JavaSpecialists

Refactoring to Streams 2.1

Dr Heinz M. Kabutz

Last updated 2023-04-26

Copyright Notice

JavaSpecialists

Introduction

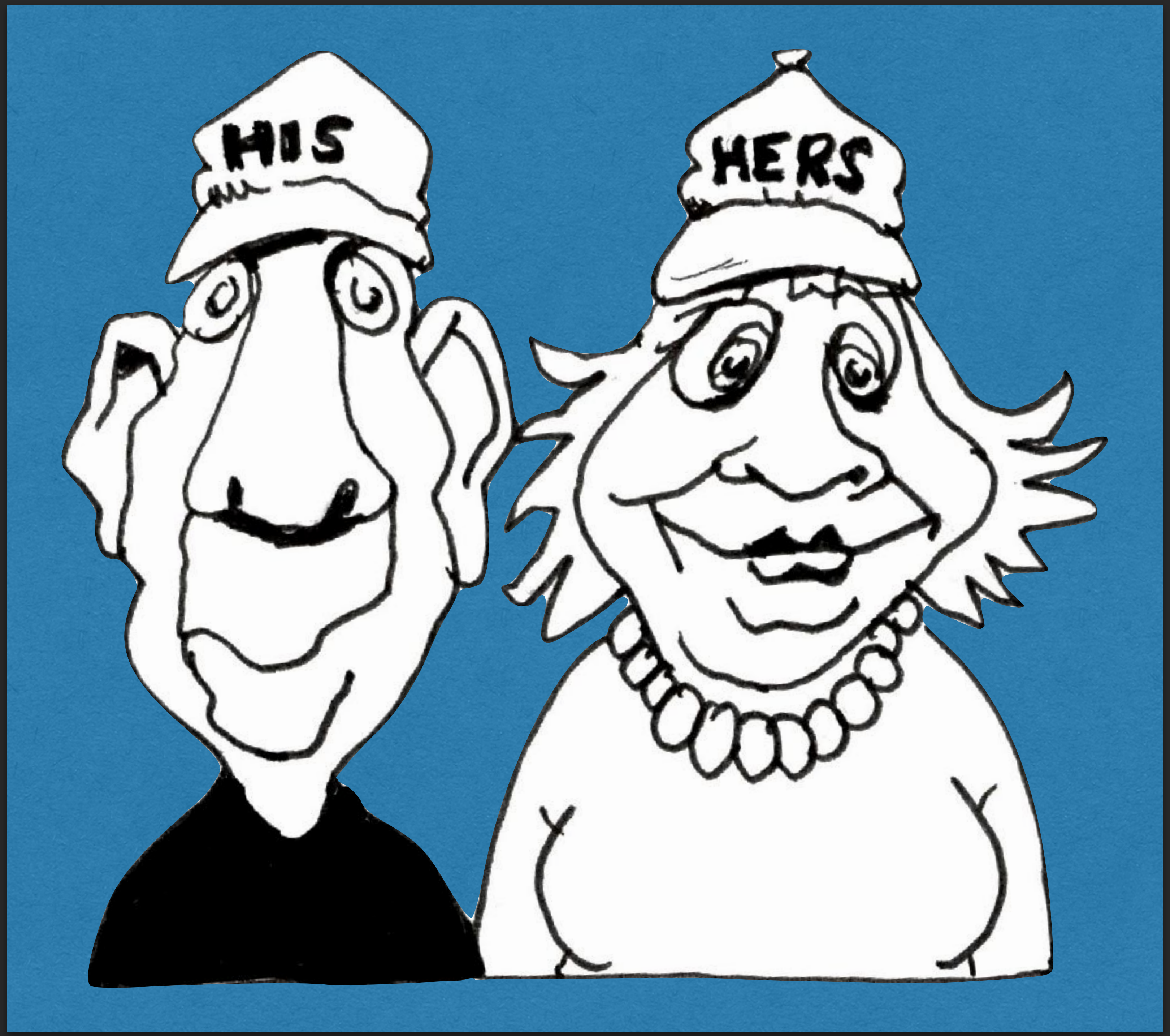
Warm Welcome - Course Author

◉ **Dr Heinz Kabutz**

- Born in Cape Town, now lives on Crete, Greece
- 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 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**
  - Project currently builds only up until Java 17
- **IntelliJ IDEA 2022.1 or later**
  - – Recommended IDEA 2023.1+
- **Get project using git**
  - – Will send you URL and credentials now

JavaSpecialists

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

JavaSpecialists

Inspecting Code with IntelliJ IDEA

Quick demo

JavaSpecialists

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 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 and method references
  - Compound Map methods
    - getOrDefault(), putIfAbsent(), computeIfAbsent(), merge(), etc.
  - Collection.removeIf()
  - Optional
  - Streams
    - Primitive vs Object streams
    - allMatch(), map(), filter(), findFirst(), collect(), etc.
    - Spliterators
    - Sequential vs parallel streams

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 type can be omitted (var)
- **Java 14**
  - Switch Expressions (JEP 361)

Java Language Changes

- **Java 15**
  - Text blocks (JEP 378)
- **Java 16**
  - Records (JEP 395)
  - Pattern Matching for instanceof (JEP 394)
- **Java 17**
  - Sealed classes (JEP 409)
- **Java 21**
  - Virtual threads with Project Loom

JavaSpecialists

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());
```

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 RefactoringTasks.java**
  - 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()**
  - Use key extractors with Comparator.comparing(Function<T, U>)
  - Null is not managed correctly, compare is not transitive
    - Use Comparator.nullsFirst(Comparator)

```java
nodes.sort(new Comparator<Map<String, Object>>() {
    public int compare(Map<String, Object> node1, Map<String, Object> node2) {
        Map<String, Object> data1 = (Map<String, Object>) node1.get("data");
        Map<String, Object> data2 = (Map<String, Object>) 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()));
    }});
```

JavaSpecialists

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) - takes a value and returns boolean
  - Consumer (8) - takes a value and returns void
  - Function (25) - takes a value and returns value
  - Supplier (5) - takes no value and returns a value
- **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**

```java
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());
            }
        })));
```

JavaSpecialists

2. Lambdas

2. Lambdas

- Tedious 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 ->**
  - Some render it as →

```
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 type 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 RefactoringTasks**
  - 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**
  - Can sometimes be converted into a *method reference*

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

3. Method References

- **Lambdas often follow the same pattern**
  - Can sometimes be converted into a *method reference*

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

3. Method References

- **Lambdas often follow the same pattern**
  - Can sometimes be converted into a *method reference*

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

3. Method References

- **Lambdas often follow the same pattern**
  - Can sometimes be converted into a *method reference*

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

3. Method References

- **Lambdas often follow the same pattern**
  - Can sometimes be converted into a *method reference*

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

Examples from ofbiz

```
command -> command.getProperties()
```

Examples from ofbiz

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

Examples from ofbiz

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

Examples from ofbiz

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

Examples from ofbiz

```
connectorProp -> prepareConnector(connectorProp)
```

Examples from ofbiz

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

Examples from ofbiz

```
() -> createEntityEcaHandler()
```

Examples from ofbiz

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

Examples from ofbiz

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

Examples from ofbiz

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

Examples from ofbiz

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

Examples from ofbiz

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

Examples from ofbiz

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

Examples from ofbiz

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

Now it's your turn

- **Hyperlinks in RefactoringTasks**
  - 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()
```

JavaSpecialists

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);
}
```

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());
}
```

Map.forEach()

- Instead of iterating over entrySet

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

```
m.forEach((key, value) -> adaptee.putSingle(key, value));
```

Map.forEach()

- Instead of iterating over entrySet

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

```
m.forEach((key, value) -> adaptee.putSingle(key, value));
```

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

Now it's your turn

- **Hyperlinks in RefactoringTasks**
  - 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

JavaSpecialists

5. removeIf()

5. removeIf()

- **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 remaining items shift left

```
List<Integer> evens = new ArrayList<>();
for (int i = 0; i < 1_000_000_000; i++) {
    evens.add(i);
}
// linear performance O(n), completing in seconds
evens.removeIf(i -> i % 2 == 1);
```

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

Now it's your turn

- **Hyperlinks in RefactoringTasks**
  - task5_replaceLoopWithRemoveIf()

```
ShoppingCart.clearPaymentMethodsById()
ShoppingCart.cleanUpShipGroups()
ShoppingCart.removeFreeShippingProductPromoAction()
ShoppingCart.clearAllPromotionAdjustments()
ShoppingCartItem.removeFeatureAdjustment()
```

JavaSpecialists

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;
}
```

Map.getOrDefault()

◉ **Common coding pattern**

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

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

Map.putIfAbsent()

- Common coding pattern

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

Map.putIfAbsent()

- Common coding pattern

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

```
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;
```

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

Now it's your turn

- **Hyperlinks in RefactoringTasks**
  - task6_replaceWithCompoundMapMethods()

```
Map.getOrDefault()
  AIMRespPositions.getPosition()
  CPRespPositions.getPosition()
  RequestHandler.renderView()
  TaxAuthorityServices.rateProductTaxCalc()
Map.putIfAbsent()
  OrderReturnServices.createPaymentApplicationsFromReturnItem
  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()
```

JavaSpecialists

Streams

Object and primitive streams, lazy evaluation, debugging

Streams

- **We can create streams from any Iterable**
  - collection.stream()
  - map.entrySet().stream()
  - StreamSupport.stream(iterable.spliterator(), false)
- **Or from arrays**
  - Arrays.stream("John", "Anton", "Heinz")
    - Also Stream.of("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)

JavaSpecialists

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;
```

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

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

allMatch()

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

```
boolean hasAllPathStrings = true;
String fullPath = dir.getPath().replace('\\', '/');
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;
```

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 Arrays.stream(validOut).noneMatch(name::equals);
```

Now it's your turn

- **Hyperlinks in RefactoringTasks**
  - task7_replaceWithAllAnyNoneMatch()

```
anyMatch()
  ModelEntity.getHasFieldWithAuditLog()
  ProductPromoWorker.hasOrderTotalCondition()
allMatch()
  FileUtil.SearchTextFilesFilter.accept()
  ModelEntity.areFields()
  EntityJoinOperator.isEmpty()
noneMatch()
  LoginWorker.hasApplicationPermission()
  PcChargeApi.checkIn()
  PcChargeApi.checkOut()
```

JavaSpecialists

8. Stream.map() and collect()

8. Stream.map() and collect()

- **map() converts from one type to another**
  - mapToInt() converts elements to int for an IntStream
  - mapToLong() converts elements to long for a LongStream
  - mapToDouble() converts elements 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;
```

Transforming Loop to map()/collect()

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

```java
return modelFields.stream()
        .map(ModelField::getName)
        .collect(Collectors.toList());
```

Transforming Loop to map()/collect()

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

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

Transforming Loop to map()/collect()

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

```java
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;
```

```
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;
```

```
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;
```

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

Now it's your turn

- **Hyperlinks in RefactoringTasks**
  - task8_replaceWithMapCollect()

```
ModelEntity.getFieldNamesFromFieldVector()
ModelReader.getEntityCache()
DelegatorContainer.start()
ContainerConfig.getConfigurationPropsFromXml()
PaymentGatewayServices.capturePaymentsByInvoice()
```

JavaSpecialists

9. Collectors .toCollection()

9. Collectors.toCollection()

- **We can also specify Supplier<Collection>**
  - Thus we can create any type of collection from our stream

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

9. Collectors.toCollection()

- **We can also specify Supplier<Collection>**
  - Thus we can create any type of collection from our stream

```
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 RefactoringTasks**
  - task9_replaceWithMapCollectToCollection()

```
EntityJoinOperator.freeze()
UtilCache.values()
UtilDateTime.TimeZoneHolder.getTimeZones()
```

JavaSpecialists

10. Stream.filter()

10. Stream.filter()

- **Stream.filter() predicate of what to keep**

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

10. Stream.filter()

◉ **Stream.filter() predicate of what to keep**

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

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

Now it's your turn

- **Hyperlinks in RefactoringTasks**
  - task10_replaceWithMapFilterCollect()

```
EntityDataLoader.getUrlByComponentList()
EntityFinderUtil.ConditionList.createCondition()
ContainerConfig.Configuration.getPropertiesWithValue()
ModelReader.getEntityCache()
```

JavaSpecialists

11. Collectors .toMap()

11. Collectors.toMap()

- **We can also collect to a Map**

```
Map<String, ModelMenu> modelMenuMap = new HashMap<>();
for (Element element : childElementList(rootElement, "menu")) {
    ModelMenu menu = new ModelMenu(element, location, theme);
    modelMenuMap.put(menu.getName(), menu);
}
```

11. Collectors.toMap()

- **We can also collect to a Map**

```
Map<String, ModelMenu> modelMenuMap = new HashMap<>();
for (Element element : childElementList(rootElement, "menu")) {
    ModelMenu menu = new ModelMenu(element, location, theme);
    modelMenuMap.put(menu.getName(), menu);
}
```

```
Map<String, ModelMenu> modelMenuMap = childElementList(rootElement, "menu").stream()
    .map(element -> new ModelMenu(element, location, theme))
    .collect(Collectors.toMap(
            ModelWidget::getName, // how key is generated
            Function.identity(), // how value is generated
            (a, b) -> b)); // how duplicates are resolved
```

Now it's your turn

- **Hyperlinks in RefactoringTasks**
  - task11_replaceWithStreamCollectToMap()

```
CheckOutHelper.makeBillingAccountMap()
ComponentConfig.ComponentConfig()
MenuFactory.readMenuDocument()
ModelScreenWidget.DecoratorScreen.DecoratorScreen()

Bonus:
UtilMisc.LocaleHolder.getAvailableLocaleList()
```

JavaSpecialists

12. Stream .reduce()

12. Stream.reduce()

◉ **Can merge all values into one with reduce()**

```
BigDecimal total = BigDecimal.ZERO;
for (String value : amountMap.values()) {
    if (UtilValidate.isNotEmpty(value)) {
        total = total.add(new BigDecimal(value));
    }
}
return total;
```

12. Stream.reduce()

- **Can merge all values into one with reduce()**

```
BigDecimal total = BigDecimal.ZERO;
for (String value : amountMap.values()) {
    if (UtilValidate.isNotEmpty(value)) {
        total = total.add(new BigDecimal(value));
    }
}
return total;
```

```
return amountMap.values().stream()
        .filter(UtilValidate::isNotEmpty)
        .map(BigDecimal::new)
        .reduce(BigDecimal.ZERO, BigDecimal::add);
```

12. Stream.reduce()

- Can merge all values into one with reduce()

```
BigDecimal total = BigDecimal.ZERO;
for (String value : amountMap.values()) {
    if (UtilValidate.isNotEmpty(value)) {
        total = total.add(new BigDecimal(value));
    }
}
return total;
```

```
return amountMap.values().stream()
        .filter(UtilValidate::isNotEmpty)
        .map(BigDecimal::new)
        .reduce(BigDecimal.ZERO, BigDecimal::add);
```

12. Stream.reduce()

- **Can merge all values into one with reduce()**

```
BigDecimal total = BigDecimal.ZERO;
for (String value : amountMap.values()) {
    if (UtilValidate.isNotEmpty(value)) {
        total = total.add(new BigDecimal(value));
    }
}
return total;
```

```
return amountMap.values().stream()
        .filter(UtilValidate::isNotEmpty)
        .map(BigDecimal::new)
        .reduce(BigDecimal.ZERO, BigDecimal::add);
```

12. Stream.reduce()

- Can merge all values into one with reduce()

```
BigDecimal total = BigDecimal.ZERO;
for (String value : amountMap.values()) {
    if (UtilValidate.isNotEmpty(value)) {
        total = total.add(new BigDecimal(value));
    }
}
return total;
```

```
return amountMap.values().stream()
        .filter(UtilValidate::isNotEmpty)
        .map(BigDecimal::new)
        .reduce(BigDecimal.ZERO, BigDecimal::add);
```

12. Stream.reduce()

- Can merge all values into one with reduce()

```
BigDecimal total = BigDecimal.ZERO;
for (String value : amountMap.values()) {
    if (UtilValidate.isNotEmpty(value)) {
        total = total.add(new BigDecimal(value));
    }
}
return total;
```

```
return amountMap.values().stream()
        .filter(UtilValidate::isNotEmpty)
        .map(BigDecimal::new)
        .reduce(BigDecimal.ZERO, BigDecimal::add);
```

12. Stream.reduce()

- **Can merge all values into one with reduce()**

```java
BigDecimal total = BigDecimal.ZERO;
for (String value : amountMap.values()) {
    if (UtilValidate.isNotEmpty(value)) {
        total = total.add(new BigDecimal(value));
    }
}
return total;
```

```java
return amountMap.values().stream()
        .filter(UtilValidate::isNotEmpty)
        .map(BigDecimal::new)
        .reduce(BigDecimal.ZERO, BigDecimal::add);
```

Now it's your turn

- **Hyperlinks in RefactoringTasks**
  - task12_replaceWithReduce()

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

JavaSpecialists

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;
```

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()))
        .flatMap(cc -> cc.getClasspathInfos().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()))
        .flatMap(cc -> cc.getClasspathInfos().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()))
        .flatMap(cc -> cc.getClasspathInfos().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()))
        .flatMap(cc -> cc.getClasspathInfos().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()))
        .flatMap(cc -> cc.getClasspathInfos().stream())
        .collect(Collectors.toList());
```

Now it's your turn

- **Hyperlinks in RefactoringTasks**
  - 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;
```

Returning an Optional from Stream

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

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

```java
return keyMaps.stream()
        .filter(keyMap -> keyMap.getFieldName().equals(fieldName))
        .findFirst()
        .orElse(null);
```

Returning an Optional from Stream

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

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

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

Now it's your turn

- **Hyperlinks in RefactoringTasks**
  - task14_replaceFindFirstOrAny()

```
ModelRelation.findKeyMap()
ModelRelation.findKeyMapByRelated()
ShoppingCartItem.updatePrice()
OrderReadHelper.getShippableSizes()
```

JavaSpecialists

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>>**

```java
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);
```

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

Strings sorted alphabetically

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);
```

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

Map is now a TreeMap

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,
                        Collectors.toCollection(
                                HashSet::new))));
System.out.println(map.getClass());
map.entrySet().forEach(System.out::println);
```

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

Strings are upper case

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()
        )
);
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 RefactoringTasks**
  - task15_replaceWithCollectGroupingByMapping()

```
ModelReader.rebuildResourceHandlerEntities()
```

JavaSpecialists

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;
}
```

Throwings Facade

```java
public class Throwings {
    private Throwings() {}
    public static <T> ThrowingConsumer<T> consumer(
            ThrowingConsumer<T> c) {return c;}
    public static <T, R> ThrowingFunction<T, R> function(
            ThrowingFunction<T, R> f) {return f;}
    public static <T> ThrowingPredicate<T> predicate(
            ThrowingPredicate<T> p) {return p;}
    public static <T> ThrowingSupplier<T> supplier(
            ThrowingSupplier<T> s) {return s;}
}
```

Transforming with Exceptions

- Casting lambda to a ThrowingFunction

```
List<GenericValue> result = new LinkedList<>();
for (GenericValue value : values) {
    result.addAll(value.getRelated(
            relationName, fields, null, useCache));
}
return result;
```

Transforming with Exceptions

- Casting lambda to a ThrowingFunction

```
List<GenericValue> result = new LinkedList<>();
for (GenericValue value : values) {
    result.addAll(value.getRelated(
            relationName, fields, null, useCache));
}
return result;
```

⬇

```
return values.stream()
        .flatMap(Throwings.function(
                paymentPref -> paymentPref.getRelated(
                        relationName, fields, null, useCache).stream()))
        .collect(Collectors.toCollection(LinkedList::new));
```

Now it's your turn

- **Hyperlinks in RefactoringTasks**
  - task16_checkedExceptions

```
EntityUtil.getRelated()
ModelReader.getEntitiesByPackage()
```

JavaSpecialists

17. Performance

17. Performance

- **Streams meant to make logic more understandable**
  - There can be an initial overhead setting up the pipeline
  - When streams are large, performance is similar to loops
    - But when streams are very small or empty, factors faster
    - Start method with `if (list.isEmpty()) return;`
- **Streams make parallelism easy**
  - However, each task should do at least 10 000 instructions
  - A parallel stream is split into ≈ 4 x hardware threads tasks
    - e.g. on my 1-8-2 laptop, we will have 4 x 16 = 64 tasks
  - Thus we need to do at least 640 000 instructions
    - Otherwise the cost of setting it up will be more than benefit

JavaSpecialists

Conclusion

Conclusion

- **Where to next?**
  - Join The Java Specialists' Newsletter
    - www.javaspecialists.eu
  - Mastering Lambdas - Maurice Naftalin
  - www.lambdafaq.org
  - Practice, practice, practice
    - Use Analyze -> Inspect to find more places to refactor
  - Do task99_forTheSuperKeen()