```



```
Stream<ModelField> modelFieldsStream = modelFields.stream();  
Stream<String> nameStream = modelFieldsStream.map(ModelField::getName);  
List<String> nameList = nameStream.collect(Collectors.toList());  
return nameList;
```

```
return modelFields.stream() // Stream<ModelField>  
.map(ModelField::getName) // Stream<String>  
.collect(Collectors.toList()); // List<String>
```

Now it's your turn

- **Hyperlinks in Refactoring Tasks**
 - `task8_replaceWithMapCollect()`

`ModelEntity.getFieldNamesFromFieldVector()`

`ModelReader.getEntityCache()`

`DelegatorContainer.start()`

`ContainerConfig.getConfigurationPropsFromXml()`

`PaymentGatewayServices.capturePaymentsByInvoice()`

9. Collectors .toCollection()



9. Collectors.toCollection()

- We can also specify **Supplier<Collection>**
 - **toList()** makes **ArrayList** and **toSet()** makes **HashSet**

```
List<V> valuesList = new LinkedList<>();  
for (CacheLine<V> line: memoryTable.values()) {  
    valuesList.add(line.getValue());  
}  
return valuesList;
```



```
return memoryTable.values().stream()  
    .map(CacheLine::getValue)  
    .collect(Collectors.toCollection(LinkedList::new));
```

Now it's your turn

- **Hyperlinks in Refactoring Tasks**
 - **task9_replaceWithMapCollectToCollection()**

`EntityJoinOperator.freeze()`

`UtilCache.values()`

`UtilDateTime.TimeZoneHolder.getTimeZones()`

10. Stream.filter()



10. Stream.filter()

- Stream.filter() predicate of what to keep

```
List<EntityCondition> entityConditionList =  
    new ArrayList<>();  
for (Condition cond: conditionList) {  
    EntityCondition econd = cond.createCondition(  
        context, modelEntity, modelFieldTypeReader);  
    if (econd != null) {  
        entityConditionList.add(econd);  
    }  
}
```



```
List<EntityCondition> entityConditionList = conditionList.stream()  
    .map(cond -> cond.createCondition(  
        context, modelEntity, modelFieldTypeReader))  
    .filter(Objects::nonNull)  
    .collect(Collectors.toList());
```

Now it's your turn

- **Hyperlinks in Refactoring Tasks**
 - **task10_replaceWithMapFilterCollect()**

`EntityDataLoader.getUrlByComponentList()`

`EntityFinderUtil.ConditionList.createCondition()`

`ContainerConfig.Configuration.getPropertiesWithValue()`

11. Collectors .toMap()



11. Collectors.toMap()

- We can also collect to a Map

```
Map<String, ModelMenu> modelMenuMap = new HashMap<>();
for (Element menuElement: UtilXml.childElementList(rootElement, "menu")) {
    ModelMenu modelMenu = new ModelMenu(menuElement, menuLocation, visualTheme);
    modelMenuMap.put(modelMenu.getName(), modelMenu);
}
```



```
modelMenuMap = UtilXml.childElementList(rootElement, "menu").stream()
    .map(menuElement -> new ModelMenu(menuElement, menuLocation, visualTheme))
    .collect(Collectors.toMap(
        ModelWidget::getName, // how key is generated
        modelMenu -> modelMenu, // how value is generated (or Function.identity())
        (a, b) -> b)); // how duplicates are resolved
```

Now it's your turn

- **Hyperlinks in Refactoring Tasks**
 - **task11_replaceWithStreamCollectToMap()**

`CheckOutHelper.makeBillingAccountMap()`

`ComponentConfig.ComponentConfig()`

`MenuFactory.readMenuDocument()`

`ModelScreenWidget.DecoratorScreen.DecoratorScreen()`

12. Stream .reduce()



12. Stream.reduce()

- Can merge all values into one with reduce()

```
BigDecimal totalAmount = BigDecimal.ZERO;  
for (Map.Entry<String, String> rowEntry : amountMap.entrySet()) {  
    if (UtilValidate.isNotEmpty(rowEntry.getValue())) {  
        totalAmount = totalAmount.add(new BigDecimal(rowEntry.getValue()));  
    }  
}  
return totalAmount;
```



```
return amountMap.entrySet().stream()  
    .map(Map.Entry::getValue)  
    .filter(UtilValidate::isNotEmpty)  
    .map(BigDecimal::new)  
    .reduce(BigDecimal.ZERO, BigDecimal::add);
```

12. Stream.reduce()

- Can merge all values into one with reduce()

```
BigDecimal totalAmount = BigDecimal.ZERO;  
for (Map.Entry<String, String> rowEntry : amountMap.entrySet()) {  
    if (UtilValidate.isNotEmpty(rowEntry.getValue())) {  
        totalAmount = totalAmount.add(new BigDecimal(rowEntry.getValue()));  
    }  
}  
return totalAmount;  
  
return amountMap.entrySet().stream()  
    .map(Map.Entry::getValue)  
    .filter(UtilValidate::isNotEmpty)  
    .map(BigDecimal::new)  
    .reduce(BigDecimal.ZERO, BigDecimal::add);
```



12. Stream.reduce()

- Can merge all values into one with reduce()

```
BigDecimal totalAmount = BigDecimal.ZERO;
for (Map.Entry<String, String> rowEntry : amountMap.entrySet()) {
    if (UtilValidate.isNotEmpty(rowEntry.getValue())) {
        totalAmount = totalAmount.add(new BigDecimal(rowEntry.getValue()));
    }
}
return totalAmount;
```

↓

```
return amountMap.entrySet().stream()
    .map(Map.Entry::getValue)
    .filter(UtilValidate::isNotEmpty)
    .map(BigDecimal::new)
    .reduce(BigDecimal.ZERO, BigDecimal::add);
```

12. Stream.reduce()

- Can merge all values into one with reduce()

```
BigDecimal totalAmount = BigDecimal.ZERO;
for (Map.Entry<String, String> rowEntry : amountMap.entrySet()) {
    if (UtilValidate.isNotEmpty(rowEntry.getValue())) {
        totalAmount = totalAmount.add(new BigDecimal(rowEntry.getValue()));
    }
}
return totalAmount;
```



```
return amountMap.entrySet().stream()
    .map(Map.Entry::getValue)
    .filter(UtilValidate::isNotEmpty)
    .map(BigDecimal::new)
    .reduce(BigDecimal.ZERO, BigDecimal::add);
```

12. Stream.reduce()

- Can merge all values into one with reduce()

```
BigDecimal totalAmount = BigDecimal.ZERO;
for (Map.Entry<String, String> rowEntry : amountMap.entrySet()) {
    if (UtilValidate.isNotEmpty(rowEntry.getValue())) {
        totalAmount = totalAmount.add(new BigDecimal(rowEntry.getValue())));
    }
}
return totalAmount;
```



```
return amountMap.entrySet().stream()
    .map(Map.Entry::getValue)
    .filter(UtilValidate::isNotEmpty)
    .map(BigDecimal::new)
    .reduce(BigDecimal.ZERO, BigDecimal::add);
```

12. Stream.reduce()

- Can merge all values into one with reduce()

```
BigDecimal totalAmount = BigDecimal.ZERO;
for (Map.Entry<String, String> rowEntry : amountMap.entrySet()) {
    if (UtilValidate.isNotEmpty(rowEntry.getValue())) {
        totalAmount = totalAmount.add(new BigDecimal(rowEntry.getValue()));
    }
}
return totalAmount;
```

↓

```
return amountMap.entrySet().stream()
    .map(Map.Entry::getValue)
    .filter(UtilValidate::isNotEmpty)
    .map(BigDecimal::new)
    .reduce(BigDecimal.ZERO, BigDecimal::add);
```

12. Stream.reduce()

- Can merge all values into one with reduce()

```
BigDecimal totalAmount = BigDecimal.ZERO;  
for (Map.Entry<String, String> rowEntry : amountMap.entrySet()) {  
    if (UtilValidate.isNotEmpty(rowEntry.getValue())) {  
        totalAmount = totalAmount.add(new BigDecimal(rowEntry.getValue()));  
    }  
}  
return totalAmount;
```

```
return amountMap.entrySet().stream()  
.map(Map.Entry::getValue)  
.filter(UtilValidate::isEmpty)  
.map(BigDecimal::new)  
.reduce(BigDecimal.ZERO, BigDecimal::add);
```

Now it's your turn

- **Hyperlinks in Refactoring Tasks**
 - `task12_replaceWithReduce()`

`GeneralLedgerServices.calculateCostCenterTotal()`
`InvoiceServices.updatePaymentApplicationDefBd()`
`OrderReadHelper.calcOrderPromoAdjustmentsBd()`

13. Stream .flatMap()



13. Stream.flatMap()

- flatMap() extracts items from nested streams

```
List<ClasspathInfo> classpaths = new ArrayList<>();  
for (ComponentConfig cc : getAllComponents()) {  
    if (componentName == null  
        || componentName.equals(cc.getComponentName())) {  
        classpaths.addAll(cc.getClasspathInfos());  
    }  
}  
return classpaths;
```



```
return getAllComponents().stream()  
    .filter(cc -> componentName == null  
           || componentName.equals(cc.getComponentName()))  
    .map(ComponentConfig::getClasspathInfos)  
    .flatMap(Collection::stream)  
    .collect(Collectors.toList());
```

13. Stream.flatMap()

- flatMap() extracts items from nested streams

```
List<ClasspathInfo> classpaths = new ArrayList<>();
for (ComponentConfig cc : getAllComponents()) {
    if (componentName == null
        || componentName.equals(cc.getComponentName())))
        classpaths.addAll(cc.getClasspathInfos());
}
return classpaths;
```

```
return getAllComponents().stream()
    .filter(cc -> componentName == null
           || componentName.equals(cc.getComponentName()))
    .map(ComponentConfig::getClasspathInfos)
    .flatMap(Collection::stream)
    .collect(Collectors.toList());
```

13. Stream.flatMap()

- flatMap() extracts items from nested streams

```
List<ClasspathInfo> classpaths = new ArrayList<>();
for (ComponentConfig cc : getAllComponents()) {
    if (componentName == null
        || componentName.equals(cc.getComponentName())) {
        classpaths.addAll(cc.getClasspathInfos());
    }
}
return classpaths;
```

```
return getAllComponents().stream()
    .filter(cc -> componentName == null
        || componentName.equals(cc.getComponentName()))
    .map(ComponentConfig::getClasspathInfos)
    .flatMap(Collection::stream)
    .collect(Collectors.toList());
```

13. Stream.flatMap()

- flatMap() extracts items from nested streams

```
List<ClasspathInfo> classpaths = new ArrayList<>();
for (ComponentConfig cc : getAllComponents()) {
    if (componentName == null
        || componentName.equals(cc.getComponentName())))
        classpaths.addAll(cc.getClasspathInfos());
}
return classpaths;
```

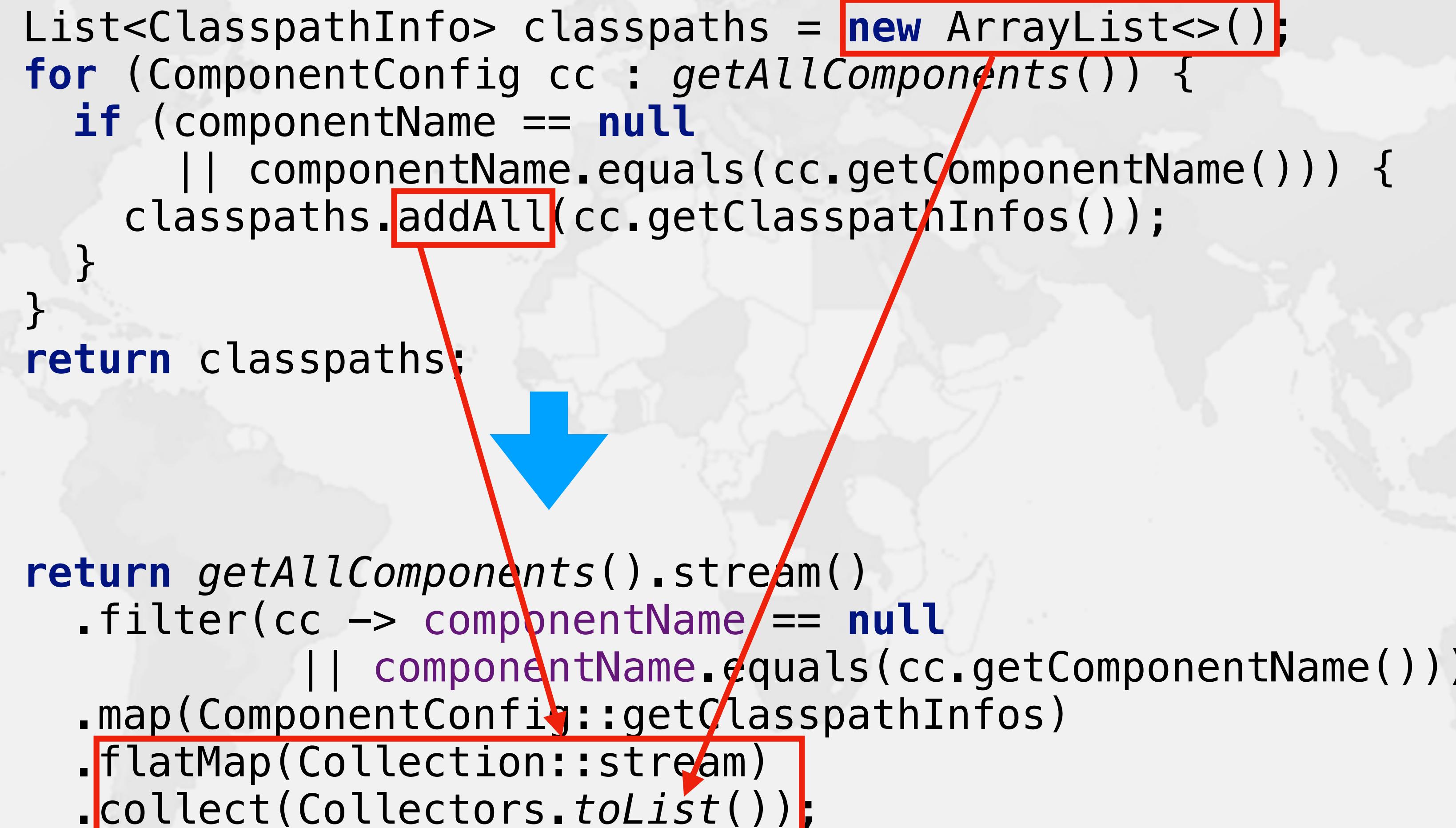


```
return getAllComponents().stream()
    .filter(cc -> componentName == null
        || componentName.equals(cc.getComponentName()))
    .map(ComponentConfig::getClasspathInfos)
    .flatMap(Collection::stream)
    .collect(Collectors.toList());
```

13. Stream.flatMap()

- flatMap() extracts items from nested streams

```
List<ClasspathInfo> classpaths = new ArrayList<>();  
for (ComponentConfig cc : getAllComponents()) {  
    if (componentName == null  
        || componentName.equals(cc.getComponentName())) {  
        classpaths.addAll(cc.getClasspathInfos());  
    }  
}  
return classpaths;  
  
return getAllComponents().stream()  
    .filter(cc -> componentName == null  
           || componentName.equals(cc.getComponentName()))  
    .map(ComponentConfig::getClasspathInfos)  
    .flatMap(Collection::stream)  
    .collect(Collectors.toList());
```



Now it's your turn

- **Hyperlinks in Refactoring Tasks**
 - **task13_replaceWithFlatMap()**

ComponentConfig.getAllClasspathInfos()

ComponentConfig.getAllConfigurations()

ComponentConfig.getAllKeystoreInfos()

ComponentConfig.getAllTestSuiteInfos()

ComponentConfig.getAllWebappResourceInfos()

14. Optional, findFirst(), findAny()



14. Optional, findFirst(), findAny()

- A method might not have a good return value
 - For example, `findFirst()` on an empty stream?
- Most important methods on `Optional` are
 - `ifPresent(Consumer)`
 - `map(Function)`
 - `orElse(other)`, `orElseGet(otherSupplier)`,
`orElseThrow(exceptionSupplier)`
 - Java 9: `ifPresentOrElse(Consumer, Runnable)`
- We create `Optional` instances with
 - `Optional.empty()`, `Optional.of(val)`, `Optional.ofNullable(val)`

Returning an Optional from Stream

- **findFirst(), findAny(), max(), min(), reduce()**

```
for (ModelKeyMap keyMap : keyMaps) {  
    if (keyMap.getFieldName().equals(fieldName))  
        return keyMap;  
}  
return null;
```



```
Optional<ModelKeyMap> modelKeyMap = keyMaps.stream()  
    .filter(keyMap -> keyMap.getFieldName().equals(fieldName))  
    .findFirst();  
return modelKeyMap.orElse(null);
```

Now it's your turn

- **Hyperlinks in Refactoring Tasks**
 - `task14_replaceFindFirstOrAny()`

`ModelRelation.findKeyMap()`

`ModelRelation.findKeyMapByRelated()`

`ShoppingCartItem.updatePrice()`

`OrderReadHelper.getShippableSizes()`

15. groupingBy(), mapping()



15. groupingBy(), mapping()

- We can create a Map from a stream
 - Function for the key
 - Collector for the downstream values
 - Can be a collection or a reduced value

groupingBy(Function)

- Stream<E> to Map<K, List<V>>

```
Stream<String> numbers = Stream.of(  
    "one", "two", "three", "four",  
    "five", "six", "seven", "eight");  
Map<Integer, List<String>> map = numbers.collect(  
    Collectors.groupingBy(  
        String::length // key in the map  
    )  
);  
System.out.println(map.getClass());  
map.entrySet().forEach(System.out::println);
```

```
class java.util.HashMap  
3=[one, two, six]  
4=[four, five]  
5=[three, seven, eight]
```

groupingBy() With Collector

- Stream<E> to Map<K, Collection<V>>

```
Stream<String> numbers = ...
Map<Integer, Collection<String>> map = numbers.collect(
    Collectors.groupingBy(
        String::length,
        // type of collection for values
        Collectors.toCollection(TreeSet::new)
    )
);
System.out.println(map.getClass());
map.entrySet().forEach(System.out::println);
```

Strings sorted alphabetically

class java.util.HashMap
3=[one, six, two]
4=[five, four]
5=[eight, seven, three]

groupingBy() With Supplier<Map>

- Stream<E> to TreeMap<K, TreeSet<V>>

```
Stream<String> numbers = ...
TreeMap<Integer, TreeSet<String>> map = numbers.collect(
    Collectors.groupingBy(
        String::length,
        TreeMap::new, // type of map
        Collectors.toCollection(TreeSet::new)
    )
);
System.out.println(map.getClass());
map.entrySet().forEach(System.out::println);
```

Map is now a TreeMap

class java.util.TreeMap
3=[one, six, two]
4=[five, four]
5=[eight, seven, three]

groupingBy() With mapping()

- Stream<E> to Map<K, HashSet<V>>

```
Stream<String> numbers = ...  
Map<Integer, HashSet<String>> map = numbers.collect(  
    Collectors.groupingBy(  
        String::length,  
        Collectors.mapping(  
            String::toUpperCase, // value in the collection  
            Collectors.toCollection(HashSet::new)  
        )  
    )  
);  
System.out.println(map.getClass());  
map.entrySet().forEach(System.out::println);
```

Strings are upper case

class java.util.HashMap
3=[SIX, ONE, TWO]
4=[FIVE, FOUR]
5=[EIGHT, THREE, SEVEN]

groupingBy() With counting()

- Stream<E> to Map<K, Long>

```
Stream<String> numbers = ...
Map<Integer, Long> map = numbers.collect(
    Collectors.groupingBy(
        String::length, // key in the map
        Collectors.counting() // count how many of each key
    )
);
System.out.println(map.getClass());
map.entrySet().forEach(System.out::println);
```

```
class java.util.HashMap
3=3
4=2
5=3
```

Now it's your turn

- **Hyperlinks in Refactoring Tasks**
 - `task15_replaceWithCollectGroupingByMapping()`
`ModelReader.rebuildResourceHandlerEntities()`

16. Checked Exceptions



16. Checked Exceptions

- Streams do not support checked exceptions
 - The Java language architects were aware of this
 - But underestimated the pain level

"Sneaky Throw"

- Uses vacuous cast trick

```
class SneakyThrower {  
    private SneakyThrower() { }  
    static void rethrow(Throwable ex) {  
        SneakyThrower.<RuntimeException>uncheckedThrow(ex);  
    }  
    @SuppressWarnings("unchecked")  
    private static <T extends Throwable>  
    void uncheckedThrow(Throwable t) throws T {  
        if (t != null)  
            throw (T) t; // rely on vacuous cast  
        else  
            throw new Error("Unknown Exception");  
    }  
}
```

Throwing Functional Interfaces

- Need to cast to the ThrowingFunction

- Custom Function we added to the project

```
public interface ThrowingFunction<T, R>
    extends Function<T, R> {
    default R apply(T t) {
        try {
            return applyWithThrow(t);
        } catch (Throwable ex) {
            SneakyThrower.rethrow(ex);
            throw new AssertionError(ex);
        }
    }
    R applyWithThrow(T t) throws Throwable;
}
```

Transforming with Exceptions

- Casting lambda to a ThrowingFunction

```
List<GenericValue> currentPayments = new LinkedList<>();  
for (GenericValue paymentPref : orderPaymentPrefs) {  
    List<GenericValue> payments =  
        paymentPref.getRelated("Payment", null, null, false);  
    currentPayments.addAll(payments);  
}
```



```
List<GenericValue> currentPayments = orderPaymentPrefs.stream()  
    .flatMap((ThrowingFunction<GenericValue, Stream<GenericValue>>)   
        paymentPref -> paymentPref.getRelated("Payment", null, null, false).stream())  
    .collect(Collectors.toCollection(LinkedList::new));
```

Now it's your turn

- **Hyperlinks in Refactoring Tasks**
 - **task16_checkedExceptions**

`EntityUtil.getRelated()`

`ModelReader.getEntitiesByPackage()`

Conclusion



Conclusion

- Where to next?
 - Mastering Lambdas - Maurice Naftalin
 - Practice, practice, practice
 - Use Analyze -> Inspect to find more places to refactor
 - Do `task99_forTheSuperKeen()`